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

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные Системы и Системное Программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему:

«Игровое средство Пятнашки»

БГУИР КП 1-40 01 01 621 ПЗ

Студент: гр. 851006 Наденeнко И. П.

Руководитель: Жиденко А. Л.

Минск 2020

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ

_____
(подпись)
Лапицкая Н.В. 2020 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Наденeнко Игорю Павловичу

1. Тема работы «Игровое средство Пятнашки»
2. Срок сдачи студентом законченной работы 01.12.2020
3. Исходные данные к работе Документация по WinApi
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1.Анализ прототипов, литературных источников и формирование требований к проектироемому ПС;

2.Разработка алгоритма;

3. Разработка программного средства;

4.Тестирование, экспериментальные исследования и анализ полученных результатов;

5. Руководство пользователя программы;

Заключение, список литературы, ведомость, приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема программы на А1

6. Консультант по курсовому проекту Жиденко А. Л.

7. Дата выдачи задания 05.09.2020 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1 к 20.09.2020 – 15 % готовности работы;

разделы 2, 3 к 13.10.2020 – 30 % готовности работы;

раздел 4 к 02.11.2020 – 60 % готовности работы;

раздел 5 к 26.11.2020 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 01.12.2020 – 100 % готовности работы. Защита курсового проекта с 01.12 по 22.12 2020 г.

РУКОВОДИТЕЛЬ_____ Жиденко А. Л.

(подпись)

Задание принял к исполнению Наденко И.П. _____ 05.09.2020 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Пятнашки (игра в 15, такен) – головоломка, придуманная в 1878 году Ноем Чепмэном. Она занимает третье место в мире по популярности после паззлов и кубика Рубика. Головоломка представляет собой набор одинаковых квадратных костяшек с нанесёнными на них числами от 1 до 15, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры – перемещая костяшки по коробке, добиться упорядочивания их по номерам.

Существует большое количество разнообразных вариантов головоломки. Они могут отличаться размерностью игрового поля, которая может достигать 7×7 или не иметь равных сторон: например, быть 3×5. В некоторых вариантах вместо упорядочения чисел может ставиться цель восстановить некоторое изображение. Интересен также вариант под названием «магический квадрат»: в нём необходимо разложить костяшки с числами так, чтобы сумма чисел каждого ряда была одинаковой.

У пятнашек нашлось прикладное применение в информатике. С 1960-х годов разные варианты игры используют в исследованиях в области искусственного интеллекта. В частности, на них испытываются и сравниваются алгоритмы поиска, при различных параметрах, влияющих на число посещённых в процессе поиска конфигураций головоломки. В таких исследованиях такие конфигурации, то есть, частные случай размещения частей головоломки, представляют в качестве вершины графа. В подобных исследованиях часто используются головоломки размеров 3×3, 4×4, 5×5, 6×6, 2×7, 3×5. [1]

Можно перечислить несколько основных причин того, почему пятнашки используются для изучения алгоритмов поиска:

– Количество состояний классических пятнашек является доступным для анализа, но всё же достаточно большим, чтобы представлять интерес и сравнивать разнообразные исследовательские подходы;

– На данный момент не известен ни один алгоритм, находящий кратчайшее решение для обобщённых пятнашек $n \times n$ за конечное время;

– Задача поиска кратчайшего решения для пятнашек проста для понимания и программного манипулирования. Головоломка поддаётся описанию с помощью небольшого и чётко определённого набора простых правил;

– Для моделирования головоломки не требуется передачи семантических тонкостей, присущих более сложным предметным областям.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

Существует достаточно небольшое количество программ для игры в пятнашки для операционной системы Windows. Большинство приложений доступны для использования только в глобальной сети Интернет и представляют собой веб-приложения. Большое же количество реализаций пятнашек существует на мобильных платформах Android и IOS. Каждое приложение имеет свои преимущества и недостатки в сравнении с остальными.

1.1.1 Magic Square 15 Puzzle

Игра бесплатно распространяется через магазин приложений Microsoft. Представляет собой простую реализацию классической головоломки с хорошей графикой и качественным музыкальным сопровождением.

Плюсы:

– Хорошая производительность на слабых процессорах;

– Бесплатное распространение;

– Возможность выбрать заранее заготовленное состояние поля;

– Имеется таймер, отсчитывающий время решения головоломки.

Минусы:

– Доступна только классическая размерность 4 × 4;

– Доступен только классический режим с использованием чисел;

– Возможно использование только в полноэкранном режиме.

Внешний вид программы представлен на рисунке 1.1.

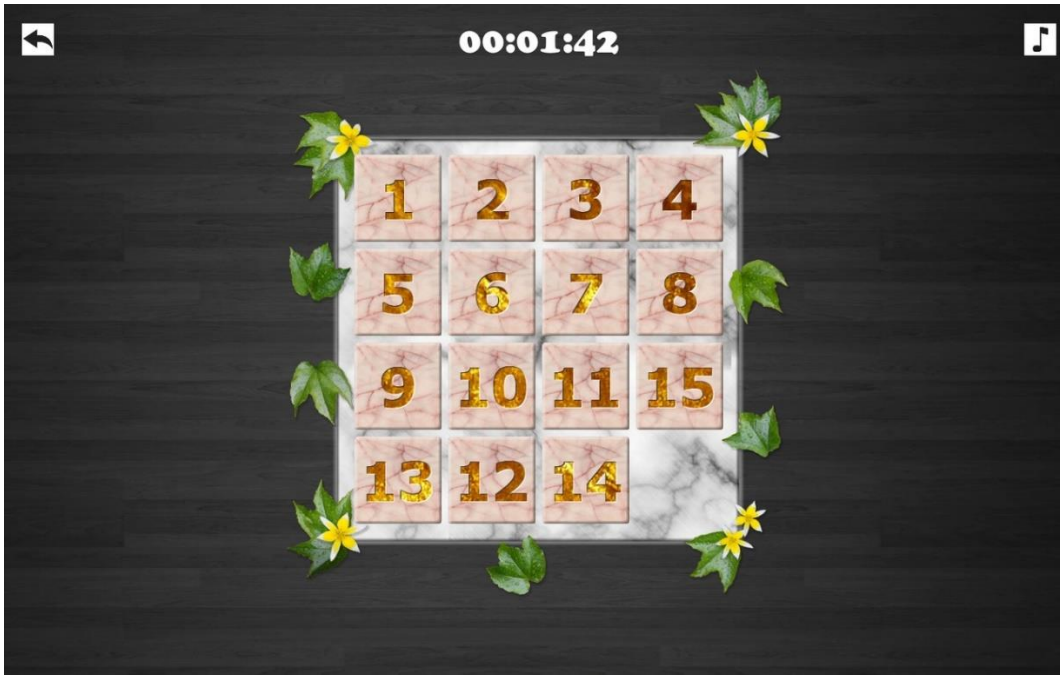

Рисунок 1.1 – программа Magic Square Fifteen Puzzle

1.1.2 Fifteen Puzzle Game

Приложение доступно в сети Интернет. Как и прошлая программа представляет собой реализацию классической версии головоломки в ещё более минималистичном исполнении.[2]

Плюсы:

– Засчет того, что данная реализация игры является веб-приложением, она также является кроссплатформенной и может быть запущена на всех современных операционных системах, в том числе и мобильных;

– Имеется таймер, отсчитывающий время решения головоломки, а также возможность поставить его на паузу;

– Имеется счетчик количества ходов, за которые головоломка решается.

