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 «Игра 2048»**

БГУИР КП 1-40 01 01 009 ПЗ

Студент Бычковский П.В.

Руководитель Данилова Г.В.

Минск 2024

СОДЕРЖАНИЕ

[Введение 5](#_Toc162258663)

[1 Анализ предметной области 6](#_Toc162258664)

[1.1 Обзор аналогов 6](#_Toc162258665)

[1.2 Постановка задачи 8](#_Toc162258666)

[2 Проектирование программного средства 9](#_Toc162258667)

[2.1 Структура программы 9](#_Toc162258668)

[2.2 Проектирование интерфейса программного средства 9](#_Toc162258669)

[2.3 Проектирование функционала программного средства 12](#_Toc162258674)

[3 Разработка программного средства 18](#_Toc162258680)

[3.1 Игровой процесс 18](#_Toc162258681)

[3.2 Динамические структуры данных 21](#_Toc162258686)

[3.3 Визуализация игровых процессов 23](#_Toc162258690)

[3.4 Работа с внешними источниками 25](#_Toc162258692)

[4 Тестирование программного средства 28](#_Toc162258698)

[5 Руководство пользователя 30](#_Toc162258699)

[5.1 Интерфейс программного средства 30](#_Toc162258700)

[Заключение 34](#_Toc162258708)

[Список использованных источников 35](#_Toc162258709)

[Приложение А. Исходный код 36](#_Toc162258710)

# ВВЕДЕНИЕ

В настоящее время игры являются неотъемлемой частью нашей повседневной жизни. Они не только предоставляют развлечение и отдых, но и оказывают значительное влияние на различные аспекты нашего мышления, включая логическое мышление, стратегическое планирование и принятие решений. С появлением современных технологий и развитием игровой индустрии игры стали доступны на различных платформах, включая компьютеры, мобильные устройства и консоли, что привело к их всеобщей популярности и разнообразию.

Одной из самых захватывающих и популярных игр, которая получила широкое признание в последние годы, является игра «2048».

История игры уходит своими корнями в 2014 год, когда итальянский разработчик Габриэле Чирулли создал эту игру в качестве тренировочного проекта.

Цель игры заключается в том, чтобы объединять плитки с одинаковыми числами, суммируя их и создавая все более крупные числа. Игра продолжается до тех пор, пока игрок не сможет получить число «2048» на одной из плиток сетки. Если на игровом полотне не осталось свободных ячеек или возможности для объединения полей, игра завершается.

Привлекательность игры «2048» заключается в ее простоте и одновременно сложности. Она требует от игрока не только умения перемещать плитки по сетке, но и разработки стратегии, чтобы достичь максимального числа. Каждый ход в игре требует внимания, анализа и принятия решений в соответствии с текущим состоянием сетки. Эта игра стала настолько популярной, что миллионы людей по всему миру проводят свое свободное время, пытаясь достичь максимального результата.

Разработка игры «2048» является увлекательной и интересной задачей, требующей от разработчика тщательного понимания правил и механики игры, а также умения перевести их в цифровой формат.

Также необходимо создать интуитивно-понятный пользовательский интерфейс, который будет удобен для всех игроков.

Цель разработки программного средства для игры «2048» состоит в создании увлекательной и захватывающей игровой среды, которая позволит игрокам наслаждаться процессом и достигать победного числа.

Разработка такого программного средства является отличной возможностью для улучшения своих навыков в области программирования, алгоритмизации и дизайна пользовательского интерфейса.

# 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.

Популярной в свое время стала веб-игра «2048» от разработчика Veewo Studio. Аудиторию привлек красивый дизайн программы, выполненный в 2D. Стоит отметить, что присутствует плавная анимация передвижения ячеек.

Однако данное программное средство имеет пару недостающих функций: изменение размера игрового поля и сохранения рекорда. Внешний вид данного приложения представлен на рисунке 1.1.

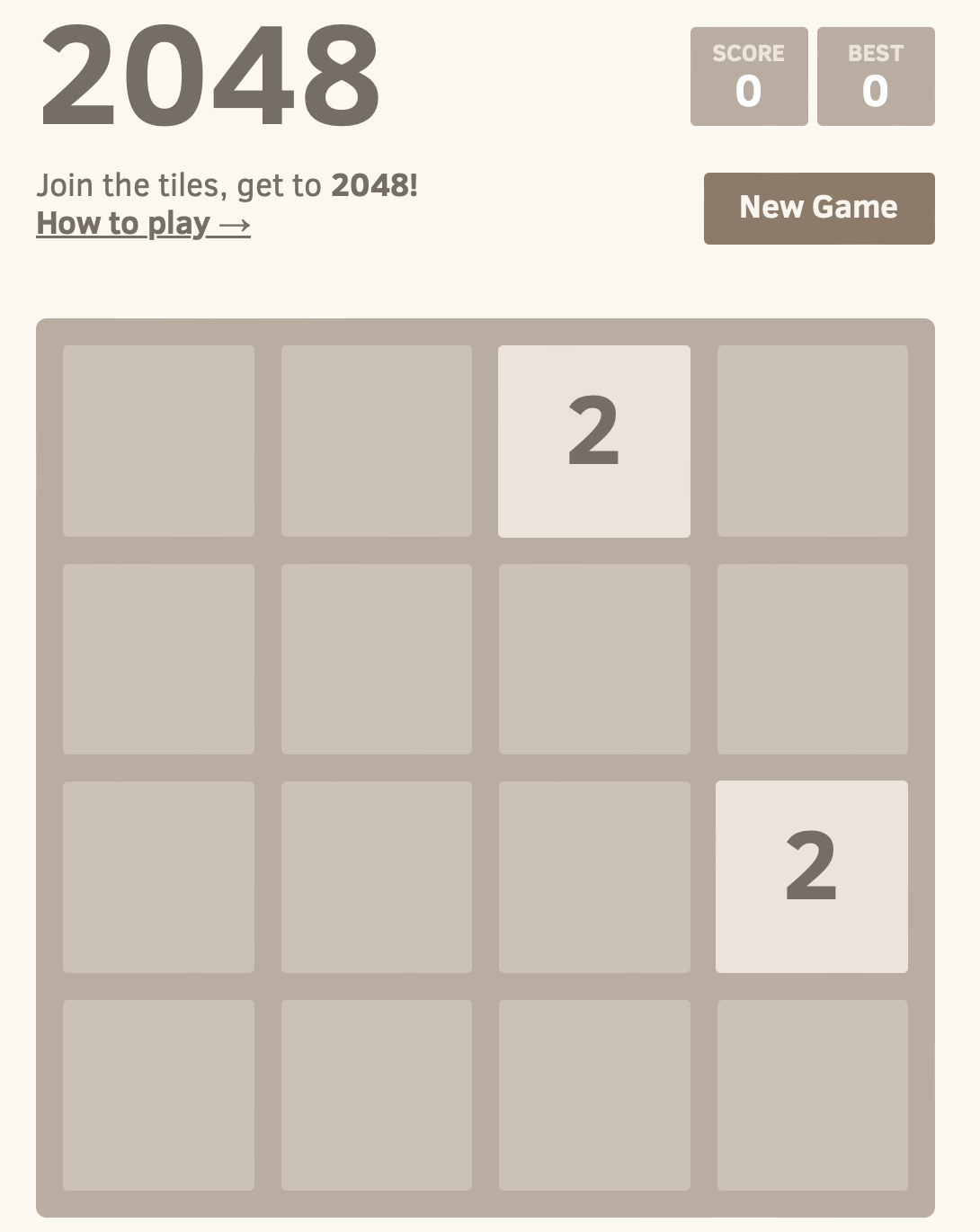


Рисунок 1.1 – Онлайн-игра «2048»

Далее стоит рассмотреть реализацию игры «2048» от разработчика Erkay Uzun. Игра оснащена рейтинговой системой и начальным обучением. Также стоит отметить, что в игре присутствует возможность вернуться на шаг назад.

Одной из приятных особенностей является то, что дизайн выполнен полностью в каноническом для игры «2048» виде. Внешний вид окна данного приложения представлен на рисунке 1.2.

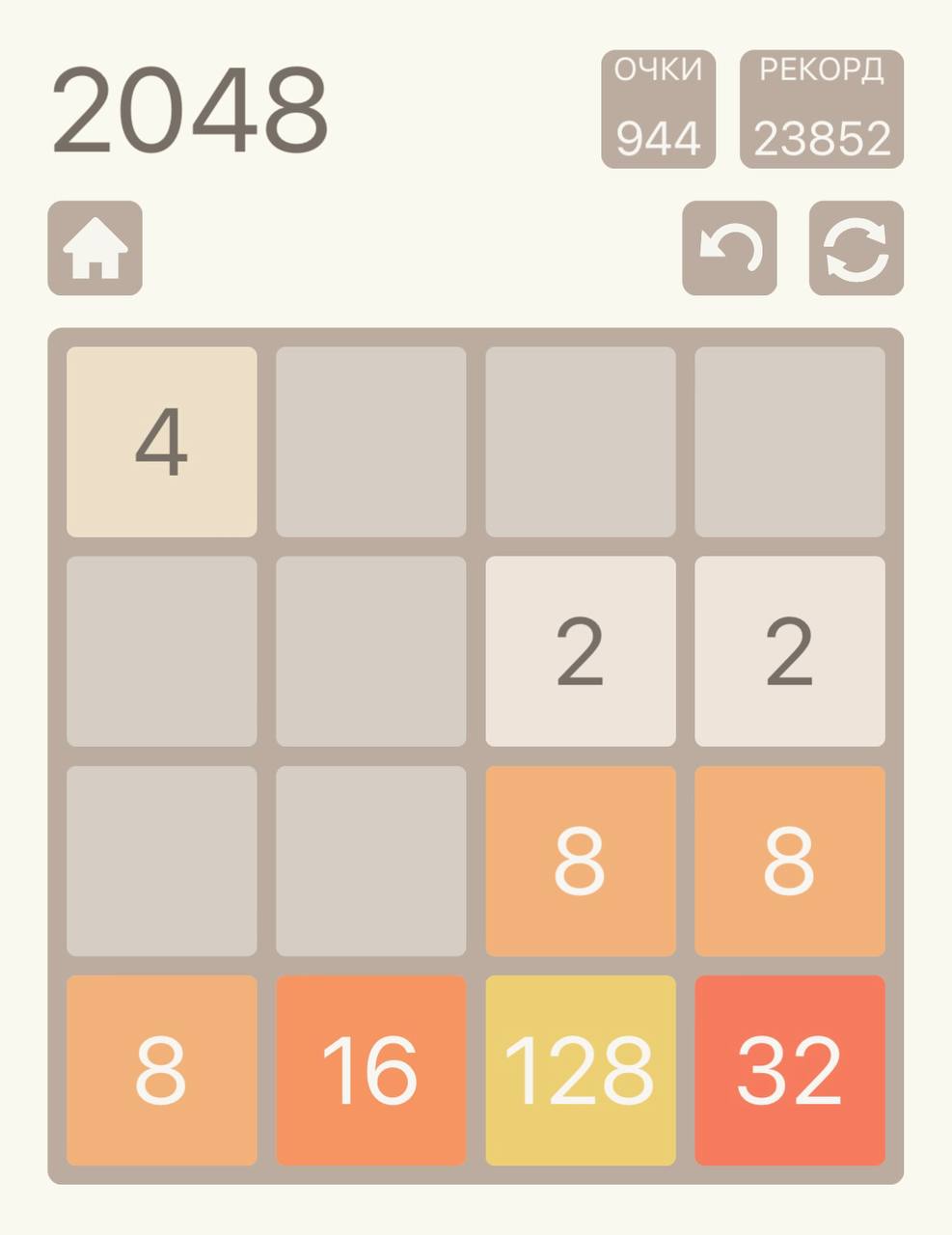


Рисунок 1.2 – Android-приложение «2048»

Не менее популярная версия игры «2048» от разработчика Ketchapp. Игра обладает легким и интуитивно-понятным дизайном. Также есть возможность менять дизайн поля и ячеек в процессе игры. Внешний вид окна данного приложения представлен на рисунке 1.3.

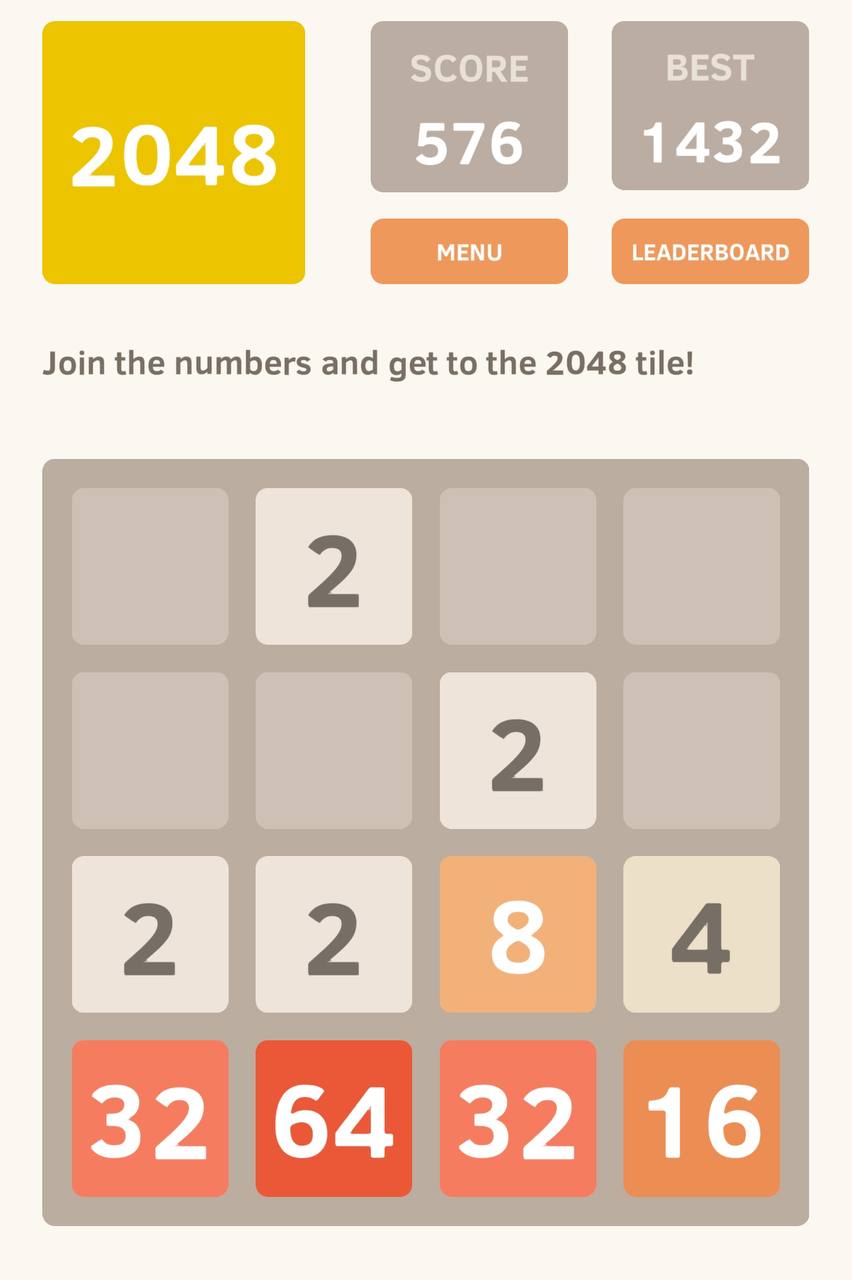


Рисунок 1.3 – iOS-приложение «2048»

## 1.2 Постановка задачи

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

* взаимодействие ячеек;
* передвижение плиток игрового поля;
* анимация передвижения ячеек;
* представление плиток в виде картинок.

