

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

Игровое приложение «2048»

БГУИР КП 1-40 01 01 321 ПЗ

Студент гр. 951003

Пташиц Е.Я.

Руководитель

Яскевич Д. А.

Минск 2021

СОДЕРЖАНИЕ

Введение.....	5
1 Анализ существующих аналогов.....	6
1.1 Классификация компьютерных игр	6
1.2 Обзор аналогов	7
1.2.1 Threes.....	7
1.2.2 Isotopic 256.....	8
1.2.3 2048+.....	8
2 Постановка задачи.....	10
3 Моделирование предметной области.....	11
3.1 Окно в Windows.....	11
3.2 Win32 API.....	11
3.2.1 Родительские окна и окна владельца	11
3.2.2 Дескрипторы окон.....	11
3.2.3 Координаты экрана и окна	12
3.2.4 WinMain: точка входа приложения	12
3.2.5 Сообщения окна	12
3.2.6 Цикл обработки сообщений	13
3.2.7 Функция GetMessage	13
3.2.8 Выход из приложения.....	13
4 Разработка программного средства.....	14
4.1 Разработка программной архитектуры	14
4.1.1 Класс MainGame.....	14
4.1.2 Класс Scene	14
4.1.3 Класс sceneManager.....	14
4.1.4 Класс ScoreManager	15
4.1.5 Класс Image.....	15
4.1.6 Класс Timer	16
5 Тестирование и проверка работоспособности программного средства.....	17
5.1 Тестирование игрового процесса	17
5.1 Тестирование модуля сохранения результата.....	19
6 Руководство по установке и использованию программного средства	21
6.1 Установка приложения	21
6.2 Работа с программным средством.....	21
Заключение	25
Список использованной литературы.....	26
Приложение А	27

ВВЕДЕНИЕ

Стремительное развитие компьютерной техники, появление мощнейших графических ускорителей и центральных процессоров способствовало не менее бурному развитию индустрии компьютерных игр. Выдающиеся разработки этой отрасли – это сложнейшие программы, как правило, с очень высокими требованиями к аппаратной части компьютера. Однако для возможности отдохнуть в перерыве от выполнения какой-либо работы пользователю компьютера не всегда требуется новейшая компьютерная игра, а зачастую использовать её не позволяет маломощное оборудование офисного компьютера. Именно этой цели – отдыху от монотонной работы служит разработанная в рамках данного курсового проекта программа.

Игры – неперенная составляющая любой человеческой культуры. Слово «игра» мы прилагаем почти ко всякому детскому занятию. Посредством игры наиболее интенсивно развивается целостное восприятие окружающего мира, наглядно-образное мышление, творческое воображение. Поэтому кроме программ, которые облегчают людям работу, программисты придумали увлекательное развлечение-компьютерные игры. Сначала в эти игры играли только они сами, но очень скоро их эстафету приняли дети и взрослые по всему миру. Сейчас уже трудно найти человека, имеющего компьютер и ни разу, не игравшего в компьютерные игры.

При выборе игры пользователи руководствуются правилом «Чем проще, тем лучше», прослеживающееся во всех казуальных играх, не требующих высокой концентрации и большого количества времени, затраченного на «разобраться, что да как». Этими факторами обусловлена популярность игры 2048 и множества её интерпретаций.

1 АНАЛИЗ СУЩЕСТВУЮЩИХ АНАЛОГОВ

1.1 Классификация компьютерных игр

Всё многообразие компьютерных игр можно делить на группы, используя множество самых различных способов. Основные признаки классификации:

- платформа;
- графическое изображение;
- количество игроков;
- жанр.

Основным видом разделения видеоигр на категории является разделение по платформам. Оно указывает, на каком конкретном устройстве можно запустить игру. Это является наиболее важным потому, что если у игрока нет соответствующей платформы, то он не сможет поиграть в игру, рассчитанную именно для этой платформы. Перечень платформ:

- персональный компьютер;
- игровая консоль или приставка;
- мобильное устройство.

Внешний вид – главное украшение игры. Многие игроки при выборе игры ориентируются именно на графику, поэтому существует разделение игр по типу и качеству графического изображения. По типу графики игры разделяют на:

- игры с отсутствующей графикой – текстовые игры, псевдографика;
- 2D-игры – объекты изображены в двумерном пространстве;
- 3D-игры – объекты изображены в трехмерном пространстве.

По количеству игроков выделяют следующие типы игр:

- одиночные игры;
- совместные игры на одном устройстве;
- многопользовательские.

Игровой жанр – группа игр, которые имеют схожую игровую механику и похожие правила игры. Указание жанра игры само по себе уже даёт общее примерное представление о содержании игры. Основная масса игроков выбирает новые игры, ориентируясь на свои жанровые предпочтения. Из всех возможных жанров можно выделить основные:

- симулятор;
- стратегия;
- ролевая игра;
- приключения;
- головоломка.

Головоломка – это жанр игр, которые требуют от игрока задействовать его интуицию, стратегию и логику. Они могут быть представлены как 2D, так и 3D играми.

1.2 Обзор аналогов

Головоломки – задачи, для решения которых, как правило, требуется сообразительность, а не специальные знания высокого уровня. Разгадывание головоломок является одним из любимых занятий большинства ценителей интеллектуального досуга. На сегодняшний день существует множество вариантов различных головоломок.