Минусы:

– Доступна только классическая размерность 4 × 4;

– Доступен только классический режим с использованием чисел;

– Использование требует постоянного подключения к сети интернет;

Внешний вид приложения представлен на рисунке 1.2.

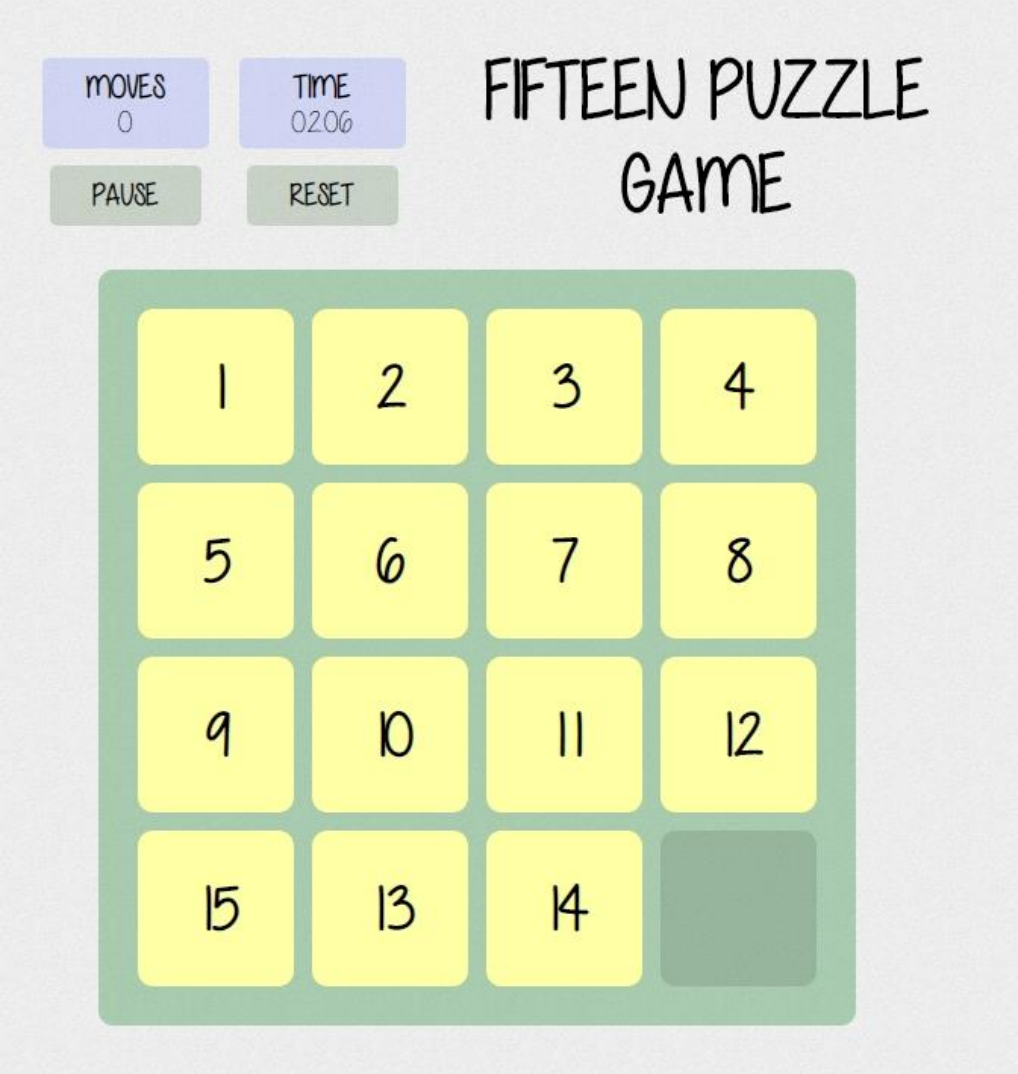

Рисунок 1.2 – программа Fifteen Puzzle Game

1.1.3 15!

Приложение доступно для загрузки в магазине приложений App Store на мобильной операционной системе IOS. Отличительной особенностью приложения является наличие режима сборки изображений из предустановленного набора.

Плюсы:

– Хорошо проработанный в области визуального оформления интерфейс пользователя;

– Игра разбита на уровни сложности, которые открываются по мере прохождения предыдущих. С каждым уровнем добавляются определенные нововведения в игровой процесс такие как, например, сборка чисел, кратных четырем или нечетных чисел от меньшего к большему. Это добавляет разнообразия игровому процессу;

– Возможность через приложение делиться своим результатом через социальные сети и мессенджеры.

– Имеется режим игры с использованием частей изображения.

Минусы:

– Недоступность для настольных операционных систем Windows, Linux и Mac OS;

– Невозможность загрузки своих изображений в игру, доступно использование только ограниченного числа предустановленных.

Внешний вид приложения представлен на рисунке 1.3.

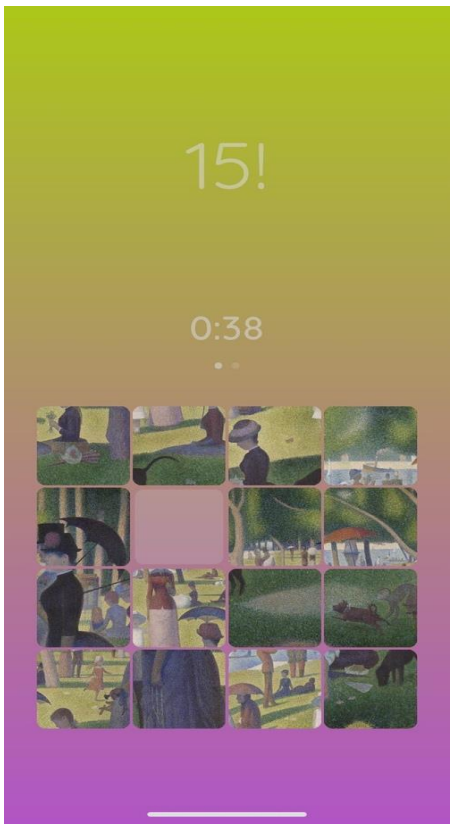

Рисунок 1.3 – программа 15!