Планируется работа с внешними устройствами:

* проверка существования пользователя;
* сохранение результатов в файл;

Планируется добавление такого функционала как:

* подсчет текущего числа очков;
* выбор количества плиток, которое появляется каждый ход;
* динамический размер поля;
* изменение максимального веса плитки, которая может появиться;
* хранение информации о пользователях.

Для разработки программного средства будет использоваться язык программирования Delphi и среда разработки Embarcadero Delphi 11 Community Edition.

# ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

# Структура программы

При разработке программного средства будет использоваться девять модулей:

* LoginUnit – модуль, обеспечивающий отображение меню входа пользователя;
* MainGameUnit – модуль, в котором происходят все основные игровые действия;
* InstructionUnit – модуль, в котором отображаются основные правила игры;
* MenuUnit – модуль, в котором отображаются все настройки игры;
* FieldSetupUnit – модуль, содержащий настройки размера игрового поля;
* SetupAmountUnit – модуль, содержащий настройки количества появляющихся новых ячеек каждый ход;
* SetupRangeUnit – модуль, содержащий настройки диапазона случайных чисел в новых ячейках;
* ContinueChoiceUnit – модуль, который, в случае победы игрока, предлагает ему сыграть в бесконечную игру;
* LoseUnit – модуль, который, в случае поражения игрока, предлагает сыграть новую игру.

# Проектирование интерфейса программного средства

При разработке программного средства за основу будет взят дизайн из классической игры «2048», который был представлен компанией Ketchapp.

# Главное меню

Главное меню приложения будет состоять из кнопки и поля для ввода:

* по центру будет располагаться поле для ввода игрового никнейма пользователя для авторизации;
* ниже располагается кнопка «Играть», при нажатии на которую произойдет проверка на существование пользователя.

Дизайн главного меню приложения представлен на рисунке 2.1.

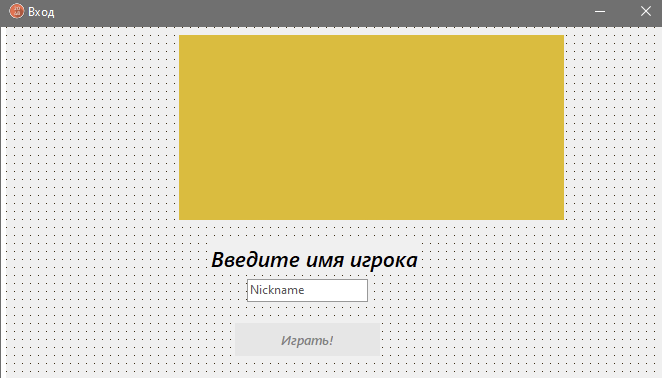
****



Рисунок 2.1 – Окно входа

# Окно главного меню

Окно главного меню будет состоять из трех типов компонентов:

* изображение (TImage), которое будет играть роль заднего фона;
* текст (TLabel), представляющий собой логотип игры;
* кнопки (TButton), которые выполняют определенные действия при нажатии.

Дизайн главного меню представлен на рисунке 2.2.

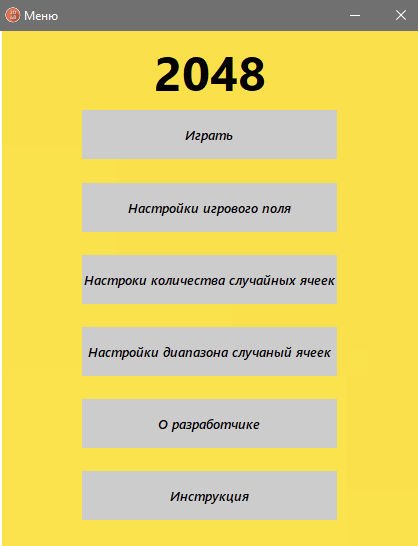


Рисунок 2.2 – Окно главного меню

# Окно игровых правил

Окно игровых правил будет состоять из двух компонентов:

* текст (TLabel), предоставляющий информацию о правилах игры, а также помогающий игроку понять основы игровых механик;
* изображение (TImage), которое будет играть роль заднего фона.

Текст будет оформлен по определенным правилам, чтобы игроку было легче ориентироваться в нем. Дизайн данного окна представлен на рисунке 2.3.

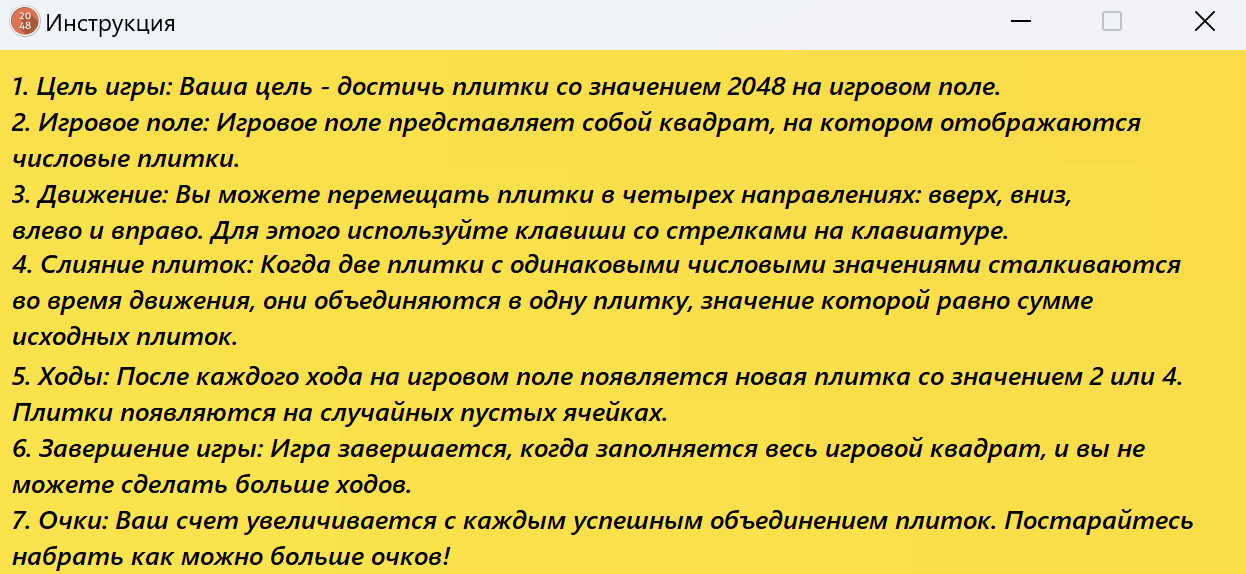
****

Рисунок 2.3 – Окно игровых правил

# Окно проигрыша

Окно проигрыша играет важную роль, потому что информирует игрока о том, что игра для него закончилась и для продолжения надо запустить новую. Данное окно появляется после того, как на игровом поле закончилось место для создания новых ячеек. Дизайн данного окна представлен на рисунке 2.4.

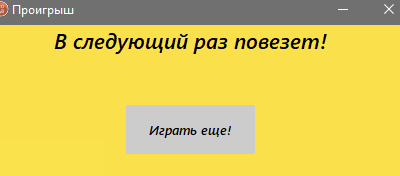


Рисунок 2.4 – Окно проигрыша

* + 1. Окно игрового процесса

Окно игрового процесса будет состоять из четырех частей:

* В верхней части экрана будет бар (TMainMenu) с перечнем кнопок управления полем.
* В правой верхней части экрана будет располагаться информация о текущем количестве очков и рекорде игрока.
* В левом нижнем фрагменте окна будут находиться кнопки (TSpeedButton). Одна из них будет полностью очищать поле, другая будет возвращать состояние поля на шаг назад, третья будет вести в главное меню.
* В правой нижней доле экрана будет находиться изображение (TImage) игрового поля с числами, представляющими собой изображения (TImage).

Дизайн окна игрового процесса представлен на рисунке 2.5.

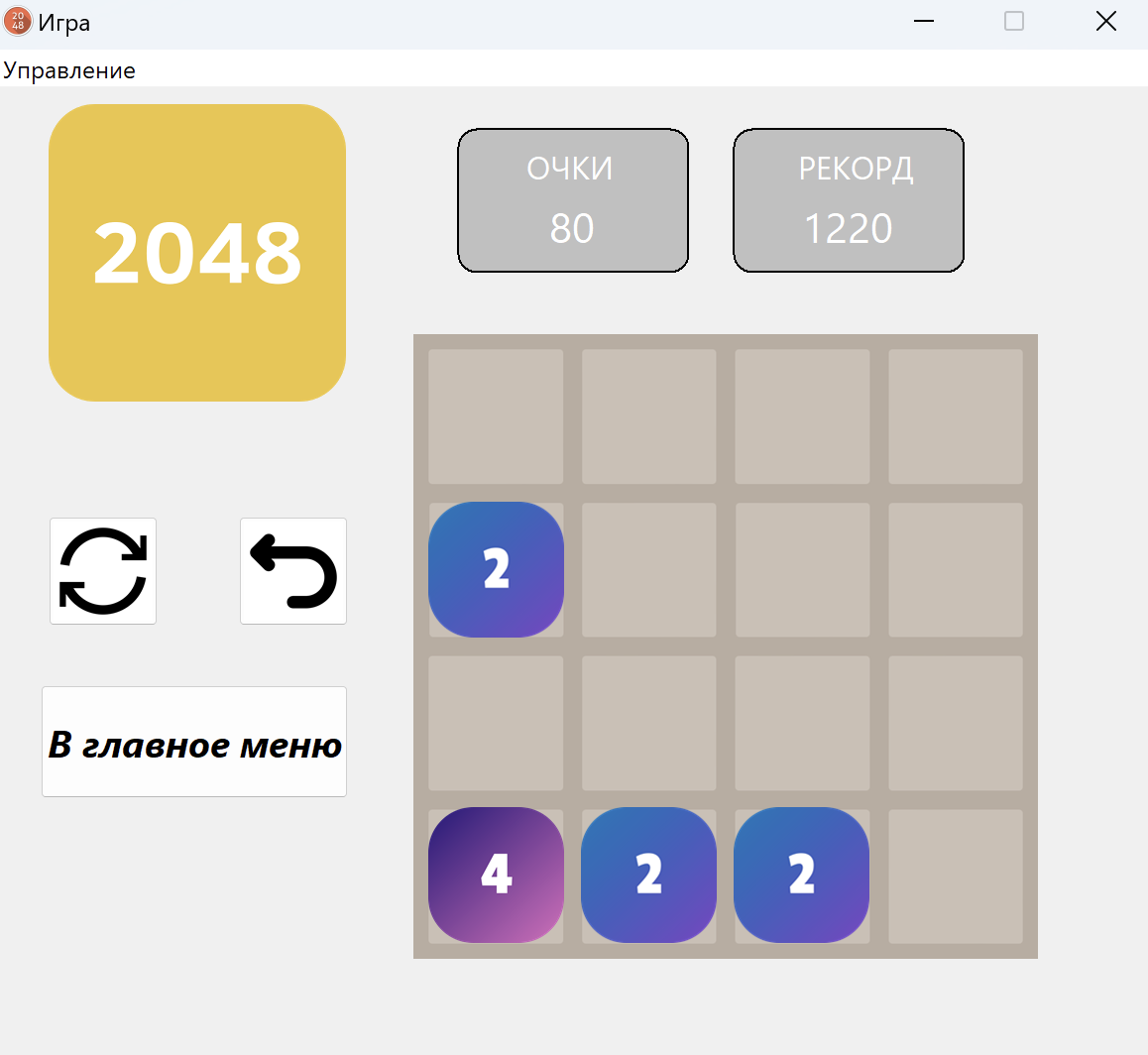


Рисунок 2.5 – Окно игрового процесса

# Проектирование функционала программного средства

Разработка алгоритмов является ключевой фазой в проектировании программного средства. Игровое приложение «2048» должно предоставлять пользователю такой минимальный функционал как:

* генерация случайных ячеек;
* создание изображений для ячеек с числами;
* передвижение ячеек;
* подсчет текущего количества очков;
* анимация передвижения ячеек поля.

# Генерация случайных ячеек

Одна из основных механик этой игры заключается в генерации чисел и их размещении на игровом поле. Этот процесс реализуется в процедуре FillRandomNumber, которая находится внутри модуля MainGameUnit. В этой процедуре генерируется случайное число для пустой ячейки на поле. Сначала выбирается случайная ячейка на поле, а затем генерируется случайное значение для плитки, которое является степенью числа два. Блок-схема алгоритма данного метода приведена на рисунке 2.6.

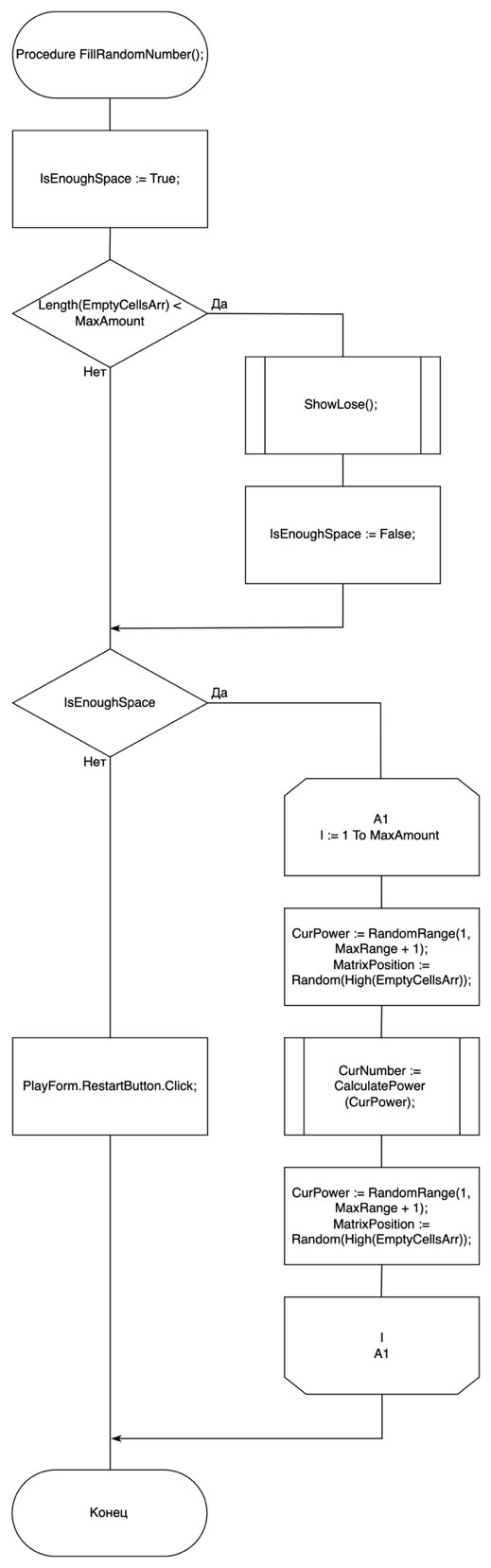


Рисунок 2.6 – Блок-схема процедуры FillRandomNumber

# Создание изображений ячеек с числами

В модуле MainGameUnit осуществляется отображение ячеек на игровом поле. Игровое поле представляет собой двумерный массив, содержащий изображения (объекты TImage) для каждой ячейки с числом. Для создания и инициализации изображений была реализована процедура GetStartImages. В этой процедуре динамически создаются объекты TImage с определенными параметрами и устанавливается соответствующее изображение. Затем созданное изображение добавляется в двумерный массив, который содержит все изображения для ячеек поля. Блок-схема алгоритма приведена на рисунке 2.7.