1.2.1 Threes

Прообразом «2048» является коммерческая игра Threes. Игровой процесс Threes представляет собой пронумерованную сетку, где игрок может перемещать плитки с цифрами, складывая одинаковые числа и образуя плитки с более крупными числами. Игра заканчивается, когда на сетке больше не остаётся свободных ячеек и, если плитки невозможно сложить. Игровой сеанс в Threes длится обычно несколько минут и заканчиваются, когда на сетке не осталось ходов. Цель игрока — продержаться как можно дольше и заработать максимально возможное количество баллов. Главным недостатком игры является то, что для игры на ОС Windows для запуска игры необходимо интернет-подключение, так как она представлена онлайн-приложением. [1]

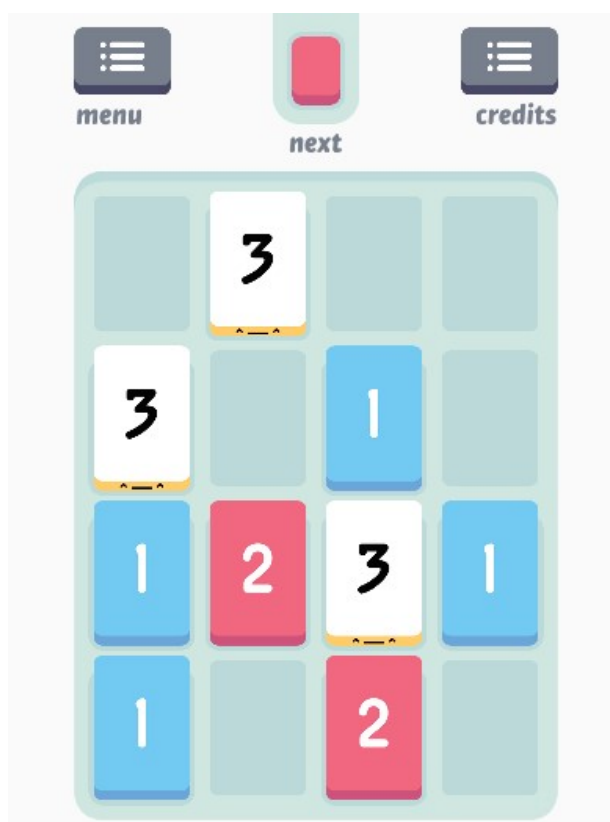


Рисунок 1.1 – Интерфейс Threes

1.2.2 Isotopic 256

Isotopic 256 представляет собой интересную вариацию игры 2048. Игровое поле представлено матрицей 3x3. Главная особенность в том, что вместо чисел плитки содержат изотопы химических веществ, которые крайне нестабильны. Сложность игры состоит в том, что нестабильные изотопы живут определенное количество ходов. Как только счётчик элемента становится равен нулю, ячейка освобождается. Недостатки игры состоят в том, что игра представлена в виде онлайн-приложения и требует постоянного интернет подключения. Ещё один недостаток также связан с реализацией игры. Результат предыдущей игры невозможно сохранить. Это значит, что при повторном запуске приложения результат будет обнулен и игра начата заново.

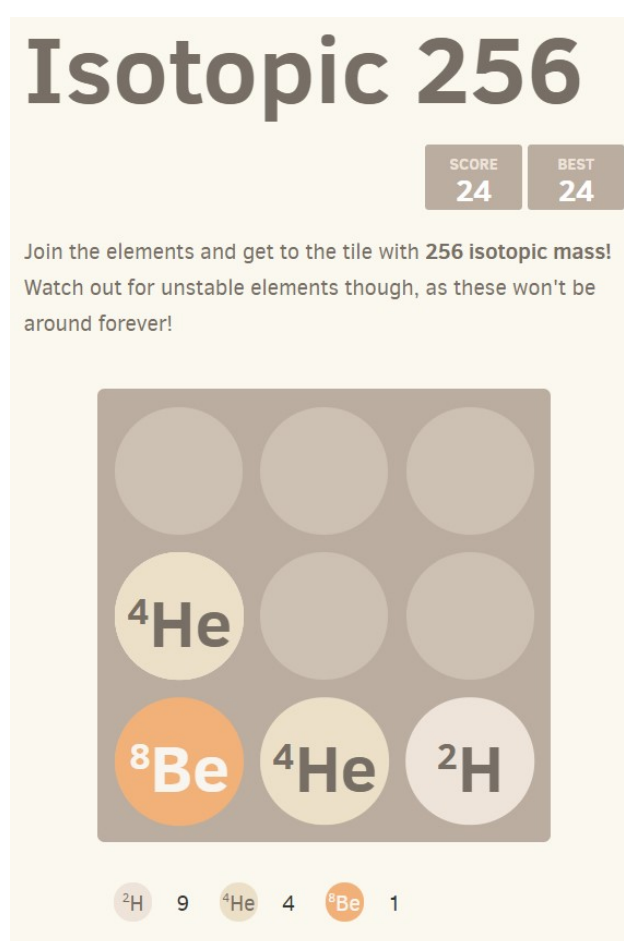


Рисунок 1.2 – Интерфейс Isotopic 256

1.2.3 2048+

Самая популярная интерпретация игры 2048 на ОС Windows представлена приложением 2048. Интерфейс игры максимально схож с интерфейсом оригинальной игры. Главным недостатком игры является

невозможность сохранить текущий прогресс. В данном приложении предусмотрено только лучшего значение очков. Учитывая то, что игровая сессия может длиться очень долгое время, это существенный критерий при выборе игры.