1.2 Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Пятнашки».

Изучив аналоги моего программного средства были определены следующие требования к проекту:

– Классический режим игры в пятнашки с размерностью поля 4×4 и использованием чисел 1 – 15;

– Изменение размеров игрового поля, с поддержкой их до 7×7, что является оптимальным размером с точки зрения сложности разрешения головоломки и комфортного зрительного распознавания элементов игрового поля;

– Режим игры с использованием частей изображения в качестве элементов игрового поля. Целью пользователя будет восстановить изначальное изображение;

– Загрузка пользователем любого изображения для использования в игре;

– Сохранение игрового прогресса на диск с возможностью последующего восстановления;

– Масштабирование игрового поля пропорционально увеличению или уменьшению размеров окна приложения.

Главной задачей ставится разработать простое в использовании и функциональное приложение, которое будет обеспечивать комфортное и быстрое взаимодействие с пользователем, а также хороший уровень производительности.

Для разработки программного средства будет использоваться язык программирования C++, библиотека GDI+ и операционная система Windows 8.1.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Структура программы

Составляющие программы:
– Файл с исходным кодом main.cpp;
– Заголовочный файл Game.h;
– Файл с исходным кодом Game.cpp.

2.2 Файл Main.cpp

В данном файле создается оконный класс программы, с последующим созданием окна на его основе. Прописывается поддержка стандарта кодирования символов Unicode. Также в данном файле инициализируется графическая библиотека GDI+.

2.2.1 Создание окна

Для создания окна в WinAPI были произведены следующие действия:

– Создана функция WinMain(). Это основная функция программы, принимающая в качестве аргументов дескриптор экземпляра приложения, переменную для запуска окна в режиме командной строки и стиль отображения окна. Данная функция – аналог main() в консольных приложениях;

– Описан и зарегистрирован оконный класс WNDCLASSEX. Этот класс является шаблоном для создания окон, который можно будет использовать в дальнейшем для создания нескольких окон с одинаковыми конфигурациями. В его полях хранится адрес функции обработки сообщений, приходящих окну, адрес иконки окна, курсора и цвет заднего фона;

– Создан дескриптор окна, который будет использован для идентификации окна в программе. Этот дескриптор примет значение, которое вернет функция CreateWindow(). Эта функция принимает в качестве аргументов созданный оконный класс, имя, стили, начальное положение окна, его ширину, высоту, дескрипторы родительского окна, меню и экземпляра приложения;

– Создан цикл для обработки сообщений. Программа получает вызовом функции GetMessage(), принимающей в качестве аргументов адрес структуры MSG, используемой для хранения информации об очередном сообщении, дескриптор окна программы и фильтры для отбора сообщений, которые в моей программе принимают нулевые значения для приёма всех типов сообщений. Условием выхода из цикла обработки сообщений является получение сообщения WM_QUIT. При его получении вместе с окончанием цикла закончится и наша программа. Для интерпретации текущего сообщения в теле цикла используются функции TranslateMessage() и DispatchMessage(). Работа цикла продемонстрирована на рисунке 2.1; [3]

После создания окна необходимо создать функцию обработки сообщений, которые будут приходить ему.

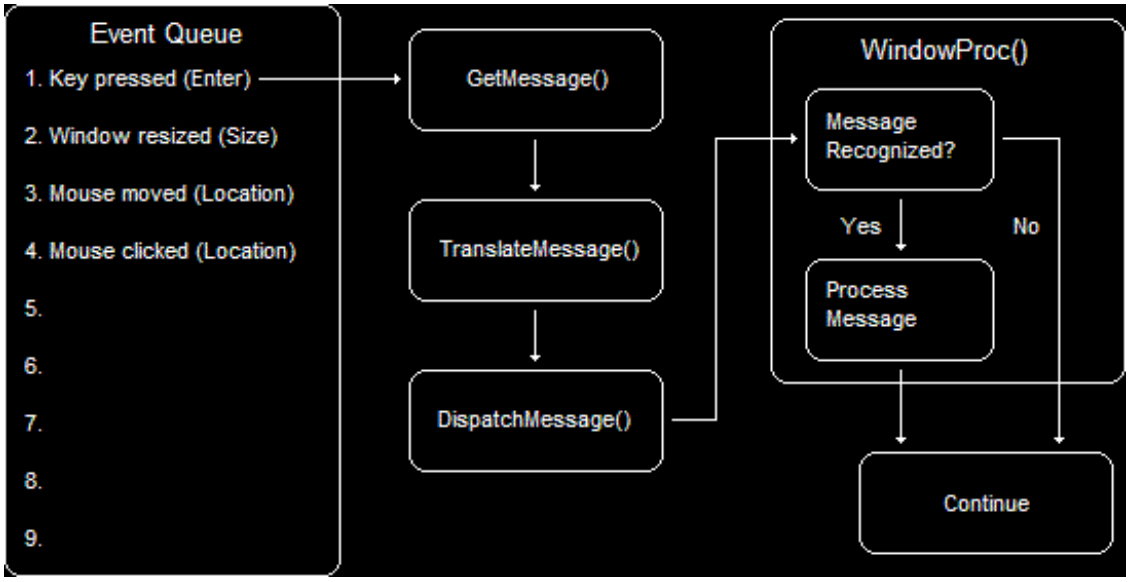

Рисунок 2.1 – Цикл обработки сообщений Windows

2.2.2 Библиотека GDI+

Для работы с изображениями в программе используется стандартная библиотека GDI+ от Microsoft. Она предназначена для замены устаревшего программного интерфейса GDI. Ее использование позволило облегчить переход на 64-битные платформы и максимально полно использовать изобразительные способности и вычислительные мощности современных компьютеров. Она представляет собой набор из 600 функций, реализованных в файле gdiplus.dll. Эти функции обёрнуты в 40 классов языка C++. Для подключения библиотеки необходимо вызвать функцию GdiplusStartup(), после окончания работы с ней – GdiplusShutdown().

Для хранения изображений в библиотеке GDI+ предназначен класс Image**.** Его потомки, классы Bitmap и Metafile, реализуют, соответственно, функциональность для работы с растровыми и векторными изображениями. В программе поддерживается работа с растровыми изображениями, поэтому для их хранения был выбран класс Bitmap.