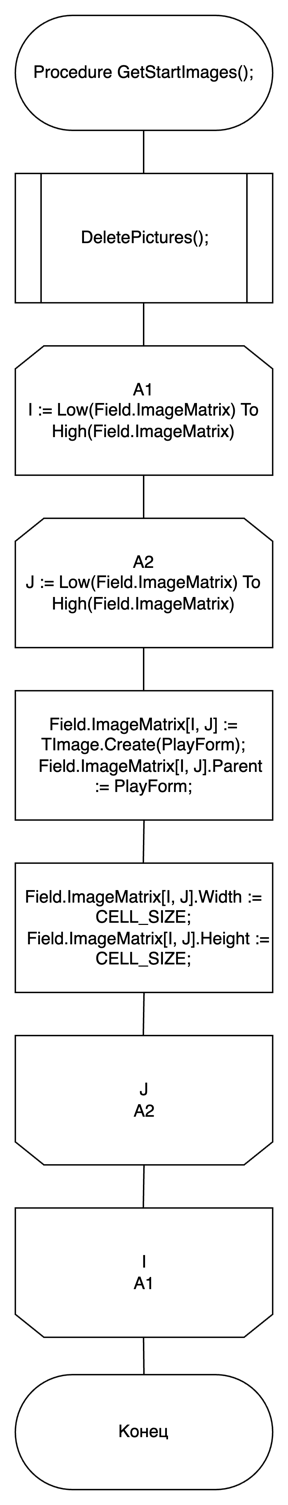


Рисунок 2.7 – Блок-схема процедуры GetStartImages

# Передвижение ячеек

Важной частью этого программного средства является алгоритм перемещения ячеек на игровом поле, который описан в процедуре FieldMove. В начале алгоритма текущее состояние поля сохраняется в стеке, чтобы иметь возможность откатить изменения. Затем в зависимости от выбранного направления сдвига происходит вызов других подпрограмм для фактического изменения игрового поля. Алгоритм процедуры FieldMove представлен в виде блок-схемы на рисунке 2.8.

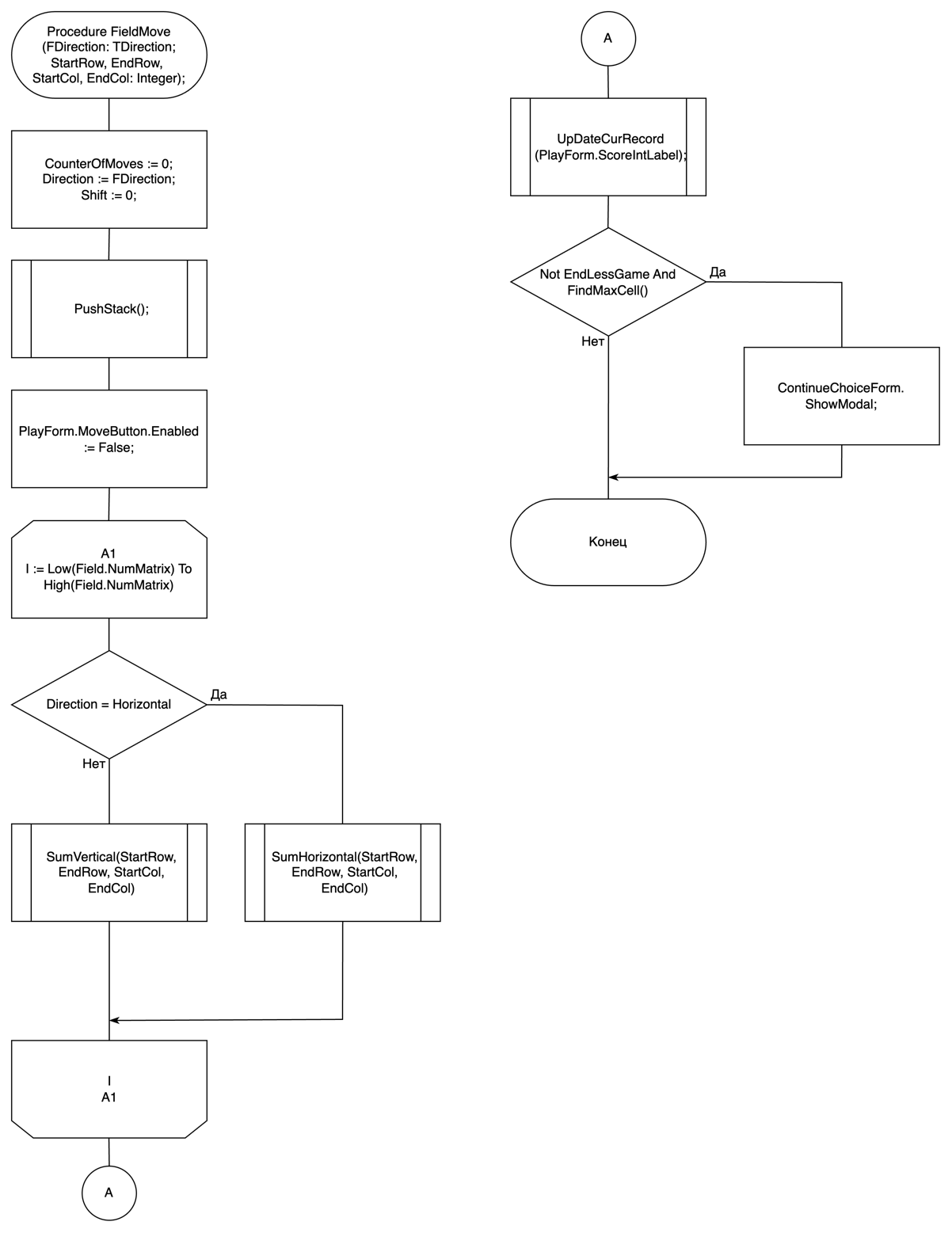


Рисунок 2.8 – Блок-схема метода FieldMove

# Подсчет текущего количества очков

Алгоритм CalculatePoints играет важную роль в игре, поскольку он позволяет игрокам устанавливать новые рекорды и стимулирует их соревноваться друг с другом. Этот алгоритм достаточно прост, но приносит большой интерес в игру. Он заключается в обходе всех ячеек на поле и подсчете их суммы. Благодаря этому алгоритму игроки могут увеличивать свой счет и стремиться к достижению максимально возможной суммы. Блок-схема алгоритма CalculatePoints представлена на рисунке 2.9.

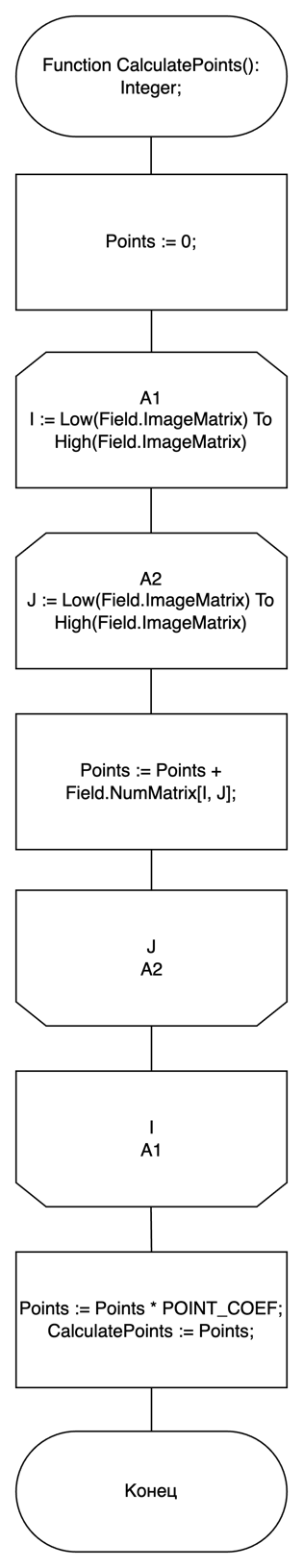


Рисунок 2.9 – Блок-схема процедуры CalculatePoints

# Анимация передвижения ячеек поля

Пользовательский интерфейс игры является одним из ключевых аспектов, поскольку он играет важную роль в привлечении новых игроков. В этом контексте особое значение имеет алгоритм AnimationTimer**.** Этот алгоритм обеспечивает плавные анимационные эффекты при перемещении ячеек, что делает игру более привлекательной для игроков. Блок-схема алгоритма AnimationTimer представлена на рисунке 2.10.

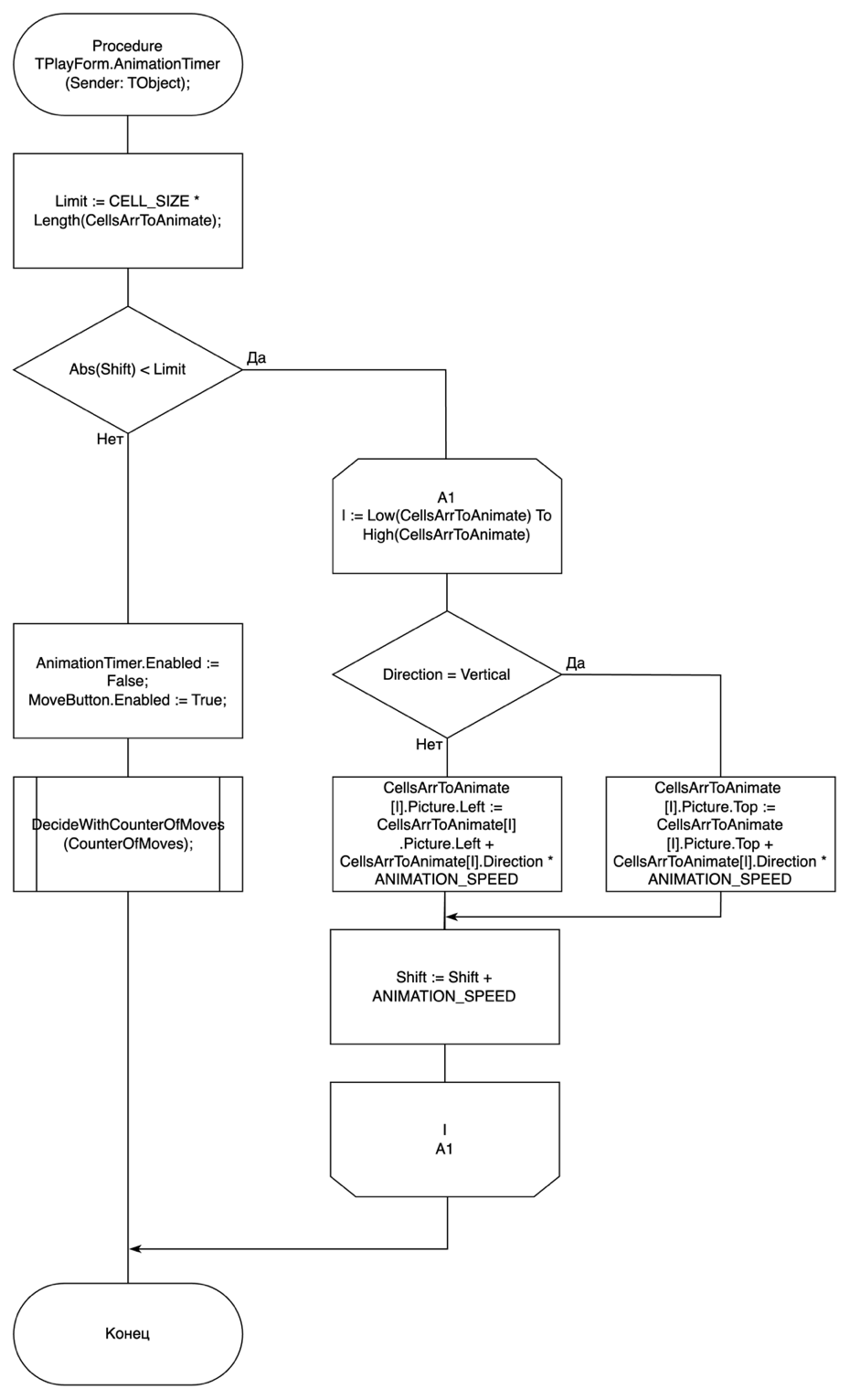


Рисунок 2.10 – Блок-схема процедуры AnimationTimer

Данные подпрограммы играют важную роль в игровом процессе и качестве пользовательского опыта.

1. **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**
   1. **Игровой процесс**

При создании данного программного средства основное внимание уделялось разработке игровой логики и взаимодействию с пользователем. Для реализации игровых механик были использованы соответствующие подпрограммы. Эти функции выполняют различные задачи, такие как перемещение ячеек поля, обработка столкновений плиток, генерация новых элементов. Они взаимодействуют между собой и с другими компонентами программы, чтобы обеспечить правильное функционирование игры и достижения желаемых игровых механик.

* + 1. Начало игры

В данном программном средстве была добавлена возможность начать игру после нажатия на кнопку «Играть». Логика этой кнопки реализована в процедуре PlayButtonClick. Ниже приведен пример кода данного метода.

Procedure TSetupForm.PlayButtonClick(Sender: TObject);

Begin

// ищем пользователя в истории

FindUserInHistory();

// если нашли, то правильно располагаем его предыдущий рекорд

GetCorrectPlaceForRecord();

// убираем форму настроек

SetupForm.Hide;

// отрисовываем форму игры

PlayForm.Show;

// включаем кнопки управления ходами

PlayForm.MoveButton.Enabled := True;

End;

* + 1. Сдвиг ячеек по горизонтали

С помощью стрелочек на клавиатуре влево и вправо можно двигать ячейки поля в горизонтальном направлении. Для этого была реализована процедура SumHorizontal. Код данной подпрограммы приведен ниже.