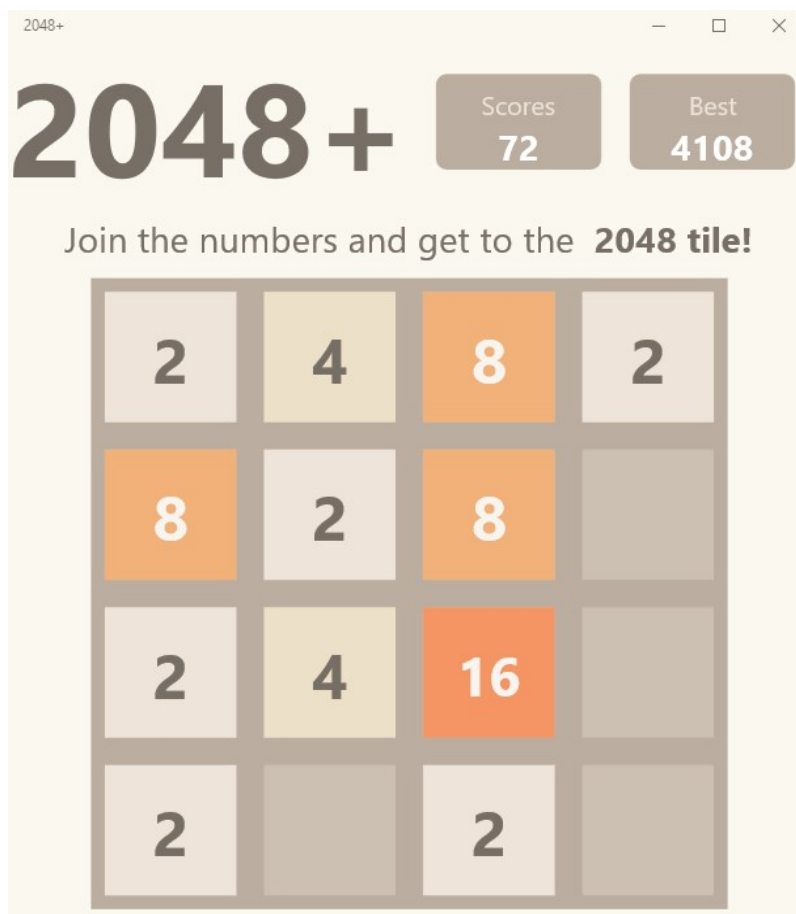


Рисунок 1.2 – Интерфейс игры 2048+

2 ПОСТАНОВКА ЗАДАЧИ

Реализовать прототип игрового приложения «2048».

При разработке программного средства выполнить следующие задачи:

- реализовать модуль, реализующий основную логику игры, механизм генерации и слияния плиток;
- выводить на экран результат текущей игры;
- выводить на экран лучший результат;
- реализовать модуль, реализующий графический интерфейс приложения;
- реализовать модуль, отвечающий за анимацию движения плиток;
- реализовать модуль, реализующий сохранение игрового процесса в файл для последующей загрузки и восстановления.

Для разработки программного средства выбраны:

- язык программирования C++, как наиболее быстрый и удобный язык для разработки игровых приложений;
- интерфейс WinAPI, позволяющий создавать полноценные графические приложения, содержащая огромный функционал по работе с графикой.

В качестве среды разработки была выбрана Microsoft Visual Studio 2019;

3 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

3.1 Окно в Windows

Окно – центральный элемент операционной системы Windows.

Основное окно обычно содержит рамку с заголовком, кнопки сворачивания и разворачивания, а также другие стандартные элементы пользовательского интерфейса. Рамка называется не клиентской областью окна, так как операционная система управляет этой частью окна. Область внутри рамки является клиентской областью. Это часть окна, которую управляет программа. [2]

3.2 Win32 API

Win32 API (также называемый Windows API) - это исходная платформа для собственных приложений Windows C / C ++, которым требуется прямой доступ к Windows и оборудованию. Он обеспечивает первоклассный опыт разработки, не зависящий от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Это делает Win32 API платформой выбора для приложений, которым требуется высочайший уровень производительности и прямой доступ к системному оборудованию. [2]

3.2.1 Родительские окна и окна владельцы

В случае элемента управления пользовательского интерфейса окно элемента управления называется дочерним элементом окна приложения. Окно приложения является родительским для окна управления. Родительское окно предоставляет систему координат, используемую для размещения дочернего окна. Наличие родительского окна влияет на аспекты внешнего вида окна. Например, дочернее окно обрезается таким образом, что никакая часть дочернего окна не может отображаться вне границ родительского окна. [2]

3.2.2 Дескрипторы окон

Окна являются объектами — они содержат как код, так и данные, но они не являются классами C++. Вместо этого программа ссылается на окно, используя значение, называемое дескриптор. По сути, это просто число, используемое операционной системой для обнаружения объекта. Вы можете представить Windows в виде большой таблицы всех созданных окон. Она использует эту таблицу для поиска окон по их дескрипторам. Для дескрипторов окон используется тип данных HWND. Дескрипторы окон возвращаются функциями, которые создают окна: CreateWindow и

CreateWindowEx. [2]

3.2.3 Координаты экрана и окна

Координаты измеряются в аппаратно-независимых пикселях.

В зависимости от задачи можно измерять координаты относительно экрана, относительно окна (включая рамку) или относительно клиентской области окна. Например, можно разместить окно на экране с помощью экранных координат, но нарисовать внутри окна с помощью клиентских координат. В каждом случае источник (0, 0) всегда находится в левом верхнем углу области. [2]

3.2.4 WinMain: точка входа приложения

Каждая программа Windows включает функцию точки входа, которая называется WinMain или wWinMain. Ниже приведена сигнатура для wWinMain.

```
int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE  
hPrevInstance, PWSTR pCmdLine, int nCmdShow);
```

При вызове wWinMain передаются четыре параметра:

- hInstance называется «обработчиком» экземпляра или «обработчиком» для модуля. Операционная система использует это значение для задания исполняемого файла (EXE) при его загрузке в память. Он необходим для определенных функций Windows, например для загрузки значков или растровых изображений.

- hPrevInstance использовался в 16-разрядной Windows, но теперь всегда равен нулю.

- pCmdLine содержит аргументы командной строки в виде строки Юникода

- nCmdShow – флаг, который определяет, будет ли главное окно приложения отображаться в обычном режиме, развернуто или отображаться обычным образом. [2]

3.2.5 Сообщения окна

Приложение графического пользовательского интерфейса должно реагировать на события от пользователя и из операционной системы.

К событиям пользователя относятся все способы взаимодействия с программой: щелчки мыши, клавиши, жесты с сенсорным экраном и т. д.

События операционной системы включают все «вне» программы, которая может повлиять на работу программы. например, пользователь может подключить новое аппаратное устройство или Windows может

перейти в режим пониженного энергопотребления (спящий или спящий режим).