2.3 Файлы Draw.h и Draw.cpp

Данные файлы содержат функцию обработки сообщений, которая в зависимости от типа сообщения вызывает различные функции, содержащие работы программы.

2.3.1 Функция обработки сообщений

Для взаимодействия с пользователем программа использует различные типы сообщений WinAPI. При вызове функции DispatchMessage() будет вызвана функция WindowProc, которая принимает в качестве аргументов

дескриптор окна, структуру сообщения и параметры, в которых может содержаться дополнительная информация передаваемая с вызовом. Данная функция определяет тип пришедшего сообщения и вызывает в зависимости от этого нужный код. Данные типы сообщений обрабатываются в программе:

– WM_PAINT – отправляется, когда приложение делает запрос на перерисовку окна или его области. При обработке данного сообщения в программе вызывается функция BeginPaint(), подготавливающая окно к закрашиванию. Данная функция обязательно используется в паре с функцией EndPaint().

– WM_COMMAND – отправляется при выборе пункта меню или при посылке элементом управления, например кнопкой, уведомления о взаимодействии с ним.

– WM_TIMER – отправляется при истечении времени у таймера. Является сообщением с низким приоритетом. Функция GetMessage() помещает это сообщение в очередь только тогда, когда, никакие другие высокоприоритетные сообщения не находятся в очереди сообщений программы.

– WM_ERASEBKGND – отправляется тогда, когда фон окна должен быть стерт, например, если окно необходимо перерисовать. Сообщение отправляется, чтобы подготовить ставшую недействительной часть окна для окрашивания. Приложение должно возвратить не нуль в ответ на данное сообщение, если оно обрабатывает его и стирает фон, это указывает, что никакое дополнительное стирание не требуется.

– WM_CREATE – отправляется при создании окна. Код, находящийся в обработчике этого сообщения используется для инициализации элементов окна.

– WM_GETMINMAXINFO – отправляется, когда размер или позиция окна собираются измениться. Программа использует это сообщение, чтобы окно не меняло свои размеры меньше минимально указанных.

– WM_DESTROY – отправляется тогда, когда окно разрушается. Это сообщение отправляется сначала разрушаемому окну, а затем дочерним окнам, если таковые имеются. Программа использует в обработчике данного сообщения функцию PostQuitMessage(). Данная функция помещает в очередь сообщение WM_QUIT и указывает системе, что потоку требуется прекратить свою работу.

– WM_DRAWITEM – отправляется окну, являющемуся владельцем созданной со стилем BS_OWNERDRAW кнопки, когда внешний вид кнопки изменился. Данные кнопки являются важной частью интерфейса программы и представляют собой части изображения, которое необходимо собрать при решении пятнашек.

2.3.2 Отображение игрового поля

Программа поддерживает два режима игры: классический и режим игры с загрузкой пользователем изображения.

– Классический режим

По умолчанию, при создании окна и обработке сообщения WM_CREATE создаётся поле в данном режиме размером 4 × 4 и заполненное числами 1-15. Игровое поле представляет собой стандартные элементы управления WinAPI кнопки, с нанесенными на них числами. Для того, чтобы создать кнопки используется функция CreateWindow(), со строковым литералом «Button» в качестве первого аргумента. Для того, чтобы кнопки отображали текст, в функции CreateWindow() необходимо указать стиль BS_PUSHBUTTON наряду с другими стилями. Классический режим игры показан на рисунке 2.2.

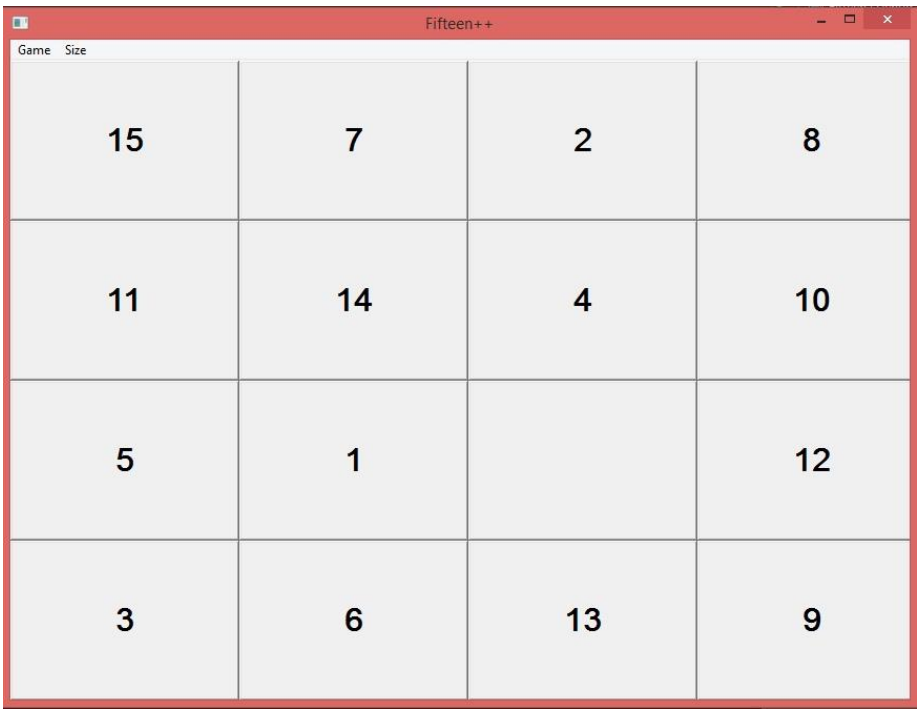

Рисунок 2.2 – Классический режим игры

– Режим с загрузкой изображения

С помощью диалогового окна пользователь программы может открыть изображение с расширением .jpeg, .bmp или .png. На основе данного изображения создается объект класса Gdiplus::Bitmap, который далее с помощью функции Bitmap::Clone() разбивается на необходимое количество отдельных объектов, содержащих фрагменты исходного. Эти фрагменты помещаются на кнопки, при создании которых использовался стиль BS_OWNERDRAW. Данный стиль позволяет отображать изображение на кнопке, при этом предоставляет возможность задавать логику отрисовки этого изображения с помощью посылки сообщения WM_DRAWITEM. В обработчике данного сообщения прописана логика отрисовки растрового изображения на кнопке, применяя метод двойной буферизации.

Двойная буферизация – метод отрисовки, который обеспечивает возможность отображать изображение и одновременно с этим подготавливать новое в оперативной памяти. Этот метод использует дополнительный буфер,

в котором изображение предварительно отрисовывается, а затем копируется на экран. Это позволяет избежать видимости на экране результата постоянной перерисовки изображения, например при изменении его размеров и последующим мерцанием заднего фона.