Procedure SumHorizontal(StartCol, EndCol: Integer);

Var

I, J, Direction, FindEmpty: Integer;

Begin

Direction := 1;

// если начальный столбец больше конечного

// меняем направление (в нашем случае Direction = -1)

If StartCol > EndCol Then

Direction := -1;

J := StartCol;

While Abs(J - StartCol) < High(Field.NumMatrix) Do

Begin

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

FindEmpty := 0;

// проверка на пустой соседний элемент, а выбранный не равен 0

// если это так, то сдвигаем данный элемент на позицию соседнего

If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I, J + Direction] = 0) Then

Begin

Inc(CounterOfMoves);

// меняем значения плиток местами

Field.NumMatrix[I, J + Direction] := Field.NumMatrix[I, J];

Field.NumMatrix[I, J] := 0;

While (EmptyCellsArr[FindEmpty].MatrixI <> I) Or

(EmptyCellsArr[FindEmpty].MatrixJ <> J + Direction) Do

Inc(FindEmpty);

// Добавляем в массив пустых ячеек плитку, которая теперь стала равна

// нулю

EmptyCellsArr[FindEmpty].MatrixJ := J;

// Добавляем элеленты которые меняем местами в массив ячеек, которые

// нужно анимировать

AddToAnimationArr(Field.ImageMatrix[I, J], Direction);

AddToAnimationArr(Field.ImageMatrix[I, J + Direction],

-Direction);

// Меняем изображения местами

SwapPictures(0, Direction, I, J);

End

// проверка на равные соседние ячейки

Else If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I, J + Direction] = Field.NumMatrix[I, J]) Then

Begin

Inc(CounterOFMoves);

// значение одной ячейки умножаем на 2, другой присваиваем 0

Field.NumMatrix[I, J + Direction] := Field.NumMatrix[I, J] \* 2;

Field.NumMatrix[I, J] := 0;

// меняем иконки ячеек на актуальные

SwitchCellsIcon(Field.ImageMatrix[I, J],

Field.ImageMatrix[I, J + Direction],

Field.NumMatrix[I, J + Direction]);

// меняем изображения местами

SwapPictures(0, Direction, I, J);

// добавляем пустую ячейку в массив пустых ячеек

AddToEmptyCellsArr(I, J);

// добавляем эти ячейки в массив для анимации ячеек

AddToAnimationArr(Field.ImageMatrix[I, J], -Direction);

AddToAnimationArr(Field.ImageMatrix[I, J + Direction],

Direction);

End;

End;

J := J + Direction;

End;

End;

* + 1. Нахождение максимальной ячейки

В игре главной целью является победа, поэтому была разработана функция FindMaxCell, которая определяет текущую максимальную ячейку на игровом поле. Если значение этой ячейки соответствует победному значению, функция возвращает результат, указывающий на победу игрока. Ниже приведен код данной подпрограммы.

Function FindMaxCell(): Boolean;

Var

IsMaxNum: Boolean;

I, J: Integer;

Begin

IsMaxNum := False;

// запускаем цикл по всем ячейкам

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

// если была найдена ячейка с победным числом, то меняем флаг

If Field.NumMatrix[I,J] = NUMBER\_TO\_WIN Then

IsMaxNum := True;

FindMaxCell := IsMaxNum;

End;

* + 1. Добавление рекорда в файл

Для поддержки соревновательной игры необходимо сохранять рекорды всех игроков, чтобы каждый из них мог улучшать свои результаты с каждой игрой. Поэтому была разработана процедура AddUserToFile, которая анализирует данные в файле и на основе этого записывает новые данные игрока. Ниже приведен код данной подпрограммы.

Procedure AddUserToFile(NickName: String; CurRecord: Integer);

Var

UserFile: TUSerFile;

UserArr: Array Of TUser;

I, Position: Integer;

IsFound, IsExist: Boolean;

Begin

User.NickName := NickName;

User.CurrentRecord := CurRecord;

AssignFile(UserFile, HistoryFilePath);

// проверка файла на существование

IsExist := CheckFileExist(UserFile);

// если файл существует, то читаем всех пользователей в массив

If IsExist Then

Begin

I := 0;

While Not EOF(UserFile) Do

Begin

SetLength(UserArr, I + 1);

Read(UserFile, UserArr[I]);

Inc(I);

End;

End;

Rewrite(UserFile);

Position := 0;

// проверка на то, что пользователь с таким именем есть в файле

IsFound := IsUserInArr(UserArr, NickName, Position);

If IsFound Then

Begin

// если найден, то перезаписываем его данные

If UserArr[Position].CurrentRecord < CurRecord Then

UserArr[Position] := User;

End

Else

Begin

// если не найден, то добавляем нового игрока в массив

SetLength(UserArr, Length(UserArr) + 1);

UserArr[High(UserArr)] := User;

End;

// записываем историю обратно в файл

For I := Low(UserArr) To High(UserArr) Do

Write(UserFile, UserArr[I]);

CloseFile(UserFile);

End;

* 1. **Динамические структуры данных**

Структуры данных играют важную роль при разработке программного средства. В данной программе реализован стек, основанный на двусвязном списке. Эта структура данных позволяет хранить прошлые состояния поля и добавляет новые возможности в приложение. Двусвязный список обеспечивает эффективные операции добавления и удаления элементов. Далее будут рассмотрены подпрограммы, реализованные в данном проекте, для работы со стеком на основе двусвязного списка.

* + 1. Добавление элемента в стек

При работе со стеком одной из ключевых операций является добавление элементов в него. В данной игре часто вызывается процедура PushStack. Эта процедура добавляет текущее состояние игрового поля на вершину стека. Ниже представлен код для подпрограммы PushStack.

Procedure PushStack();

Var

Prev: PStack;

Begin

Prev := Stack;

// выделяем память для нового элемента

New(Stack.Next);

Stack := Stack.Next;

Setlength(Stack.CurField.NumMatrix, FieldSize, FieldSize);

SetLength(Stack.CurField.ImageMatrix, FieldSize, FieldSize);

// присваиваем данному элементы предыдущее состояние игрового поля

CopyField(Field);

Stack.Next := Nil;

Stack.Prev := Prev;

// включаем кнопку возврата хода

PlayForm.BackButton.Enabled := True;

End;

* + 1. Откат хода

Важной процедурой является BackButtonClick, которая позволяет игроку вернуться на предыдущий шаг. В ней реализована функция извлечения предыдущего состояния игрового поля из стека. Этот функционал позволяет пользователю экспериментировать и принимать решения, не беспокоясь о возможных последствиях. Ниже приведен код для данной процедуры.

Procedure TPlayForm.BackButtonClick(Sender: TObject);

Var

I, J: Integer;

Begin

// присваиваем текущему состоянию поля прошлое, которое хранится на вершине стека

Field := Stack.CurField;

Stack := Stack.Prev;

// возвращаем изображения ячеек на стартовые положения

GetStartImages();

GetStartPositions();

// присваиваем изображениям правильные картинки

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

If Field.NumMatrix[I, J] = 0 Then

Field.ImageMatrix[I, J] := Nil

Else

Field.ImageMatrix[I, J].Picture.LoadFromFile

(IntToStr(Field.NumMatrix[I, J]) + IMAGE\_EXTENSION);

End;

// записываем в массив пустые ячейки

GetEmptyCells();

// обновляем количество очков игрока

UpDateCurRecord(ScoreIntLabel);

// если стек пустой, то отключаем кнопку возврата хода

If Stack.Prev = Nil Then

BackButton.Enabled := False;

// включаем кнопки управления полем

MoveButton.Enabled := True;

End;

* + 1. Удаление вершины стека

В программном средстве также присутствует процедура PopStack, которая заслуживает внимания. Ее задачей является удаление верхнего элемента стека. Этот функционал позволяет использовать меньше памяти и оптимизировать работу игры. Ниже приведен код для процедуры PopStack.

Procedure PopStack();

Var

ElementToDelete: PStack;

Begin

// присваиваем во временную переменную адрес элемента,

// который нужно удалить

ElementToDelete := Stack;

// присваиваем указателю на вершину стека

// указатель на элемент перед ним

Stack := Stack.Prev;

Stack.Next := Nil;

// освобождение занятой памяти

Dispose(ElementToDelete);

End;

* 1. **Визуализация игровых процессов**

При разработке игры важным аспектом является визуализация игровых процессов. Интуитивно-понятный и привлекательный интерфейс способен привлечь внимание и улучшить пользовательский опыт. В данной игре были созданы различные графические объекты, такие как два изображения (TImage) и две геометричекие фигуры (TShape). Эти объекты значительно улучшают графическую составляющую игры и способствуют лучшему взаимодействию пользователя с ней:

* BackgroundImage;
* LogoImage;
* ScoreShape;
* RecordShape.

Ниже будут представлены подпрограммы, которые создают приятный глазу пользовательский интерфейс.

* + 1. Отображение значений ячеек на экране

Важной подпрограммой в этом контексте является отображение значений ячеек игрового поля. Для этой цели используется динамическое создание компонентов TImage. Затем изображения чисел игрового поля присваиваются этим компонентам. Задача создания и настройки компонентов TImage выполняется в процедуре GetStartImages. Ниже приведен код данной подпрограммы.

Procedure GetStartImages();

Var

I, J: Integer;

Begin

With PlayForm Do

Begin

// удаляем прошлые изображения

DeletePictures();

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

Begin

// создаем объект картинки

Field.ImageMatrix[I, J] := TImage.Create(PlayForm);

Field.ImageMatrix[I, J].Parent := PlayForm;

// присваиваем размеры картинки

Field.ImageMatrix[I, J].Width := CELL\_SIZE;

Field.ImageMatrix[I, J].Height := CELL\_SIZE;

End;

End;

End;

* + 1. Стартовое размещение изображений ячеек на экране

Необходимо присвоить всем изображениям начальное положение на игровом поле. Для решения этой задачи была разработана процедура GetStartPositions. Код этой подпрограммы представлен ниже.

Procedure GetStartPositions();

Var

I, J: Integer;

Begin

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

Begin

// присваивание координаты верха изображению

Field.ImageMatrix[I, J].Top := START\_TOP + I \*

(Field.ImageMatrix[I, J].Height + OFFSET);

// присваивание координаты левой стороны изображению

Field.ImageMatrix[I, J].Left := START\_LEFT + J \*

(Field.ImageMatrix[I, J].Width + OFFSET);

End;

End;

* + 1. Удаление объектов изображения

Важной процедурой является удаление объектов изображений, которые больше не используются, чтобы разгрузить ресурсы компьютера. Для этого была разработана подпрограмма DeletePictures. Ниже приведен код этой процедуры.

Procedure DeletePictures();

Var

I, J: Integer;

Begin

// обход всех объектов

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

// проверка существования изображения

If Field.ImageMatrix[I, J] <> Nil Then

// освобождение памяти, занятой объектом

Field.ImageMatrix[I, J].Free;

End;

* + 1. Анимация передвижения плиток

Одной из наиболее важных подпрограмм для пользовательского интерфейса является анимация передвижения ячеек. Благодаря этой анимации плитки плавно перемещаются по игровому полю, что делает игру по-настоящему привлекательной. Для реализации этой анимации была разработана процедура AnimationTimer. Ниже приведен код этой подпрограммы.

Procedure TPlayForm.AnimationTimer(Sender: TObject);

Var

Limit, I: Integer;

Begin

// расчет сдвига для всех ячеек

Limit := CELL\_SIZE \* Length(CellsArrToAnimate);

// условие конца анимации

If Shift < Limit Then

// обход всех ячеек которые нужно анимировать

For I := Low(CellsArrToAnimate) To High(CellsArrToAnimate) Do

Begin

// проверка на направление анимации

If Direction = Vertical Then

// анимация по вертикали

CellsArrToAnimate[I].Picture.Top := CellsArrToAnimate[I]

.Picture.Top + CellsArrToAnimate[I].Direction \* ANIMATION\_SPEED

Else

// анимация по горизонтали

CellsArrToAnimate[I].Picture.Left := CellsArrToAnimate[I]

.Picture.Left + CellsArrToAnimate[I].Direction \* ANIMATION\_SPEED;

// изменение текущего сдвига

Shift := Shift + ANIMATION\_SPEED;

End

Else

Begin

// выключение таймера для анимации

Animation.Enabled := False;

// очистка массива ячеек для анимации

Setlength(CellsArrToAnimate, 0);

// проверка на проигрыш

DecideWithCounterOfMoves(CounterOfMoves);

// включение кнопок управления полем

MoveButton.Enabled := True;

End;

End;

* 1. **Работа с внешними источниками**

Работа с типизированными файлами в программном средстве игры «2048» позволяет обеспечить хранение рекордов игроков и создать соревновательный дух в игре. Типизированные файлы используются для сохранения информации о пользователях, таких как их имена и текущие рекорды.

Когда игрок достигает нового рекорда, его имя и соответствующий рекорд записываются в файл. Это позволяет игрокам сохранять свои достижения даже после выключения программы и повторного запуска.

Соревновательный дух в игре поддерживается путем сохранения рекордов всех игроков. Игроки могут стремиться улучшить свои результаты и попытаться побить рекорды других игроков.

Таким образом, внешние источники стимулируют игроков к достижению лучших результатов и улучшению своих навыков в игре.

* + 1. Запись пользователей

Процедура AddUserToFile является важной частью функционала сохранения пользовательской информации в игре «2048». Она позволяет добавлять новых игроков в типизированный файл, чтобы сохранять их имена и текущие рекорды для последующего использования. Код данной подпрограммы приведен ниже.

Procedure AddUserToFile(NickName: String; CurRecord: Integer);

Var

UserFile: TUSerFile;

UserArr: Array Of TUser;

I, Position: Integer;

IsFound, IsExist: Boolean;

Begin

User.NickName := NickName;

User.CurrentRecord := CurRecord;

AssignFile(UserFile, HistoryFilePath);

// проверка файла на существование

IsExist := CheckFileExist(UserFile);

If IsExist Then

Begin

// если файл существует

// то читаем всех пользователей в массив

I := 0;

While Not EOF(UserFile) Do

Begin

SetLength(UserArr, I + 1);

Read(UserFile, UserArr[I]);

Inc(I);

End;

End;

Rewrite(UserFile);

Position := 0;

// проверка существования пользователя с таким именем в файле

IsFound := IsUserInArr(UserArr, NickName, Position);

If IsFound Then

Begin

// если найден

// то перезаписываем его данные

If UserArr[Position].CurrentRecord < CurRecord Then

UserArr[Position] := User;

End

Else

Begin

// если не найден

// то добавляется новый игрок в массив

SetLength(UserArr, Length(UserArr) + 1);

UserArr[High(UserArr)] := User;

End;

// записываем историю обратно в файл

For I := Low(UserArr) To High(UserArr) Do

Write(UserFile, UserArr[I]);

CloseFile(UserFile);

End;

* + 1. Чтение информации пользователей

Одной из первых подпрограмм, которая вызывается в данной игре является процедура FindUserInHistory. Главной задачей этого метода является выгрузка информации о пользователях из типизированного файла. Если в файле был найден пользователь с таким же именем, то его рекорд будет использоваться в игре. Код процедуры FindUserInHistory представлен ниже.

Procedure FindUserInHistory();

Var

UserFile: TUserFile;

FindUser: TUser;

NickNameToFind: String;

IsFileExist: Boolean;

Begin

IsFileExist := True;

NickNameToFind := LoginForm.NickNameEdit.Text;

AssignFile(UserFile, HistoryFilePath);

// проверка на существование файла

IsFileExist := CheckFileExist(UserFile);

If IsFileExist Then

Begin

// чтение файла

While (Not EOF(UserFile)) And (FindUser.NickName <> NickNameToFind) Do

Read(UserFile, FindUser);

// если пользователь не был найден

// то его рекорд будет равен 0

If (FindUser.NickName <> NickNameToFind) Or Not

CheckRange(FindUser.CurrentRecord) Then

PlayForm.RecordIntLabel.Caption := '0'

Else

// если был найден

// то его рекорд будет из файла

PlayForm.RecordIntLabel.Caption := IntToStr(FindUser.CurrentRecord);

CloseFile(UserFile);

End;

End;

Главным достоинством данных подпрограмм является то, что они отслеживают различные ошибки и обрабатывают их. Тем самым пользователь может быть уверен, что программа будет работать без сбоев, даже если данные будут случайным образом потеряны или испорчены.