Эти события могут возникать в любое время, пока программа выполняется, в почти в любом порядке. Чтобы структурировать программу, поток выполнения которой нельзя прогнозировать заранее, чтобы решить эту проблему, Windows использует модель передачи сообщений. Операционная система взаимодействует с окном приложения, передавая ему сообщения. Сообщение — это просто числовой код, обозначающий определенное событие. [2]

3.2.6 Цикл обработки сообщений

Приложение получит тысячи сообщений во время выполнения. Кроме того, приложение может иметь несколько окон, каждое из которых имеет собственную процедуру окна. Приложению требуется цикл для извлечения сообщений и их отправки в правильные окна.

Для каждого потока, создающего окно, операционная система создает очередь для сообщений окна. Эта очередь содержит сообщения для всех окон, созданных в этом потоке. Сама очередь скрыта от программы. Управлять очередью напрямую нельзя. Тем не менее можно извлечь сообщение из очереди, вызвав функцию GetMessage. [2]

3.2.7 Функция GetMessage

GetMessage удаляет первое сообщение из заголовка очереди. Если очередь пуста, функция блокируется до постановки в очередь другого сообщения. Тот факт, что функция блокируется, не сделает программу не отвечающей. Если сообщений нет, программа не будет выполнять никаких действий. Если необходимо выполнить фоновую обработку, можно создать дополнительные потоки, которые продолжают выполняться, а функция GetMessage ожидает другого сообщения.

Функция принимает 4 параметра. Первый параметр представляет адрес структуры сообщения MSG. Другие три параметра позволяют фильтровать сообщения, получаемые из очереди. Почти во всех случаях эти параметры будут заданы равными нулю. [2]

3.2.8 Выход из приложения

Если необходимо выйти из приложения и приостановить цикл обработки сообщений, вызовите функцию PostQuitMessage.

Функция PostQuitMessage помещает сообщение WM_Quit в очередь сообщений. WM_QUIT — это специальное сообщение: оно приводит к тому, что сообщение возвращает ноль, и сообщает о завершении цикла обработки сообщений.

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

4.1 Разработка программной архитектуры

Для обеспечения функционала программного средства необходимо разработать следующие модули:

- модуль, являющийся точкой входа в программу;
- модуль, реализующий управление и переключение между игровыми сценами;
- модуль, реализующий обработку нажатий клавиш;
- модуль, отвечающий за сохранение и загрузку игрового процесса;
- модуль, реализующий работу с изображениями (загрузка и отображение);

4.1.1 Класс MainGame

Данный модуль отвечает за инициализацию игрового процесса.

Метод `init` инициализирует список игровых сцен, и отвечает за загрузку главного меню.

Метод `render` отображает задний фон приложения.

Метод `imageInit` инициализирует список изображений и выполняет загрузку необходимых изображений.

4.1.2 Класс Scene

Данный модуль является родительским для всех модулей, реализующих игровые сцены приложения.

Загружает локально сохранённые настройки пользовательских серверов.

Метод `initd` инициализирует таймер, все необходимые для работы приложения модули.

Метод `release` вызывается во время завершения программы. Данный метод освобождает занятую память и останавливает запущенный при запуске таймер.

Метод `wndProc` отвечает за обработку входящих сообщений.

Метод `getMemDC` возвращает контекст устройства для отображения интерфейса.

4.1.3 Класс sceneManager

Функционал менеджера игровых сцен.

Модуль содержит список из всех загруженных игровых сцен и поле, содержащее адрес текущей сцены.

Метод `release` выгружает из памяти все загруженные игровые сцены и

очищает список, в котором они хранились.

Метод `update` обновляет данные текущей игровой сцены.

Метод `render` отвечает за отображение текущей сцены после обновления данных.

Метод `addScene` принимает в качестве параметров кодовое имя и класс игровой сцены, после чего инициализирует его и добавляет в конец списка всех игровых сцен пару ключ-значение, где ключ – кодовое имя, а значение – указатель на объект сцены.

4.1.4 Класс `ScoreManager`

Модуль отвечает за работу с данными игры.

Метод `saveData` принимает в качестве параметров имя файла, в который необходимо сохранить данные, и массив, который хранил данные игры.

Первый элемент массива хранит лучший результат.

Второй элемент массива хранит значение результата текущей игры.

Значение каждой клетки игрового поля содержится в следующих 16 элементах массива.

Метод `loadData` принимает в качестве параметров имя файла, из которого необходимо загрузить данные последней игры и массив размером 18 элементов, в которых эти данные будут записаны. Метод возвращает `false`, если во время загрузки произошла ошибка и `true`, если данные загружены корректно.

Метод `readString` принимает в качестве параметров дескриптор открытого файла и адрес строки, в который будет записана прочтенная из файла строка. Метод возвращает количество прочтенных символов.

4.1.5 Класс `Image`

Модуль `Image` служит для работы с изображением. Он обеспечивает инициализацию, вывод изображения в нескольких режимах и хранит данные о самом изображении.

Метод `init` инициализирует изображение без альфа-канала.

Метод `blendImageInit` создает дополнительную структуру `blendImage`, которая необходима для работы с изображением как с `BITMAP` с альфа-каналом.

Метод `render` выполняет загрузку изображения в указанный контекст устройства. Метод может принимать такие параметры как: координаты целевого устройства, начальные координаты исходного изображения, а также размер изображения.

Метод `renderCenter` принимает в качестве параметров координаты точки, являющейся центром выводимого изображения.

Метод `alphaRender` выводит изображение как `BITMAP` с альфа-

каналом. Уровень прозрачности изображения указывается в параметре `alpha`.
Метод `getWidth` возвращает ширину исходного изображения.
Метод `getHeight` возвращает высоту исходного изображения.
Метод `getHDC` возвращает контекст изображения.

4.1.6 Класс Timer

Класс `Timer` реализует механизм фиксирования кадров в секунду.