В WinAPI данный метод может быть реализован созданием двух контекстов устройств, окна и памяти. Сначала с помощью функции CreateCompatibleDC() создается контекст памяти, который совместим с контекстом окна, а затем с помощью функции BitBlt() отображается на окне, копируясь из памяти. В случае рисования игрового поля в программе изображение отрисовывается в контексте текущей кнопки. Режим игры с загрузкой изображения показан на рисунке 2.3. [4]

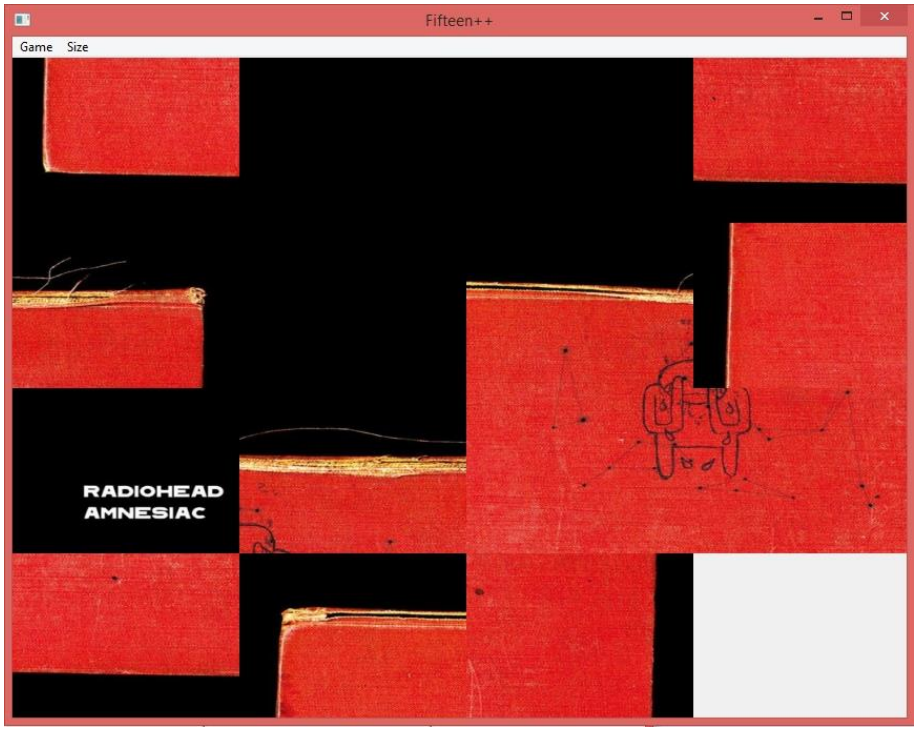

Рисунок 2.3 – Режим игры с загрузкой изображения

2.3.3 Внутренняя реализация игрового поля

В программе игровая клетка представлена структурой Cell. Структура имеет следующие поля:

– Id – идентификатор клетки, представленный целым числом. Нумерация клеток начинается с единицы, пустая клетка имеет id равный нулю. Данное поле позволяет не только явно идентифицировать отдельную клетку, но и имеют практическое применение, например они представляют числа на клетках в классическом режиме игры, а также используются в алгоритме проверки наличия решения;

– Btn – дескриптор кнопки, которой клетка визуально представлена на окне;

– IsEmpty – поле логического типа, для непустой клетки равно false, для пустой – true;

– Bitmap – дескриптор части изображения, которая нанесена на данную клетку в режиме с загрузкой изображения.

В качестве структуры хранения данных для программы была выбрана динамическая матрица на основе шаблонного класса языка C++ std::vector. Этот класс построен на основе динамического массива и может хранить в себе шаблонный тип. Для реализации матрицы на его основе можно создать std::vector с типом std::vector.

В реализации программы std::vector обладает преимуществами в скорости работы над другими структурами данных. Матрица клеток заполняется один раз при инициализации игры, впоследствии в неё не добавляется новых элементов и из неё не удаляются старые, элементы матрицы лишь меняются местами друг с другом. Следовательно, самой частой операцией с матрицей клеток будет получение её элемента по индексу. Как видно на рисунке 2.4, эта операция при использовании массива, на котором построен std::vector имеет сложность O(1), в сравнении со связанным списком, где она имеет линейную сложность. [5]

	Array	Linked List
Cost of accessing elements	O(1)	O(n)
Insert/Remove from beginning	O(n)	O(1)
Insert/Remove from end	O(1)	O(n)
Insert/Remove from middle	O(n)	O(n)

Рисунок 2.4 – Сравнение сложности операций в массиве и связанном списке

2.3.4 Генерация игрового поля

Главной особенностью игры пятнашки является то, что ровно 50% случайно полученных раскладок игрового поля могут быть решены. Например раскладка, показанная на рисунке 2.5, демонстрирует знаменитую нерешаемую раскладку головоломки.

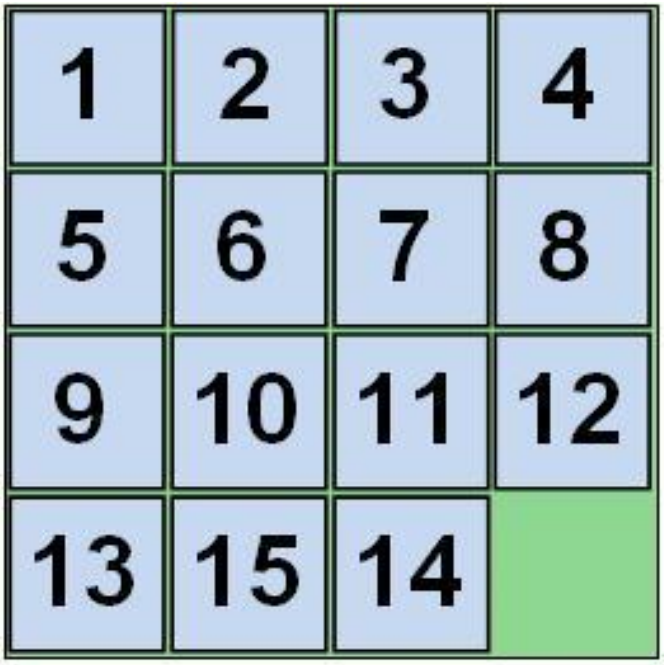

Рисунок 2.5 – Пример нерешаемых пятнашек

В результате математических исследований было выяснено условие наличия решения пятнашек. [6] Пусть клетка с числом i расположена до m (если считать слева - направо и сверху - вниз) клеток с числами меньшими i. Будем считать a_i = m. Также введем число e – номер ряда пустой клетки (считая с единицы). Для поля размерности n, где непустых клеток будет $k = n^2 - 1$ посчитаем:

$$N = \sum_{i=1}^{k} a_i + e,$$

Если получившееся число N является нечетным, то решения не существует. В программе данное условие должно быть учтено при генерации раскладки поля.