1. **ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

В ходе тестирования игры было обнаружено, что кнопку «Вернуться на шаг назад» на игровом экране можно было нажимать даже в тех случаях, когда не было предыдущих состояний поля. Это приводило к ошибке Access Violation. Проблема иллюстрируется на рисунке 4.1.

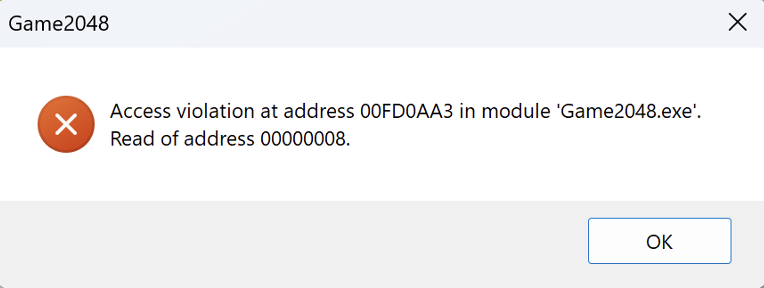


Рисунок 4.1 – Ошибка access violation

Для решение данной проблемы было решено изменить подпрограмму. В процедуре BackButtonClick была добавлена проверка того, что текущий стек не является пустым. Код исправленной подпрограммы находится ниже.

Procedure TPlayForm.BackButtonClick(Sender: TObject);

Var

I, J: Integer;

Begin

Field := Stack.CurField;

Stack := Stack.Prev;

GetStartImages();

GetStartPositions();

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

If Field.NumMatrix[I, J] = 0 Then

Field.ImageMatrix[I, J].Picture := Nil

Else

Field.ImageMatrix[I, J].Picture.LoadFromFile

(IntToStr(Field.NumMatrix[I, J]) + IMAGE\_EXTENSION);

End;

GetEmptyCells();

UpDateCurRecord(ScoreIntLabel);

// выключить кнопку если стек пустой

If Stack.Prev = Nil Then

BackButton.Enabled := False;

MoveButton.Enabled := True;

End;

Теперь процедура проверяет стек на пустоту. Если стек пустой, то кнопка отката шага становится недоступной.

Было обнаружено, что если игрок нажмет кнопку передвижения ячеек во время анимации, то местоположение плиток станет некорректным до конца игры. Эта проблема иллюстрируется на рисунке 4.2.

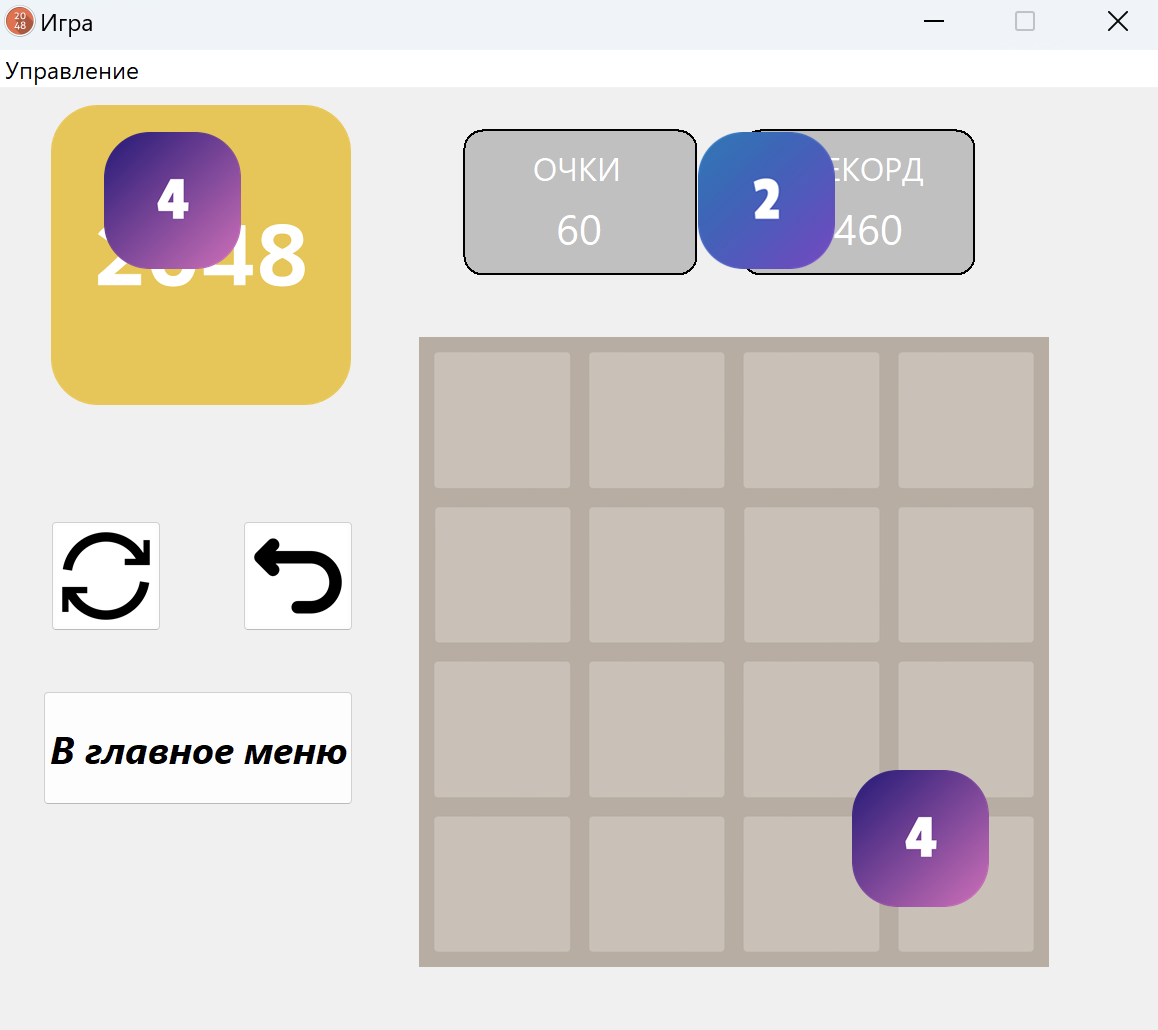


Рисунок 4.2 – Некорректное отображение плиток

Для решения этой проблемы было принято запрещать нажатия клавиш управления игровым полем во время анимаций. Исправления были внесены в процедуру FieldMove, так как именно она отвечает за сдвиги ячеек. Ниже представлен код исправленной подпрограммы.

Procedure FieldMove(FDirection: TDirection; StartRow, EndRow,

StartCol, EndCol: Integer);

Var

I: Integer;

Begin

CounterOfMoves := 0;

PushStack();

Direction := FDirection;

Shift := 0;

// блокируется нажатие кнопок управления

PlayForm.MoveButton.Enabled := False;

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

If Direction = Horizontal Then

SumHorizontal(StartRow, EndRow, StartCol, EndCol)

Else

SumVertical(StartRow, EndRow, StartCol, EndCol);

PlayForm.AnimationTimer.Enabled := True;

UpDateCurRecord(PlayForm.ScoreIntLabel);

If Not EndLessGame And FindMaxCell() Then

ContinueChoiceForm.ShowModal;

End;

В результате тестирования было обнаружено, что программное средство для игры «2048» работает корректно и соответствует заданным требованиям. Все основные функции игры были успешно протестированы.

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

* 1. **Интерфейс программного средства**

Интерфейс программного средства для игры «2048» предоставляет пользователю средства взаимодействия с игрой. Он включает в себя главное меню, игровое поле, плитки с числовыми значениями, элементы управления и отображение текущего состояния игры, такого как счет и сообщения о завершении игры. Пользователь может использовать элементы управления, чтобы передвигать плитки по игровому полю и объединять их. Интерфейс также включает в себя настройки игры, позволяющие пользователю настроить различные параметры игры.

* + 1. Экран входа

Окно входа состоит из трех компонентов:

* Первым располагается поле для ввода игрового никнейма.
* Вторым располагается кнопка «Играть!», нажав на которую, игрок будет перенаправлен на экран настроек.
* Третьим является кнопка со значком вопроса. Она вызывает показ подсказки для пользователя.

Внешний вид главного меню приложения представлен на рисунке 5.1.

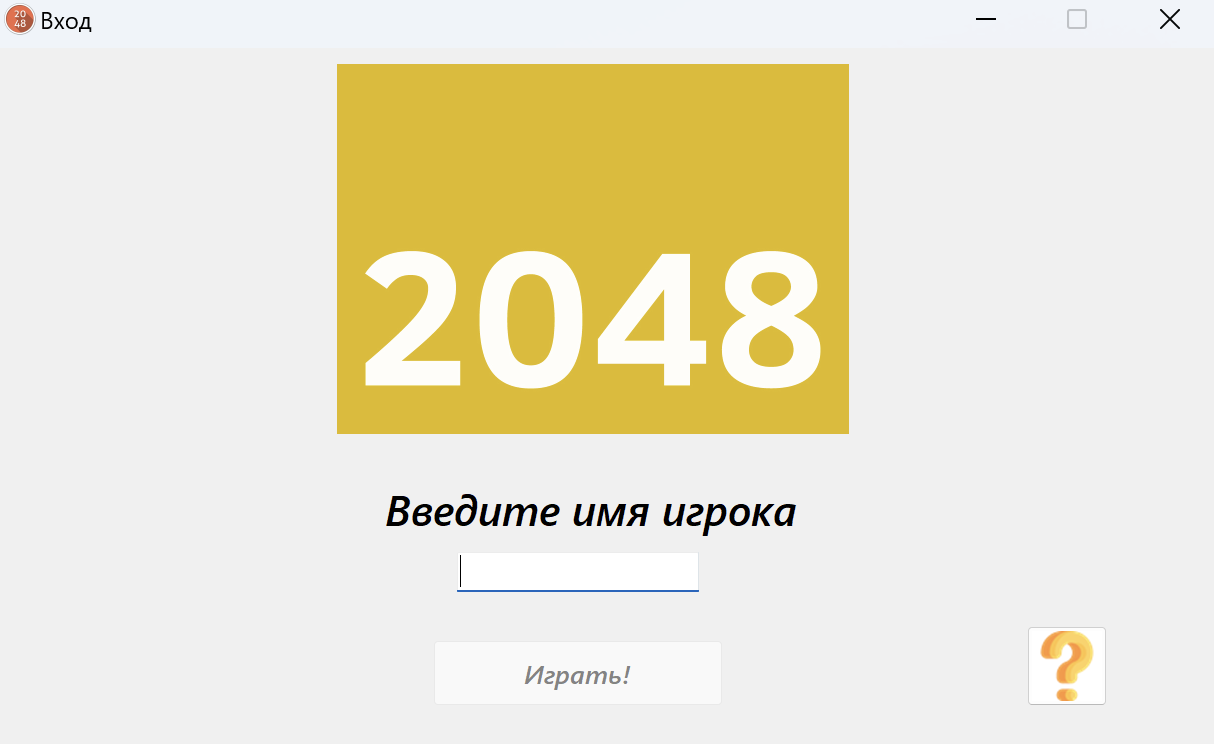
****

Рисунок 5.1 – Главное меню приложения до входа

* + 1. Окно главного меню

Окно главного меню содержит несколько компонентов, включая в себя логотип игры, изображение в качестве фона и несколько кнопок. Кнопки позволяют игроку перейти на игровой экран, настроить параметры игрового поля, выбрать количество случайных ячеек и настроить диапазон значений для этих ячеек. Кроме того, есть кнопка для отображения информации о разработчике и инструкция, которая объясняет механику и правила игры. Внешний вид окна главного меню представлен на рисунке 5.2.

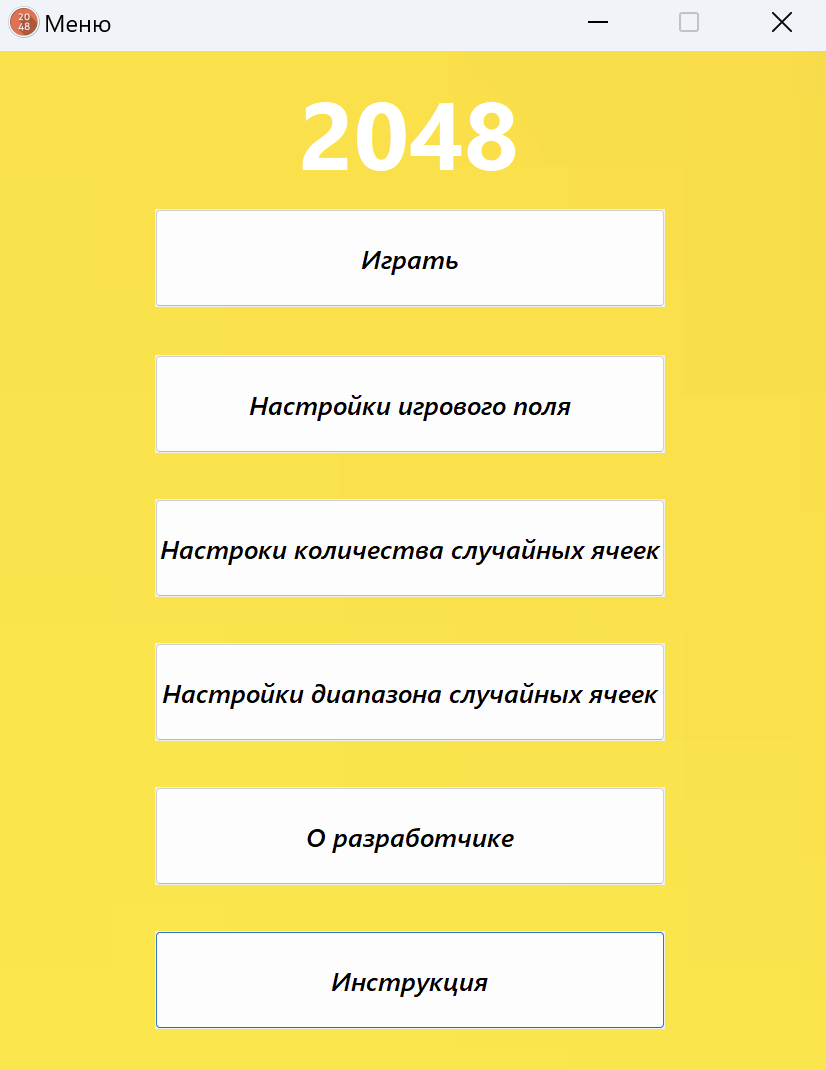
****

Рисунок 5.2 – Окно главного меню

* + 1. Окно игрового процесса

Данное окно включает в себя игровое поле, кнопки передвижения плиток, логотип, список изображений с весом плиток и скругленные прямоугольники с информацией о счете игрока и рекорде. Кнопка «Управление» содержит кнопки для передвижения плиток в разные направления. Игровое поле представлено изображением, а кнопки «Очистить поле» и «Вернуться на шаг назад» выполняют соответствующие функции. Внешний вид данного окна представлен на рисунке 5.3.