Модуль содержит следующие поля:

- `timeScale` – коэффициент умножения времени;
- `timeElapsed` – разница текущего времени и времени последнего вызова метода;
- `curTime` – текущее время;
- `lastTime` – время, прошедшее с прошлого вызова метода;

Метод `tick` получает фиксированное значение кадров в секунду и ожидает до того момента, пока `timeElapsed` не будет больше чем $1/\text{необходимое кадры в секунду}$.

5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

5.1 Тестирование игрового процесса

Таблица 5.1 – Тест 1

Тестовая ситуация	Начало новой игры
Исходный набор данных	Нажатие клавиши «New game»
Ожидаемый результат	Установка значения Score в 0, очистка игрового поля, генерация двух стартовых ячеек.
Полученный результат	

Таблица 5.2 – Тест 2

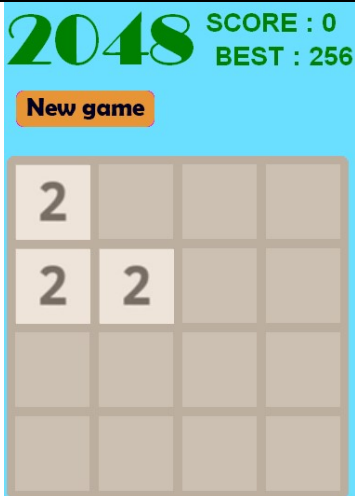
Тестовая ситуация	Смещение клеток в сторону
Исходный набор данных	Нажатие клавиши «Влево»
Ожидаемый результат	Сдвиг всех ячеек в сторону, соответствующую нажатой клавише. Генерация новой ячейки.
Полученный результат	

Таблица 5.3 – Тест 3

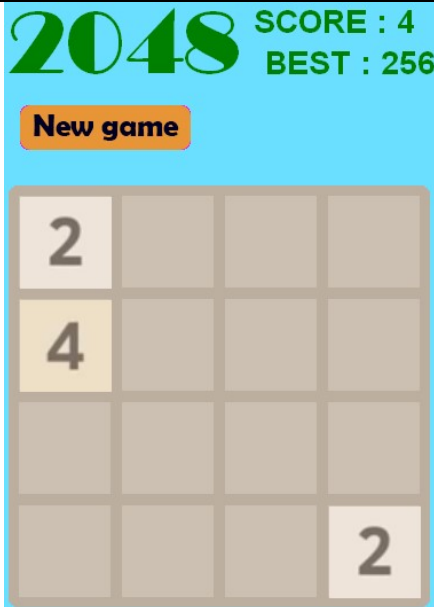

Тестовая ситуация	Слияние клеток
Исходный набор данных	Нажатие клавиши «Влево»
Ожидаемый результат	Генерация новой клетки номиналом 4, увеличение текущего счёта на 4.
Полученный результат	

Таблица 5.4 – Тест 4

Тестовая ситуация	Игра проиграна
Исходный набор данных	Отсутствие доступных ходов
Ожидаемый результат	Появление надписи Game Over и отсутствие реакции на нажатие клавиш
Полученный результат	

5.1 Тестирование модуля сохранения результата

Таблица 5.5 – Тест 5

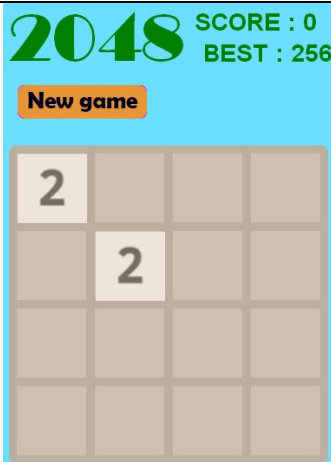
Тестовая ситуация	Начало игры без загрузки результата
Исходный набор данных	Отсутствие файла userdata.save
Ожидаемый результат	Инициализация новой игры. Значение текущего и лучшего счёта равно 0.
Полученный результат	

Таблица 5.6 – Тест 6

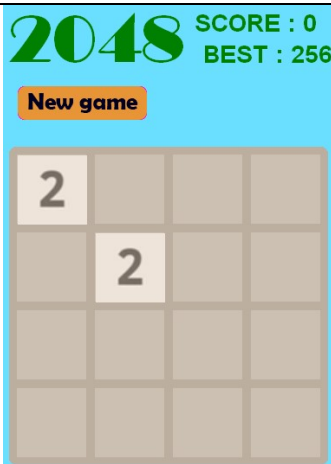
Тестовая ситуация	Начало игры без загрузки результата
Исходный набор данных	Поврежденный файла userdata.save
Ожидаемый результат	Инициализация новой игры. Значение текущего и лучшего счёта равно 0.
Полученный результат	

Таблица 5.7 – Тест 7

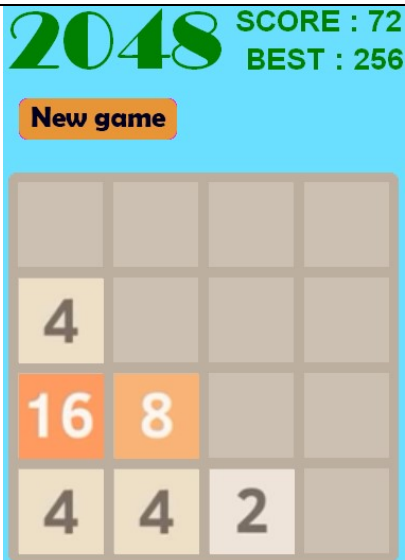

Тестовая ситуация	Начало игры без загрузки результата
Исходный набор данных	Корректный файл userdata.save
Ожидаемый результат	Загрузка сохранённой игры.
Полученный результат	

Таблица 5.8 – Тест 8

Тестовая ситуация	Загрузка проигранной сессии
Исходный набор данных	Корректный файл userdata.save с сохранённой проигранной сессией
Ожидаемый результат	Загрузка сохранённой игры.
Полученный результат	

6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

6.1 Установка приложения

Для установки игрового приложения необходимо распаковать архив 2048_Game.rar. Для начала работы с приложением требуется запустить файл 2048Game.exe.