Генерация поля представляет собой перемешивание случайных элементов матрицы между собой достаточно большое количество раз и последующую проверку условия существования решения. Если условие выполняется, происходит отрисовка игрового поля.

2.3.5 Изменение размеров игрового поля

Программа поддерживает изменение размеров игрового поля от 3 × 3 до 7 × 7. Данная функция доступна пользователю через раздел меню Size. Она перестраивает игровое поле с новыми размерами и подготавливает новую игру с учётом условия существования решения. На рисунке 2.6 представлен пример изменения размера поля игры в режиме загрузки изображения до 7 × 7, а на рисунке 2.7 изменение поля в классическом режиме до 3 × 3.

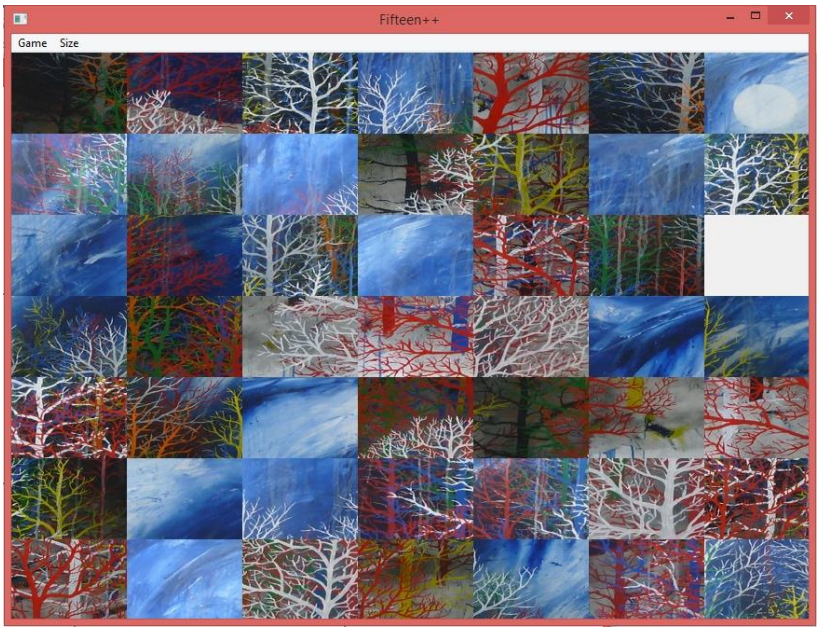

Рисунок 2.6 – Пример изменения размера игрового поля до 7 × 7

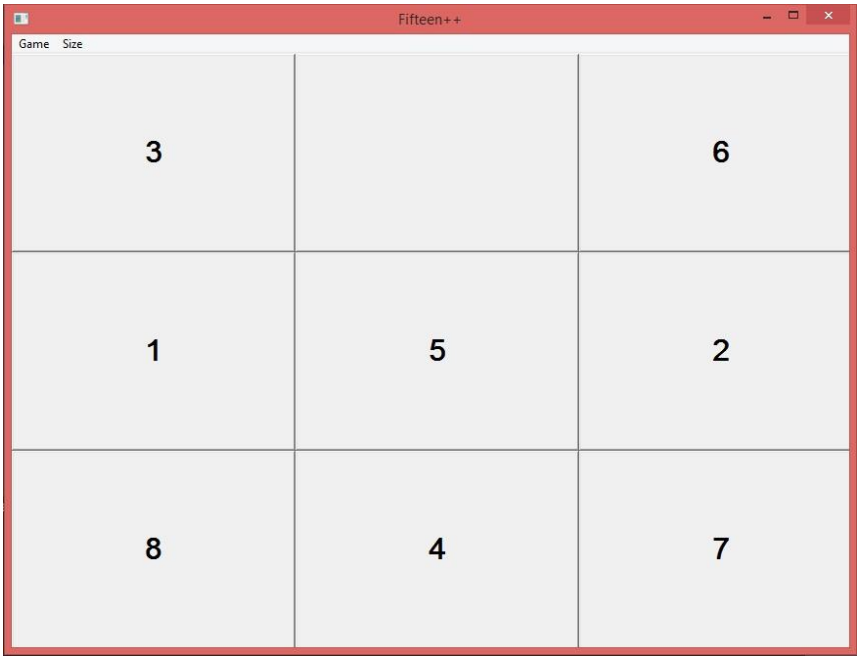

Рисунок 2.7 – Пример изменения размера игрового поля до 3 × 3

2.3.6 Сохранение прогресса

При игре в пятнашки часто может понадобиться сохранить прогресс партии для последующего завершения в другое время. При сохранении игры в классическом режиме создается файл с расширением .dat, куда записываются следующие данные в установленном порядке через пробел:

1) Размерность поля;
2) Индекс столбца пустой клетки;
3) Индекс строки пустой клетки;

4) Идентификаторы непустых клеток.

При сохранении игры в режиме с загрузкой изображения кроме аналогичного файла сохраняется файл с изображением в формате .png.

Пример сохранения и восстановления прогресса показан на рисунках 2.8, 2.9 и 2.10.