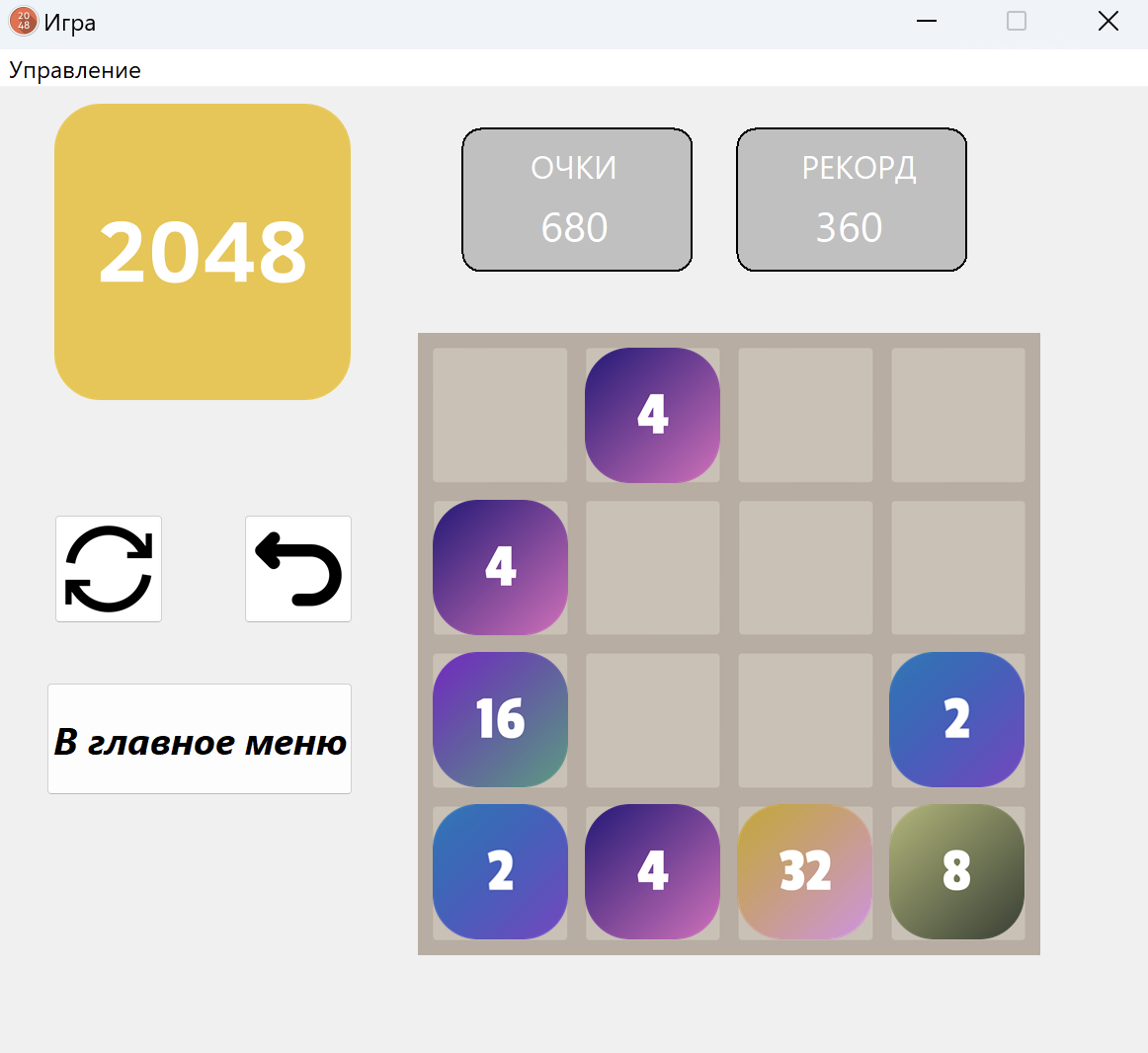
****

Рисунок 5.3 – Окно игрового процесса

* + 1. Окно с игровыми правилами

В этом окне представлена информация о том, как играть в данную игру. Расписаны самые базовые правила игры, клавиши управления игрой, условие завершение игры, а также условия, при которых игрок выигрывает. Это окно можно вызвать из главного меню. Внешний вид данного окна представлен на рисунке 5.4.

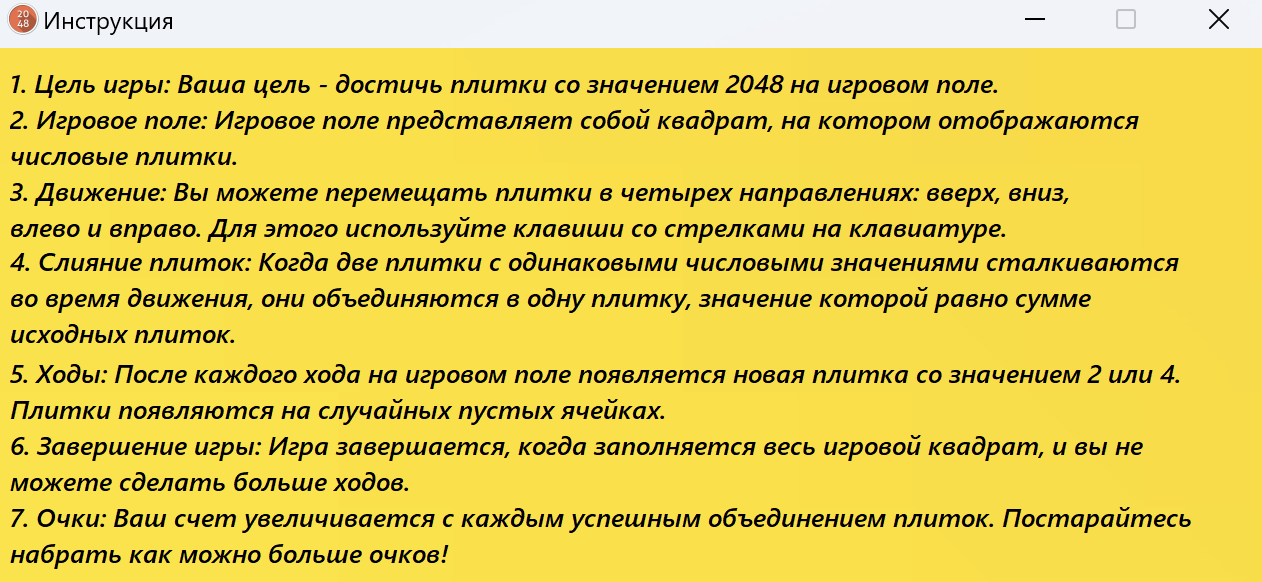
****

Рисунок 5.4 – Окно с игровыми правилами

* + 1. Окно, сообщающее о проигрыше

В случае поражения игрока, это окно сообщает пользователю о его счету

и оповещает о проигрыше, а также предлагает игроку сыграть новую игру. Внешний вид данного окна представлен на рисунке 5.5.

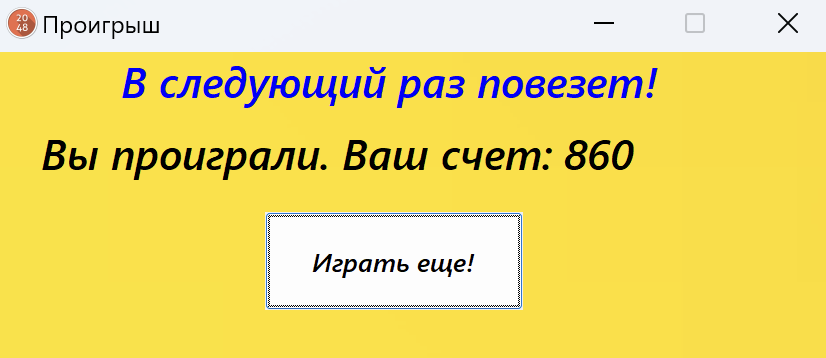


Рисунок 5.5 – Окно, сообщающее о проигрыше

Таким образом все экраны приложения делают его вид привлекательным для пользователей, а интуитивно-понятный интерфейс помогает привлекать игроков разных возрастных категорий.

# ЗАКЛЮЧЕНИЕ

Компьютерные игры играют важную роль в современном мире, предоставляя людям возможность отдыха и развлечения.

В рамках данного курсового проекта было разработано программное средство под названием «2048», предназначенное для операционной системы Windows. Это приложение позволяет пользователям наслаждаться захватывающим игровым процессом игры «2048» и провести время с увлекательной головоломкой. Также оно подходит как для новичков, так и для опытных игроков, которые хотят развить свои навыки игры и достичь высоких результатов. При разработке программного средства были успешно реализованы функции:

* взаимодействие ячеек;
* передвижение плиток игрового поля;
* анимация передвижения ячеек;
* представление плиток в виде картинок.
* проверка существования пользователя
* сохранение результатов в файл;
* подсчет количества очков;
* выбор количества плиток, которое появляется каждый ход;
* динамический размер поля;
* изменение максимального веса плитки, которая может появиться;
* хранение информации о пользователях.

Реализация игры «2048» позволила применить и закрепить навыки программирования. Она требует от программиста умения работать с компонентами визуального интерфейса, обрабатывать события и разрабатывать эффективные алгоритмы для управления игровым процессом.

Игра «2048» предлагает пользователям погрузиться в увлекательный мир числовой головоломки, позволяет освоить ее правила и стратегии, а также насладиться игровым процессом в виртуальной среде. Данная игра идеально подходит как для новичков, так и для опытных игроков, желающих улучшить свои навыки в решении числовых головоломок.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Род Стивене Delphi Готовые алгоритмы [Текст]. 2-е изд., СПб : Питер, 2004 – 384 с. – ISBN 5-94074-202-5
2. Объектно-Ориентированное Программирование [Интернет-ресурс]. – https://www.bsuir.by/m/12\_100229\_1\_90135.pdf.
3. Парижский С.М. Delphi. Учимся на примерах /Под ред. Ю. А. Шпака [Текст]. – Киев: МК-Пресс, 2005. – 216 с. – ISBN 978-5-9775-0516-1.
4. Embarcadero Documentation [Интернет-ресурс]. – https://docwiki.embarcadero.com/RADStudio/Sydney/en/Documentation
5. Олег Годик. «Программирование на Delphi 7. Основы создания приложений для Windows» [Текст]. – ДМК Пресс, 2004. – 270 с.
6. Фаулер М. Предметно-ориентированные языки программирования / [Текст]. – М.: Вильямс, 2011. – 576 с. – ISBN 978-5-8459-1738-6.
7. Виктор Рубцов. «Delphi 7. Самоучитель» [Текст]. – БХВ-Петербург, 2003. – 105 с.
8. Глухова, Л. А. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. / Л. А. Глухова, Е. Е. Фадеева, Е. П. Фадеева. – Минск: БГУИР, 2007. – Ч. 3. – 51 с.
9. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. [Текст]. – СПб, 2003. – 47 с.

# 

# 

# ПРИЛОЖЕНИЕ А

Исходный код

Unit LoginUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.Imaging.Pngimage,

Vcl.StdCtrls, MainGameUnit, MenuUnit, SetupAmountU;

Type

TLoginForm = Class(TForm)

LogoImage: TImage;

UserNickLabel: TLabel;

NickNameEdit: TEdit;

PlayButton: TButton;

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

Procedure NickNameEditChange(Sender: TObject);

Procedure PlayButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

LoginForm: TLoginForm;

Implementation

{$R \*.dfm}

Const

SET\_OF\_ALPHABET = ['a' .. 'z', 'A' .. 'Z', #8];

Procedure TLoginForm.NickNameEditChange(Sender: TObject);

Var

I: Integer;

IsCorrect: Boolean;

Begin

I := 1;

IsCorrect := True;

While IsCorrect And (I <= Length(NickNameEdit.Text)) Do

Begin

If (Not(NickNameEdit.Text[I] In SET\_OF\_ALPHABET)) Then

Begin

IsCorrect := False;

NickNameEdit.Text := '';

End;

Inc(I);

End;

If Length(NickNameEdit.Text) < 1 Then

PlayButton.Enabled := False

Else

PlayButton.Enabled := True;

End;

Procedure TLoginForm.NickNameEditKeyPress(Sender: TObject; Var Key: Char);

Begin

If Not(Key In SET\_OF\_ALPHABET) Then

Key := #0;

End;

Procedure TLoginForm.PlayButtonClick(Sender: TObject);

Begin

Setupform.Show;

LoginForm.Hide;

End;

End.

Unit MenuUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, MainGameUnit, SetupAmountU,

SetupRangeUnit, FieldSetupUnit, InstructionUnit, Vcl.Imaging.Pngimage,

Vcl.ExtCtrls;

Type

TSetupForm = Class(TForm)

PlayButton: TButton;

FieldSizeSetUpBtn: TButton;

AmountSetupButton: TButton;

RangeSetupBtn: TButton;

AboutDevButton: TButton;

InstructionButton: TButton;

BackGroundImage: TImage;

LogoLabel: TLabel;

Procedure AmountSetupButtonClick(Sender: TObject);

Procedure RangeSetupBtnClick(Sender: TObject);

Procedure FieldSizeSetUpBtnClick(Sender: TObject);

Procedure PlayButtonClick(Sender: TObject);

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

Procedure AboutDevButtonClick(Sender: TObject);

Procedure InstructionButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Const

HistoryFilePath = 'UserFile.txt';

RECORD\_LABEL\_LEFT = 860;

Type

TUser = Record

NickName: String[10];

CurrentRecord: Integer;

End;

Var

SetupForm: TSetupForm;

User: TUser;

Implementation

Uses

LoginUnit;

Const

MAX\_VALUE = 99999;

MIN\_VALUE = 0;

Type

TUSerFile = File Of TUser;

{$R \*.dfm}

Function CheckFileExist(Var F: TUSerFile): Boolean;

Var

IsExist: Boolean;

Begin

IsExist := True;

Try

Reset(F);

Except

IsExist := False;

End;

CheckFileExist := IsExist;

End;

Function CheckRange(Num: Integer): Boolean;

Var

InRange: Boolean;

Begin

InRange := True;

If (Num < MIN\_VALUE) Or (Num > MAX\_VALUE) Then

InRange := False;

CheckRange := InRange;

End;

Procedure FindUserInHistory();

Var

UserFile: TUserFile;

FindUser: TUser;

NickNameToFind: String;

IsFileExist: Boolean;

Begin

IsFileExist := True;

NickNameToFind := LoginForm.NickNameEdit.Text;

AssignFile(UserFile, HistoryFilePath);

IsFileExist := CheckFileExist(UserFile);

If IsFileExist Then

Begin

While (Not EOF(UserFile)) And (FindUser.NickName <> NickNameToFind) Do

Read(UserFile, FindUser);

If (FindUser.NickName <> NickNameToFind) Or

Not CheckRange(FindUser.CurrentRecord) Then

PlayForm.RecordIntLabel.Caption := '0'

Else

PlayForm.RecordIntLabel.Caption := IntToStr(FindUser.CurrentRecord);

CloseFile(UserFile);

End;

End;

Procedure GetCorrectPlaceForRecord();

Var

CurRecord: Integer;

Begin

CurRecord := StrToInt(PlayForm.RecordIntLabel.Caption);

PlayForm.RecordIntLabel.Left := RECORD\_LABEL\_LEFT;

While CurRecord <> 0 Do

Begin

CurRecord := CurRecord Div 10;

PlayForm.RecordIntLabel.Left := PlayForm.RecordIntLabel.Left - 10;

End;

End;

Procedure TSetupForm.AboutDevButtonClick(Sender: TObject);

Begin

MessageBox(0,

'Разработал Бычковский Павел Васильевич, студент группы 351005',

'О разработчике', 0);

End;

Procedure TSetupForm.AmountSetupButtonClick(Sender: TObject);

Begin

SetupAmountForm.Show;

SetupForm.Hide;

End;

Procedure TSetupForm.FieldSizeSetUpBtnClick(Sender: TObject);

Begin

FieldSetupForm.Show;

SetupForm.Hide;

End;

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

Begin

LoginForm.Close;

End;

Procedure TSetupForm.InstructionButtonClick(Sender: TObject);

Begin

InstructionForm.ShowModal;

End;

Procedure TSetupForm.PlayButtonClick(Sender: TObject);

Begin

FindUserInHistory();

GetCorrectPlaceForRecord();

SetupForm.Hide;

PlayForm.Show;

PlayForm.MoveButton.Enabled := True;

End;

Procedure TSetupForm.RangeSetupBtnClick(Sender: TObject);

Begin

SetupForm.Hide;

SetupRangeForm.Show;

End;

End.