6.2 Работа с программным средством

После запуска приложения на экране появится окно приложения. Данное окно представляет главное меню приложения и предлагает пользователю начать игру или выйти из приложения.



Рисунок 5.1 – Главное меню приложения

Для начала игры необходимо нажать на кнопку «START!», при нажатии на кнопку «EXIT» приложение закроется.

После нажатия кнопки «START!» загружается сцена с игровым полем, где пользователь может начать новую игру, или, если был найден файл сохранения, продолжить сохраненную.

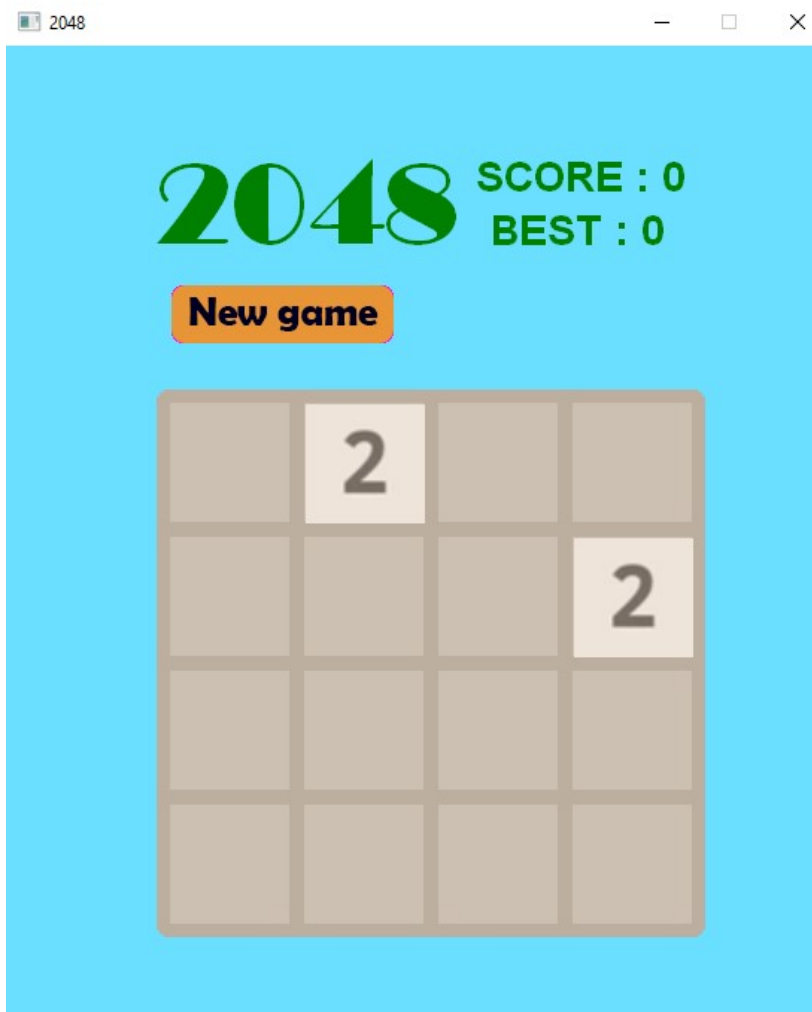


Рисунок 5.2 – Игровое поле

Управление плитками происходит нажатиями клавиш «вверх», «вниз», «вправо», «влево».

После нажатия клавиши, ячейки сдвигаются в соответствующую клавише сторону.

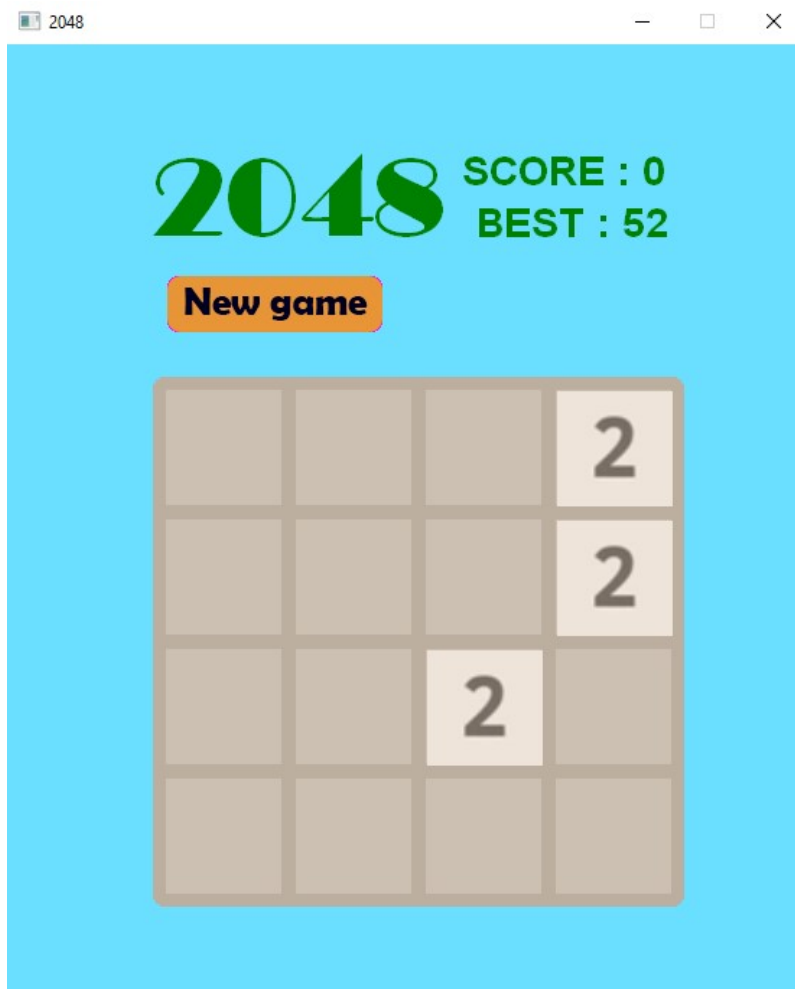


Рисунок 5.3 – Игровое поле после нажатия клавиши «вправо»

Если при сбрасывании две плитки одного номинала «налетают» одна на другую, то они превращаются в одну, номинал которой равен сумме соединившихся плиток. После каждого хода на свободной секции поля появляется новая плитка номиналом «2» или «4».