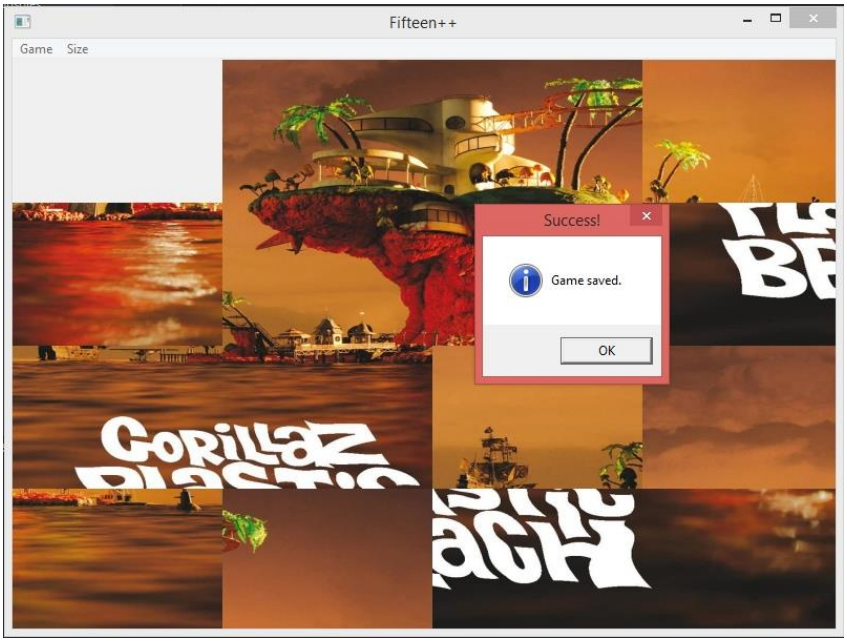

Рисунок 2.8 – Сохранение прогресса

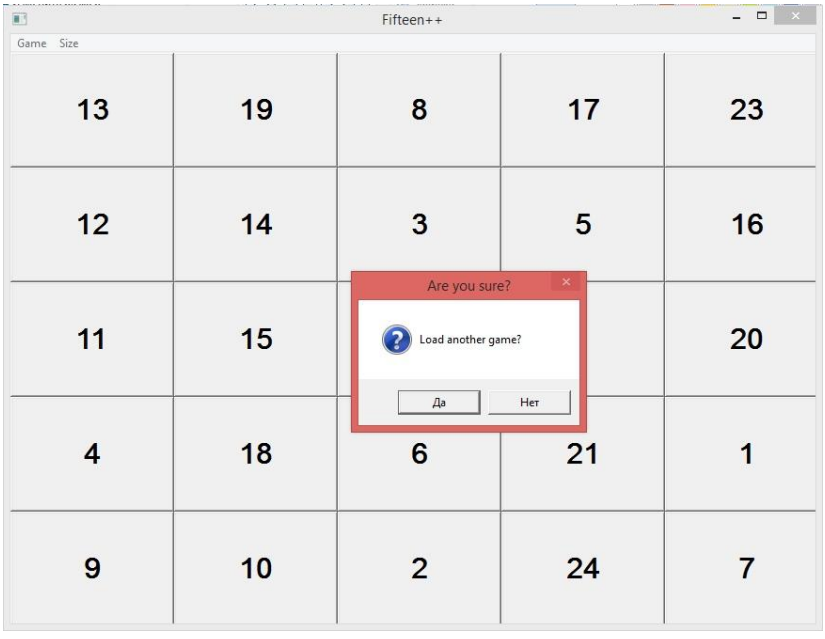

Рисунок 2.9 – Загрузка сохранения

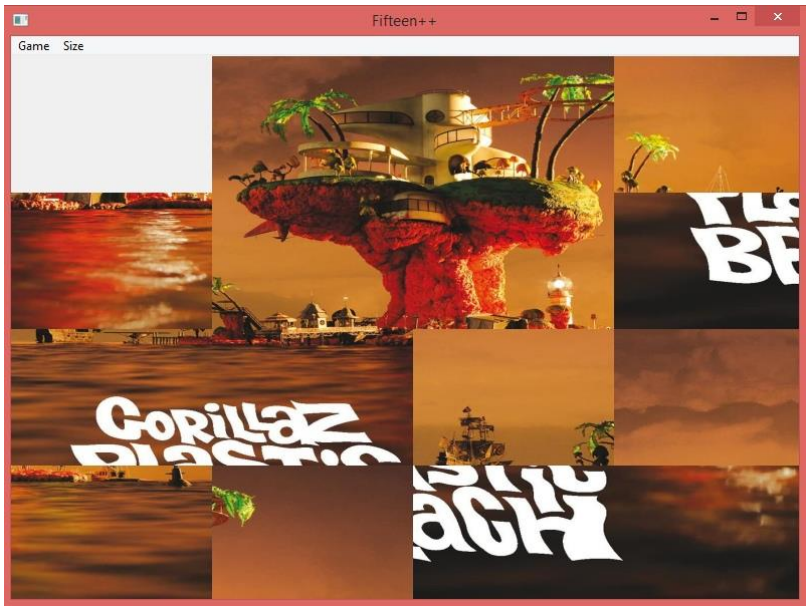

Рисунок 2.10 – Результат загрузки сохранения

2.3.7 Окончание игры

После каждого хода игрока (перемещения непустой клетки на место пустой) программа проверяет, не является ли положение клеток на игровом поле выигрышным. Для этого сначала проверяется, находится ли в правой нижней клетке пустая, если это условие выполнено проверяется расположение всех остальных клеток. Если их идентификаторы расположены по порядку, выводится сообщение с поздравлением с выигрышем и предложением начать новую игру, что можно увидеть на рисунке 2.11.

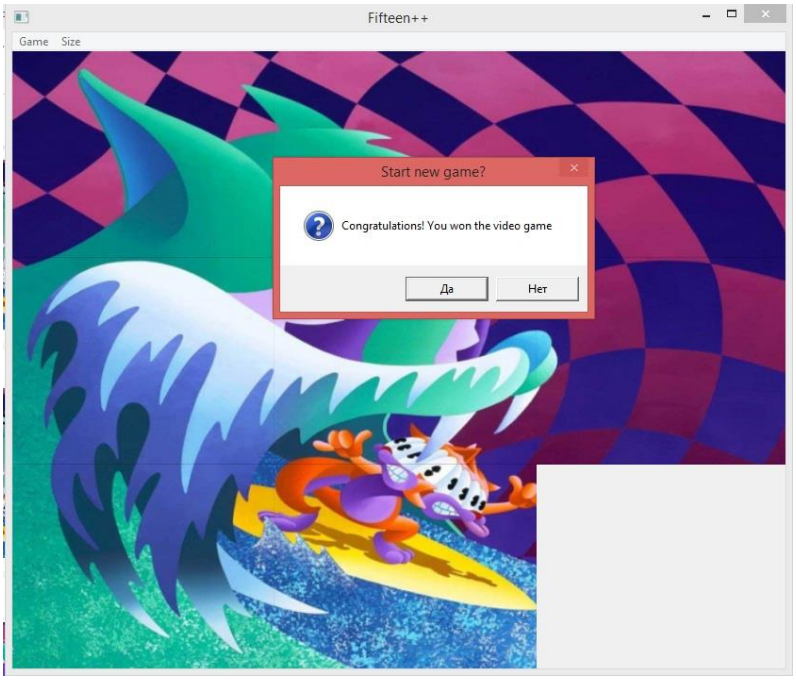

Рисунок 2.11 – Сообщение об успешном окончании игры

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения не было выявлено недостатков программного средства. Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