Unit MainGameUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, System.Math,

Vcl.Menus, Vcl.Buttons, Vcl.ExtCtrls, Vcl.Imaging.Pngimage, Vcl.ComCtrls,

System.ImageList, Vcl.ImgList, ContinueChoiceUnit, LoseUnit;

Type

TPlayForm = Class(TForm)

MainMenu1: TMainMenu;

MoveButton: TMenuItem;

UpButton: TMenuItem;

DownButton: TMenuItem;

LeftButton: TMenuItem;

RightButton: TMenuItem;

MainImage: TImage;

AnimationTimer: TTimer;

ScoreShape: TShape;

ScoreLabel: TLabel;

ScoreIntLabel: TLabel;

RecordShape: TShape;

RecordLabel: TLabel;

RecordIntLabel: TLabel;

LogoImage: TImage;

MainMenuButton: TSpeedButton;

RestartButton: TSpeedButton;

BackButton: TSpeedButton;

Procedure UpButtonClick(Sender: TObject);

Procedure DownButtonClick(Sender: TObject);

Procedure LeftButtonClick(Sender: TObject);

Procedure RightButtonClick(Sender: TObject);

Procedure AnimationTimerTimer(Sender: TObject);

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

Procedure FormShow(Sender: TObject);

Procedure BackButtonClick(Sender: TObject);

Procedure RestartButtonClick(Sender: TObject);

Procedure MainMenuButtonClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

PlayForm: TPlayForm;

Type

TEmptyCells = Record

MatrixI: Integer;

MatrixJ: Integer;

End;

TAnimationCell = Record

Picture: TImage;

Direction: Integer;

End;

TArrOfCellsToAnimate = Array Of TAnimationCell;

TEmptyCellsArr = Array Of TEmptyCells;

TField = Record

NumMatrix: Array Of Array Of Integer;

ImageMatrix: Array Of Array Of TImage;

End;

PStack = ^TStack;

TStack = Record

CurField: TField;

Prev: PStack;

Next: PStack;

End;

TDirection = (Vertical, Horizontal);

Var

Direction: TDirection;

Stack: PStack;

FieldSize, MaxRange, MaxAmount: Integer;

CellsArrToAnimate: TArrOfCellsToAnimate;

Field: TField;

EmptyCellsArr: TEmptyCellsArr;

CounterOfMoves: Integer;

EndlessGame: Boolean;

Implementation

Uses

MenuUnit, FieldSetupUnit, SetupRangeUnit, SetupAmountU, LoginUnit;

Const

FIELD\_3\_BOTTOM\_BORDER = 900;

FIELD\_3\_RIGHT\_BORDER = 1100;

FIELD\_4\_BOTTOM\_BORDER = 1100;

FIELD\_4\_RIGHT\_BORDER = 1200;

FIELD\_5\_BOTTOM\_BORDER = 1230;

FIELD\_5\_RIGHT\_BORDER = 1325;

DIGIT\_SIZE = 13;

Type

TUSerFile = File Of TUser;

{$R \*.dfm}

Procedure ScoreChangePlace(CurLabel: TLabel; Direction: Integer);

Begin

CurLabel.Left := CurLabel.Left - DIGIT\_SIZE \* Direction;

End;

Const

POINT\_COEF = 10;

Function CalculatePoints(): Integer;

Var

I, J, Points: Integer;

Begin

Points := 0;

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Points := Points + Field.NumMatrix[I, J];

Points := Points \* POINT\_COEF;

CalculatePoints := Points;

End;

Procedure ShowLose();

Begin

LoseForm.ShowModal;

End;

Function UpdateCurRecord(CurLabel: TLabel): Integer;

Var

I, J, CurRecord: Integer;

OldLabel: String;

Begin

CurRecord := 0;

OldLabel := CurLabel.Caption;

CurRecord := CalculatePoints();

CurLabel.Caption := IntToStr(CurRecord);

If Length(OldLabel) < Length(IntToStr(CurRecord)) Then

ScoreChangePlace(CurLabel, 1)

Else If Length(OldLabel) > Length(IntToStr(CurRecord)) Then

ScoreChangePlace(CurLabel, -1);

End;

Function IsUserInArr(Const UserArr: Array Of TUser; Const UserNickName: String;

Var Num: Integer): Boolean;

Var

IsFound: Boolean;

Begin

Num := 0;

IsFound := False;

While (Not IsFound) And (Num < Length(UserArr)) Do

Begin

If UserArr[Num].NickName = UserNickName Then

IsFound := True

Else

Inc(Num);

End;

IsUserInArr := IsFound;

End;

Function CheckFileExist(Var F: TUSerFile): Boolean;

Var

IsExist: Boolean;

Begin

IsExist := True;

Try

Reset(F);

Except

IsExist := False;

End;

CheckFileExist := IsExist;

End;

Procedure AddUserToFile(NickName: String; CurRecord: Integer);

Var

UserFile: TUSerFile;

User: TUser;

UserArr: Array Of TUser;

I, Position: Integer;

IsFound, IsExist: Boolean;

Begin

User.NickName := NickName;

User.CurrentRecord := CurRecord;

AssignFile(UserFile, HistoryFilePath);

IsExist := CheckFileExist(UserFile);

If IsExist Then

Begin

I := 0;

While Not EOF(USerFile) Do

Begin

SetLength(UserArr, I + 1);

Read(UserFile, UserArr[I]);

Inc(I);

End;

End;

Rewrite(UserFile);

Position := 0;

IsFound := IsUserInArr(UserArr, NickName, Position);

If IsFound Then

Begin

If UserArr[Position].CurrentRecord < CurRecord Then

UserArr[Position] := User;

End

Else

Begin

SetLength(UserArr, Length(UserArr) + 1);

UserArr[High(UserArr)] := User;

End;

For I := Low(UserArr) To High(UserArr) Do

Write(UserFile, UserArr[I]);

CloseFile(UserFile);

End;

Procedure CopyField(Field: TField);

Var

I, J: Integer;

Begin

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

Stack.CurField.NumMatrix[I, J] := Field.NumMatrix[I, J];

Stack.CurField.ImageMatrix[I, J] := Field.ImageMatrix[I, J];

End;

End;

Procedure PushStack();

Var

Prev: PStack;

Begin

Prev := Stack;

New(Stack.Next);

Stack := Stack.Next;

Setlength(Stack.CurField.NumMatrix, FieldSize, FieldSize);

SetLength(Stack.CurField.ImageMatrix, FieldSize, FieldSize);

CopyField(Field);

Stack.Next := Nil;

Stack.Prev := Prev;

PlayForm.BackButton.Enabled := True;

End;

Function CalculatePower(NumPower: Integer): Integer;

Var

I, Num: Integer;

Begin

Num := 1;

For I := 1 To NumPower Do

Num := 2 \* Num;

CalculatePower := Num;

End;

Const

IMAGE\_FILE\_EXTENSION = '.png';

Procedure FillRandomNumber();

Var

I, MatrixPosition, CurPower, CurNumber: Integer;

ImagePath: String;

IsEnoughSpace: Boolean;

Begin

IsEnoughSpace := True;

If Length(EmptyCellsArr) < MaxAmount Then

Begin

ShowLose();

IsEnoughSpace := False;

End;

If IsEnoughSpace Then

For I := 1 To MaxAmount Do

Begin

CurPower := RandomRange(1, MaxRange + 1);

CurNumber := CalculatePower(CurPower);

MatrixPosition := Random(High(EmptyCellsArr));

Field.NumMatrix[EmptyCellsArr[MatrixPosition].MatrixI,

EmptyCellsArr[MatrixPosition].MatrixJ] := CurNumber;

ImagePath := IntToStr(CurNumber) + IMAGE\_FILE\_EXTENSION;

Field.ImageMatrix[EmptyCellsArr[MatrixPosition].MatrixI,

EmptyCellsArr[MatrixPosition].MatrixJ].Picture.LoadFromFile

(ImagePath);

EmptyCellsArr[MatrixPosition] := EmptyCellsArr[High(EmptyCellsArr)];

SetLength(EmptyCellsArr, High(EmptyCellsArr));

End

Else

PlayForm.RestartButton.Click;

End;

Var

Delta: Integer;

Procedure SwapPictures(YDirection, XDirection, I, J: Integer);

Var

BuffImage: TImage;

Begin

BuffImage := Field.ImageMatrix[I + YDirection, J + XDirection];

Field.ImageMatrix[I + YDirection, J + XDirection] :=

Field.ImageMatrix[I, J];

Field.ImageMatrix[I, J] := BuffImage;

End;

Procedure AddToAnimationArr(Image: TImage; Direction: Integer);

Begin

SetLength(CellsArrToAnimate, Length(CellsArrToAnimate) + 1);

CellsArrToAnimate[High(CellsArrToAnimate)].Picture := Image;

CellsArrToAnimate[High(CellsArrToAnimate)].Direction := Direction;

End;

Procedure AddToEmptyCellsArr(I, J: Integer);

Begin

Setlength(EmptyCellsArr, Length(EmptyCellsArr) + 1);

EmptyCellsArr[High(EmptyCellsArr)].MatrixI := I;

EmptyCellsArr[High(EmptyCellsArr)].MatrixJ := J;

End;

Procedure SwitchCellsIcon(ImageToNewIcon, ImageToEmptyIcon: TImage;

NewNum: Integer);

Begin

ImageToNewIcon.Picture.LoadFromFile(IntToStr(NewNum) +

IMAGE\_FILE\_EXTENSION);

ImageToEmptyIcon.Picture := Nil;

End;

Procedure GetEmptyCells();

Var

I, J: Integer;

Begin

SetLength(EmptyCellsArr, 0);

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

If Field.NumMatrix[I, J] = 0 Then

AddToEmptyCellsArr(I, J);

End;

Procedure SumDown(StartRow, EndRow, StartCol, EndCol: Integer);

Var

I, J, Direction, FindEmpty: Integer;

Begin

Direction := 1;

FindEmpty := 0;

If StartRow > EndRow Then

Begin

Direction := -1;

End;

I := StartRow;

While Abs(I - StartRow) < High(Field.NumMatrix) Do

Begin

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

FindEmpty := 0;

If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I + Direction, J] = 0) Then

Begin

Inc(CounterOFMoves);

Field.NumMatrix[I + Direction, J] := Field.NumMatrix[I, J];

Field.NumMatrix[I, J] := 0;

AddToAnimationArr(Field.ImageMatrix[I, J], Direction);

AddToAnimationArr(Field.ImageMatrix[I + Direction, J],

-Direction);

While (EmptyCellsArr[FindEmpty].MatrixI <> I + Direction) Or

(EmptyCellsArr[FindEmpty].MatrixJ <> J) Do

Inc(FindEmpty);

EmptyCellsArr[FindEmpty].MatrixI := I;

SwapPictures(Direction, 0, I, J);

End

Else If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I + Direction, J] = Field.NumMatrix[I, J]) Then

Begin

Inc(CounterOFMoves);

FindEmpty := 0;

Field.NumMatrix[I + Direction, J] := Field.NumMatrix[I, J] \* 2;

Field.NumMatrix[I, J] := 0;

SwitchCellsIcon(Field.ImageMatrix[I, J],

Field.ImageMatrix[I + Direction, J],

Field.NumMatrix[I + Direction, J]);

SwapPictures(Direction, 0, I, J);

AddToAnimationArr(Field.ImageMatrix[I, J], -Direction);

AddToAnimationArr(Field.ImageMatrix[I + Direction, J],

Direction);

AddToEmptyCellsArr(I, J);

End;

End;

I := I + Direction;

End;

End;

Const

NUMBER\_TO\_WIN = 2048;

Function FindMaxCell(): Boolean;

Var

IsMaxNum: Boolean;

Begin

IsMaxNum := False;

For Var I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For Var J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

If Field.NumMatrix[I, J] = NUMBER\_TO\_WIN Then

IsMaxNum := True;

FindMaxCell := IsMaxNum;

End;

Procedure Sumright(StartRow, EndRow, StartCol, EndCol: Integer);

Var

I, J, Direction, FindEmpty: Integer;

Begin

Direction := 1;

If StartCol > EndCol Then

Direction := -1;

J := StartCol;

While Abs(J - StartCol) < High(Field.NumMatrix) Do

Begin

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

FindEmpty := 0;

If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I, J + Direction] = 0) Then

Begin

Inc(CounterOFMoves);

Field.NumMatrix[I, J + Direction] := Field.NumMatrix[I, J];

Field.NumMatrix[I, J] := 0;

While (EmptyCellsArr[FindEmpty].MatrixI <> I) Or

(EmptyCellsArr[FindEmpty].MatrixJ <> J + Direction) Do

Inc(FindEmpty);

EmptyCellsArr[FindEmpty].MatrixJ := J;

AddToAnimationArr(Field.ImageMatrix[I, J], Direction);

AddToAnimationArr(Field.ImageMatrix[I, J + Direction],

-Direction);

SwapPictures(0, Direction, I, J);

End

Else If (Field.NumMatrix[I, J] <> 0) And

(Field.NumMatrix[I, J + Direction] = Field.NumMatrix[I, J]) Then

Begin

Inc(CounterOFMoves);

Field.NumMatrix[I, J + Direction] := Field.NumMatrix[I, J] \* 2;

Field.NumMatrix[I, J] := 0;

SwitchCellsIcon(Field.ImageMatrix[I, J],

Field.ImageMatrix[I, J + Direction],

Field.NumMatrix[I, J + Direction]);

SwapPictures(0, Direction, I, J);

AddToEmptyCellsArr(I, J);

AddToAnimationArr(Field.ImageMatrix[I, J], -Direction);

AddToAnimationArr(Field.ImageMatrix[I, J + Direction],

Direction);

End;

End;

J := J + Direction;

End;

End;

Procedure DeletePictures();

Var

I, J: Integer;

Begin

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

If Field.ImageMatrix[I, J] <> Nil Then

Field.ImageMatrix[I, J].Free;