Игра заканчивается поражением, если после очередного хода невозможно совершить действие.

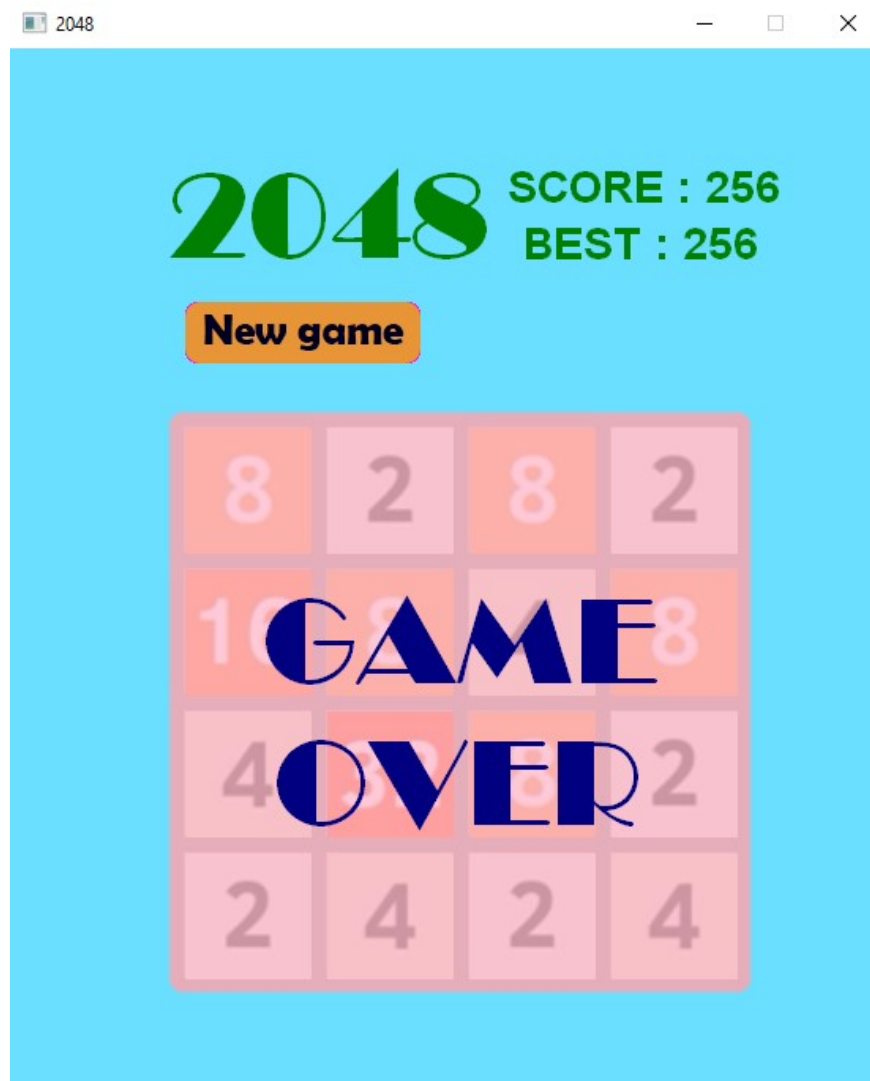


Рисунок 5.3 – Игра окончена

Для начала новой игры необходимо нажать на кнопку «New game».

Для выхода из игры необходимо нажать на крестик в меню приложения.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта был произведен анализ предметной области и реализовано программное средство. Согласно поставленной задаче, в данном приложении была реализована возможность 2048. Для комфортной игры был реализован механизм анимации движения плиток. Также была реализована возможность завершить игру и продолжить её через определённое время с сохранением результата.

Во время работы над проектом были изучены особенности работы с Windows API. Во время разработки проекта необходимо было изучить основы языка программирования C++ и ознакомиться с возможностями среды Microsoft Visual Studio 2019.

Во время проведения тестирования, программное средство успешно прошло все этапы и ситуации. Приложение показало быструю и корректную работу.

Приложение имеет необходимый набор функций для комфортной игры.

Код проекта легко читается и легко модифицируется для дальнейшего расширения функционала.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Wikipedia [Электронный ресурс]: Threes. URL: <https://ru.wikipedia.org/wiki/Threes>
- [2] Microsoft technical documentation [Электронный ресурс]: Microsoft technical documentation - Build desktop Windows apps using the Win32 API. URL: <https://docs.microsoft.com/en-us/windows/win32/>
- [3] Макс Шлее, Профессиональное программирование на C++ – М.: БХВ-Петербург, 2010. – 883 с.
- [4] Bjarne Stroustrup, The C++ Programming Language/ Addison-Wesley Professional; 4 edition (May 19, 2013)
- [5] Прата, Стивен Язык программирования C++. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2015.
- [6] Х. Дейтел, П. Дейтел. Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с. [7] Уилсон, С. Принципы проектирования и разработки программного обеспечения, учеб. курс. – СПб. : Русская Редакция, 2003 – 570 с.

ПРИЛОЖЕНИЕ А

Исходный код программы (Scene.cpp)

```
#include "stdafx.h"
#include "Scene.h"

HRESULT Scene::init()
{
    _hdc = GetDC(_hWnd);
    SetTimer(_hWnd, 1, 10, NULL);
    TIMEMANAGER->init();
    INPUT->init();
    IMAGEMANAGER->addImage("back", WINSIZEEX,
                           WINSIZEY);
    SCENEMANAGER->init();
    return S_OK;
}

void Scene::release()
{
    KillTimer(_hWnd, 1);
    TIMEMANAGER->release();
    TIMEMANAGER->releaseSingleton();
    INPUT->releaseSingleton();
    IMAGEMANAGER->releaseSingleton();
    SCENEMANAGER->release();
    SCENEMANAGER->releaseSingleton();
    ReleaseDC(_hWnd, _hdc);
}

void Scene::update()
{
    InvalidateRect(_hWnd, NULL, false);
    TIMEMANAGER->update(60);
}

void Scene::render()
{
}

LRESULT Scene::wndProc(HWND hWnd, UINT iMessage, WPARAM
wParam, LPARAM lParam)
{
    switch (iMessage)
    {
        case WM_TIMER:
            this->update();
            break;
    }
}
```

```

case WM_DESTROY:
    PostQuitMessage(0);
    break;
case WM_ERASEBKGND:
    return (LRESULT)1;
case WM_PAINT:
    this->render();
    break;
case WM_MOUSEMOVE:
    _ptMouse.x = LOWORD(lParam);
    _ptMouse.y = HIWORD(lParam);
    break;
case WM_KEYDOWN:
    switch (wParam)
    {
        case VK_ESCAPE:
            PostQuitMessage(0);
            break;
    }
    break;
}

return DefWindowProc(hWnd, iMessage, wParam, lParam);
}

```

Исходный код программы (Scene.h)

```
#include "stdafx.h"
#include "Timer.h"

#pragma comment(lib, "winmm.lib")

Timer::Timer() {}

Timer::~Timer() {}

HRESULT Timer::init()
{
    __int64 _periodFrequency;
    if
(QueryPerformanceFrequency((LARGE_INTEGER*)&_periodFrequency))
    {

        QueryPerformanceCounter((LARGE_INTEGER*)&_lastTime);

        _timeScale = 1.0f / _periodFrequency;
    }

    return S_OK;
}

void Timer::tick(float lockFPS)
{
    QueryPerformanceCounter((LARGE_INTEGER*)&_curTime)
;

    _timeElapsed = (_curTime - _lastTime) *
_timeScale;

    if (lockFPS > 0.0f)
    {
        while (_timeElapsed < 1.0f / lockFPS)
        {

            QueryPerformanceCounter((LARGE_INTEGER*)&_curTime);
            _timeElapsed = (_curTime - _lastTime) * _timeScale;
        }

        _lastTime = _curTime;
    }
}
```

Исходный код программы (ScoreManager.cpp)

```
#include "stdafx.h"
#include "ScoreManager.h"

void ScoreManager::saveData(string fileName, int
data[18])
{
    HANDLE file;
    DWORD write;
    file = CreateFile(fileName.c_str(), GENERIC_WRITE,
0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    for (int i = 0; i < 18; i++)
    {
        string str = to_string(data[i]).c_str();
        for (int j = 0; j < str.length(); j++)
        {
            str[j] += (j + 1) * (i + 1);
        }
        str += "\n"
;
        WriteFile(file, str.c_str(), str.size(),
&write, NULL);
    }
    CloseHandle(file);
}

BOOLEAN ScoreManager::loadData(string fileName, int
data[18])
{
    HANDLE file;
    LPDWORD lpFileSizeHigh = NULL;
    DWORD fileSize = 0;
    char str[128];
    file = CreateFile(fileName.c_str(), GENERIC_READ,
0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (file != INVALID_HANDLE_VALUE) {
        fileSize = GetFileSize(file, lpFileSizeHigh);
        if (fileSize != 0) {
            int lngth;
            for (int i = 0; i < 18; i++) {
                lngth = readString(file, str);
                if (lngth <= 0 && i < 18)
                {
                    CloseHandle(file);
                    return false;
                }
            }
            for (int j = 0; j < lngth; j++)
```

```

        {
            str[j] -= (j + 1) * (i + 1);
        }
        data[i] = stoi(str);
        fill(begin(str), end(str), 0);
    }
    CloseHandle(file);
    return true;
}
else
{
    CloseHandle(file);
    return false;
}
}
else return false;
}

int ScoreManager::readString(HANDLE file, char str[])
{
    DWORD read;
    char chr[1];
    int length = 0;
    do {
        ReadFile(file, chr, 1, &read, NULL);
        if (read != 0) {
            if (chr[0] != '\r' || chr[0] != '\n') {
                str[length] = chr[0];
                length++;
            }
        }
        else
            break;
    } while (chr[0] != '\r' && chr[0] != '\n');
    return length;
}

```

Исходный код программы (SceneManager.cpp)

```
#include "stdafx.h"
#include "SceneManager.h"
#include "Scene.h"

HRESULT SceneManager::init()
{
    return S_OK;
}

void SceneManager::release()
{
    for (auto scene : _mSceneList)
    {
        scene.second->release();
        SAFEDELETE(scene.second);
    }
}

void SceneManager::update()
{
    if (_currentScene)
    {
        _currentScene->update();
    }
}

void SceneManager::render()
{
    if (_currentScene)
    {
        _currentScene->render();
    }
}

Scene * SceneManager::addScene(string sceneName, Scene
* scene)
{
    if (!scene) return nullptr;

    _mSceneList.insert(make_pair(sceneName, scene));

    return scene;
}

HRESULT SceneManager::loadScene(string sceneName)
{

```

```

miSceneList find = _mSceneList.find(sceneName);
if (find == _mSceneList.end())
{
    return E_FAIL;
}

if (find->second == _currentScene)
{
    return E_FAIL;
}

if (SUCCEEDED(find->second->init()))
{
    _currentScene = find->second;
    return S_OK;
}

return E_NOTIMPL;
}

```