End;

Const

CELL\_SIZE = 150;

Procedure GetStartImages();

Var

I, J: Integer;

Begin

With PlayForm Do

Begin

DeletePictures();

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

Begin

Field.ImageMatrix[I, J] := TImage.Create(PlayForm);

Field.ImageMatrix[I, J].Parent := PlayForm;

Field.ImageMatrix[I, J].Width := CELL\_SIZE;

Field.ImageMatrix[I, J].Height := CELL\_SIZE;

End;

End;

End;

Const

START\_IMAGE\_LEFT = 443;

START\_IMAGE\_TOP = 263;

Procedure GetStartPositions();

Var

I, J: Integer;

Begin

For I := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

For J := Low(Field.ImageMatrix) To High(Field.ImageMatrix) Do

Begin

Field.ImageMatrix[I, J].Top := START\_IMAGE\_LEFT + I \*

(Field.ImageMatrix[I, J].Height + 4);

Field.ImageMatrix[I, J].Left := START\_IMAGE\_TOP + J \*

(Field.ImageMatrix[I, J].Width + 4);

End;

End;

Const

START\_SCORE\_LABEL\_LEFT = 578;

Procedure ClearField();

Begin

SetLength(Field.NumMatrix, 0, 0);

SetLength(EmptyCellsArr, 0);

DeletePictures();

SetLength(Field.ImageMatrix, 0, 0);

PlayForm.ScoreIntLabel.Caption := '0';

PlayForm.ScoreIntLabel.Left := START\_SCORE\_LABEL\_LEFT;

End;

Procedure TPlayForm.BackButtonClick(Sender: TObject);

Var

I, J: Integer;

Begin

Field := Stack.CurField;

Stack := Stack.Prev;

GetStartImages();

GetStartPositions();

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

For J := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

Begin

If Field.NumMatrix[I, J] = 0 Then

Field.ImageMatrix[I, J].Picture := Nil

Else

Field.ImageMatrix[I, J].Picture.LoadFromFile

(IntToStr(Field.NumMatrix[I, J]) + IMAGE\_FILE\_EXTENSION);

End;

GetEmptyCells();

UpDateCurRecord(ScoreIntLabel);

If Stack.Prev = Nil Then

BackButton.Enabled := False;

MoveButton.Enabled := True;

End;

Procedure GoBackScreen();

Begin

AddUserToFile(LoginForm.NickNameEdit.Text,

StrToInt(PlayForm.ScoreIntLabel.Caption));

PlayForm.Hide;

SetupForm.Show;

ClearField();

End;

Procedure PopStack();

Var

Buff: PStack;

Begin

Buff := Stack;

Stack := Stack.Prev;

Stack.Next := Nil;

Dispose(Buff);

End;

Procedure FieldMove(FDirection: TDirection; StartRow, EndRow, StartCol,

EndCol: Integer);

Var

I: Integer;

Begin

CounterOfMoves := 0;

PushStack();

Direction := FDirection;

Delta := 0;

PlayForm.MoveButton.Enabled := False;

For I := Low(Field.NumMatrix) To High(Field.NumMatrix) Do

If Direction = Horizontal Then

SumRight(StartRow, EndRow, StartCol, EndCol)

Else

SumDown(StartRow, EndRow, StartCol, EndCol);

PlayForm.AnimationTimer.Enabled := True;

UpDateCurRecord(PlayForm.ScoreIntLabel);

If Not EndLessGame And FindMaxCell() Then

ContinueChoiceForm.ShowModal;

End;

Procedure TPlayForm.DownButtonClick(Sender: TObject);

Begin

FieldMove(Vertical, 0, High(Field.NumMatrix) - 1, 0, High(Field.NumMatrix));

End;

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

Begin

GoBackScreen();

End;

Procedure FillEmptyCellsArr();

Begin

For Var I := Low(EmptyCellsArr) To High(EmptyCellsArr) Do

Begin

EmptyCellsArr[I].MatrixI := I Div FieldSize;

EmptyCellsArr[I].MatrixJ := I Mod FieldSize;

End;

End;

Procedure ChangeFormSize(Const FieldSize: Integer);

Begin

Case FieldSize Of

3:

Begin

PlayForm.Height := FIELD\_3\_BOTTOM\_BORDER;

PlayForm.Width := FIELD\_3\_RIGHT\_BORDER;

End;

4:

Begin

PlayForm.Height := FIELD\_4\_BOTTOM\_BORDER;

PlayForm.Width := FIELD\_4\_RIGHT\_BORDER;

End;

5:

Begin

PlayForm.Height := FIELD\_5\_BOTTOM\_BORDER;

PlayForm.Width := FIELD\_5\_RIGHT\_BORDER;

End;

End;

End;

Const

FILE\_NAME = 'field.png';

Procedure TPlayForm.FormShow(Sender: TObject);

Var

I: Integer;

Begin

FieldSize := FieldSetupForm.FieldSizeTrackBar.Position;

MaxRange := SetupRangeForm.RangeTrackBar.Position;

MaxAmount := SetupAmountForm.AmountTrackBar.Position;

ChangeFormSize(FieldSize);

Randomize;

Setlength(EmptyCellsArr, FieldSize \* FieldSize);

SetLength(Field.ImageMatrix, FieldSize, FieldSize);

FillEmptyCellsArr();

GetStartImages();

New(Stack);

Stack.Prev := Nil;

Stack.Next := Nil;

MainImage.Picture.LoadFromFile(IntToStr(FieldSize) + FILE\_NAME);

Setlength(Field.NumMatrix, FieldSize, FieldSize);

GetStartPositions();

FillRandomNumber();

BackButton.Enabled := False;

EndLessGame := False;

End;

Procedure TPlayForm.LeftButtonClick(Sender: TObject);

Begin

FieldMove(Horizontal, 0, High(Field.NumMatrix) + 1,

High(Field.NumMatrix), 1);

End;

Procedure TPlayForm.RestartButtonClick(Sender: TObject);

Var

I: Integer;

Begin

If StrToInt(ScoreIntLabel.Caption) > StrToInt(RecordIntLabel.Caption) Then

RecordIntLabel.Caption := ScoreIntLabel.Caption;

ClearField();

Setlength(EmptyCellsArr, FieldSize \* FieldSize);

SetLength(Field.ImageMatrix, FieldSize, FieldSize);

For I := Low(EmptyCellsArr) To High(EmptyCellsArr) Do

Begin

EmptyCellsArr[I].MatrixI := I Div FieldSize;

EmptyCellsArr[I].MatrixJ := I Mod FieldSize;

End;

GetStartImages();

New(Stack);

Stack.Prev := Nil;

Stack.Next := Nil;

Setlength(Field.NumMatrix, FieldSize, FieldSize);

GetStartPositions();

FillRandomNumber();

BackButton.Enabled := False;

MoveButton.Enabled := True;

End;

Procedure TPlayForm.RightButtonClick(Sender: TObject);

Begin

FieldMove(Horizontal, 0, High(Field.NumMatrix) + 1, 0,

High(Field.NumMatrix) - 1);

End;

Procedure TPlayForm.MainMenuButtonClick(Sender: TObject);

Begin

GoBackScreen();

End;

Procedure DecideWithCounterOfMoves(CounterOfMoves: Integer);

Begin

If CounterOfMoves <> 0 Then

Begin

FillRandomNumber()

End

Else If (CounterofMoves = 0) And (Length(EmptyCellsArr) = 0) Then

Begin

ShowLose();

End

Else

Begin

PopStack();

PlayForm.BackButton.Enabled := False;

If Stack.Prev <> Nil Then

PlayForm.BackButton.Enabled := True;

End;

End;

Const

CELLSIZE\_FOR\_ANIMATION = 154;

ANIMATION\_STEP = 22;

Procedure TPlayForm.AnimationTimerTimer(Sender: TObject);

Var

Limit: Integer;

Begin

Limit := CELLSIZE\_FOR\_ANIMATION \* Length(CellsArrToAnimate);

If Abs(Delta) < Limit Then

For Var I := Low(CellsArrToAnimate) To High(CellsArrToAnimate) Do

Begin

If Direction = Vertical Then

CellsArrToAnimate[I].Picture.Top := CellsArrToAnimate[I]

.Picture.Top + CellsArrToAnimate[I].Direction \* ANIMATION\_STEP

Else

CellsArrToAnimate[I].Picture.Left := CellsArrToAnimate[I]

.Picture.Left + CellsArrToAnimate[I].Direction \*

ANIMATION\_STEP;

Delta := Delta + ANIMATION\_STEP;

End

Else

Begin

AnimationTimer.Enabled := False;

Setlength(CellsArrToAnimate, 0);

DecideWithCounterOfMoves(CounterOfMoves);

MoveButton.Enabled := True;

End;

End;

Procedure TPlayForm.UpButtonClick(Sender: TObject);

Begin

FieldMove(Vertical, High(Field.NumMatrix), 1, 0, High(Field.NumMatrix));

End;

End.

Unit SetupRangeUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, Vcl.StdCtrls,

Vcl.Imaging.Pngimage, Vcl.ExtCtrls;

Type

TSetupRangeForm = Class(TForm)

InfoRangeLabel: TLabel;

RangeTrackBar: TTrackBar;

Label4: TLabel;

Label8: TLabel;

Label16: TLabel;

Label32: TLabel;

SaveRangeSetup: TButton;

BackGroundImage: TImage;

Procedure SaveRangeSetupClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

SetupRangeForm: TSetupRangeForm;

Implementation

Uses

MenuUnit;

{$R \*.dfm}

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

Begin

SetupRangeForm.Hide;

SetupForm.Show;

End;

Procedure TSetupRangeForm.SaveRangeSetupClick(Sender: TObject);

Begin

SetupRangeForm.Hide;

SetupForm.Show;

End;

End.

Unit SetupAmountU;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ComCtrls,

Vcl.Imaging.Pngimage, Vcl.ExtCtrls;

Type

TSetupAmountForm = Class(TForm)

AmountTrackBar: TTrackBar;

AmountLabel: TLabel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

SaveChanges: TButton;

BackGroundImage: TImage;

Procedure FormCreate(Sender: TObject);

Procedure SaveChangesClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

SetupAmountForm: TSetupAmountForm;

Implementation

Uses

MenuUnit;

{$R \*.dfm}

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

Begin

SetupAmountForm.Hide;

SetupForm.Show;

End;

Procedure TSetupAmountForm.FormCreate(Sender: TObject);

Begin

AmountLabel.Caption := 'Выберете какое количество случайных чисел ' + #13#10

+ 'будет появляться каждый ход';

End;

Procedure TSetupAmountForm.SaveChangesClick(Sender: TObject);

Begin

SetupAmountForm.Hide;

SetupForm.Show;

End;

End.

Unit FieldSetupUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.ComCtrls,

Vcl.Imaging.Pngimage, Vcl.ExtCtrls;

Type

TFieldSetupForm = Class(TForm)

SizeInfoLabel: TLabel;

FieldSizeTrackBar: TTrackBar;

Label3: TLabel;

Label4: TLabel;

Label5: TLabel;

SaveSizeSetup: TButton;

BackGroundImage: TImage;

Procedure SaveSizeSetupClick(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

FieldSetupForm: TFieldSetupForm;

Implementation

Uses

MenuUnit;

{$R \*.dfm}

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

Begin

FieldSetupForm.Hide;

SetupForm.Show;

End;

Procedure TFieldSetupForm.SaveSizeSetupClick(Sender: TObject);

Begin

FieldSetupForm.Hide;

SetupForm.Show;

End;

End.

Unit ContinueChoiceUnit;

Interface

Uses

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

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Imaging.Pngimage,

Vcl.ExtCtrls;

Type

TContinueChoiceForm = Class(TForm)

DecideLabel: TLabel;

ButtonYes: TButton;

ButtonNo: TButton;

BackGroundImage: TImage;

Procedure ButtonYesClick(Sender: TObject);

Procedure ButtonNoClick(Sender: TObject);

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

ContinueChoiceForm: TContinueChoiceForm;

Implementation

Uses

MainGameUnit;

{$R \*.dfm}

Procedure TContinueChoiceForm.ButtonNoClick(Sender: TObject);

Begin

EndLessGame := False;

ContinueChoiceForm.Close;

PlayForm.Close;

End;

Procedure TContinueChoiceForm.ButtonYesClick(Sender: TObject);

Begin

EndLessGame := True;

ContinueChoiceForm.Close;

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.Imaging.Pngimage,

Vcl.ExtCtrls;

Type

TInstructionForm = Class(TForm)

InstructionLabel: TLabel;

InstructionLabel2: TLabel;

InstructionLabel3: TLabel;

Image1: TImage;

Procedure FormShow(Sender: TObject);

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

Private

{ Private declarations }

Public

{ Public declarations }

End;

Var

InstructionForm: TInstructionForm;

Implementation

Uses

MenuUnit;

{$R \*.dfm}

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

Begin

InstructionForm.Close;

End;

Procedure TInstructionForm.FormShow(Sender: TObject);

Begin

InstructionLabel.Caption :=

'1. Цель игры: Ваша цель - достичь плитки со значением 2048 на игровом поле.'

+ #13#10 +

'2. Игровое поле: Игровое поле представляет собой квадрат, на котором

отображаются + #13#10 + 'числовые плитки.' + #13#10 +

'3. Движение: Вы можете перемещать плитки в четырех направлениях: вверх, вниз,

+ #13#10 + влево и вправо. Для этого используйте клавиши со стрелками на

клавиатуре. ';

InstructionLabel2.Caption :=

'4. Слияние плиток: Когда две плитки с одинаковыми числовыми значениями

сталкиваются + #13#10 +

'во время движения, они объединяются в одну плитку, значение которой равно

сумме ' + #13#10 + 'исходных плиток.';

InstructionLabel3.Caption :=

'5. Ходы: После каждого хода на игровом поле появляется новая плитка со

значением 2 или 4.' + #13#10 + 'Плитки появляются на случайных пустых

ячейках.' + #13#10 + '6. Завершение игры: Игра завершается, когда заполняется

весь игровой квадрат, и вы не ' + #13#10 + 'можете сделать больше ходов. ' +

#13#10 + '7. Очки: Ваш счет увеличивается с каждым успешным объединением

плиток. Постарайтесь ' + #13#10 + 'набрать как можно больше очков!';

End;

End.

unit LoseUnit;

interface

uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Imaging.pngimage,

Vcl.ExtCtrls;

type

TLoseForm = class(TForm)

LoseLabel: TLabel;

LoseButton: TButton;

BackGroudImage: TImage;

BetterLuckLabel: TLabel;

procedure FormShow(Sender: TObject);

procedure LoseButtonClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

end;

var

LoseForm: TLoseForm;

implementation

Uses

MainGameUnit;

{$R \*.dfm}

Procedure ConfirmLose();

Begin

LoseForm.Close;

PlayForm.RestartButton.Click;

End;

procedure TLoseForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin

ConfirmLose();

end;

procedure TLoseForm.FormShow(Sender: TObject);

begin

LoseLabel.Caption := 'Вы проиграли. Ваш счет: ' + PlayForm.ScoreIntLabel.Caption;

end;

procedure TLoseForm.LoseButtonClick(Sender: TObject);

begin

ConfirmLose();

end;

end.