№	Тестовые случаи	Ожидаемый результат	Полученный результат
1.	Запуск программы	Успешный запуск программы и начало игры в классическом режиме размерностью 4 × 4	Тест пройден
2.	Изменение размера поля в классическом режиме	Успешное изменение размера поля на указанный	Тест пройден
3.	Запуск новой игры в классическом режиме	Успешное начало игры в классическом режиме размерностью 4 × 4	Тест пройден
4.	Запуск новой игры в режиме с загрузкой изображения	Успешное открытие диалогового окна открытия файла	Тест пройден
5.	Выбор изображения в диалоговом окне	Успешное начало игры в режиме с загрузкой изображения размерностью 4 × 4	Тест пройден
6.	Запуск новой игры в режиме с загрузкой изображения размером 35 МБ	Успешное начало игры в режиме с загрузкой изображения размерностью 4 × 4	Тест пройден
7.	Закрытие диалогового окна без выбора изображения	Вывод сообщения об ошибке открытия изображения	Тест пройден
8.	Изменение размера поля в режиме с загрузкой изображения	Успешное изменение размера поля на указанный	Тест пройден
9.	Масштабирование окна	Успешная перестройка игрового поля в соответствии с новыми размерами окна	Тест пройден

Продолжение таблицы 3.1

10.	Нажатие на кнопку полноэкранного режима	Успешная перестройка игрового поля в соответствии с новыми размерами окна	Тест пройден
11.	Сохранение прогресса игры	Успешное создание файлов с сохраненным прогрессом	Тест пройден
12.	Загрузка прогресса игры	Успешное отображение прогресса сохраненной игры	Тест пройден
13.	Нажатие на клетку по соседству с пустой	Успешная смена местами клетки с пустой	Тест пройден
14.	Нажатие на клетку, не находящуюся по соседству с пустой	Корректная обработка нажатия, отсутствие изменений на поле	Тест пройден
15	Упорядочение клеток на поле в соответствии с правилами победы	Отображение сообщения о победе с предложением начать новую игру	Тест пройден
16	Попытка загрузки прогресса игры при отсутствии сохранений	Отображение сообщения об отсутствии сохранения	Тест пройден

Разработка программы велась с использованием системы контроля версий GitHub, позволившей сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

Путем тщательной проверки тестами, было выявлено, что ошибок в работе программы нет.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Основные требования для запуска

Минимальные требования для запуска:
– ОС: Версия Microsoft Windows от 8 и выше;
– Процессор: не менее 1 ГГц;
– RAM: От 1 Гб;
– Место на диске: от 3 Мб свободного места.

Исходя из данных требований следует, что разработанная программа может запускаться практически на любом компьютере.

4.2 Руководство по установке

Установка программы не требуется, т.к она является переносимым приложением.

Переносимое приложение — программное обеспечение, которое для своего запуска не требует процедуры установки и может полностью храниться на съёмных носителях информации, что позволяет использовать данное ПО на многих компьютерах.

4.3 Руководство использования

При запуске программы отображается начало игры в классическом режиме размерностью 4 × 4, как видно на рисунке 4.1. При нажатии на клетку, стоящую рядом с пустой, клетка поменяется местами с ней. Таким образом и собирается конечная раскладка игры, в которой числа на клетках расположены по порядку сверху-вниз и слева-направо. Пример перемещения клетки с числом показан на рисунках 4.1 и 4.2.

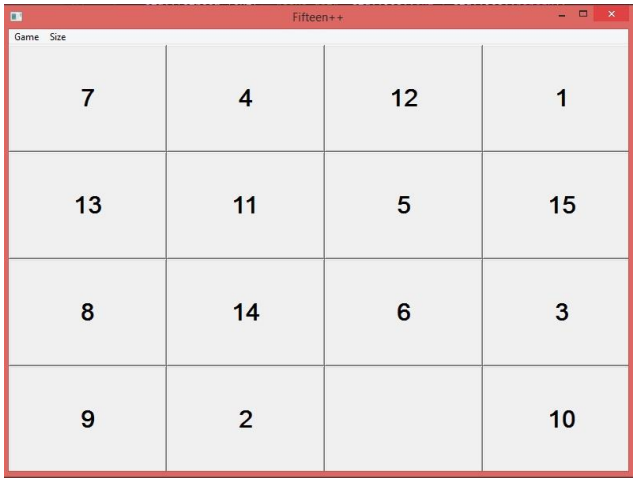

Рисунок 4.1 – Начальное состояние хода

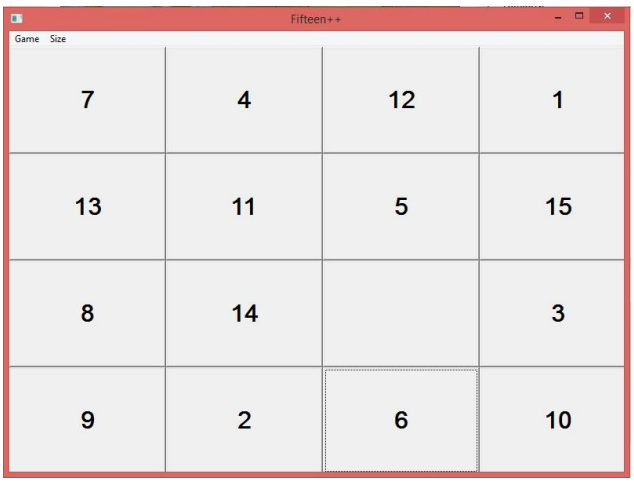

Рисунок 4.2 – Конечное состояние хода

Для запуска режима игры с загрузкой изображения необходимо выбрать пункт меню Game -> New with image, как показано на рисунке 4.3. После этого в появившемся диалоговом окне нужно открыть необходимый файл. При успешном открытии в окне появится поле 4×4 с клетками, содержащими фрагменты изображения, что видно на рисунке 4.4.

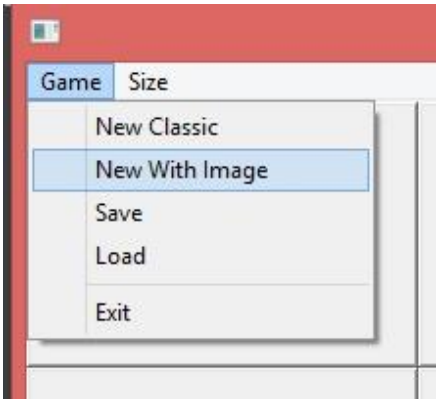

Рисунок 4.3 – Пункт меню для открытия изображения

Рисунок 4.4 – Результат открытия изображения

При необходимости сохранения игрового прогресса нужно выбрать пункт меню Game->Save, что отражено на рисунке 4.5 При успешном сохранении появится сообщение, с текстом об этом.

Рисунок 4.5 – Пункт меню для сохранения прогресса

Впоследствии, через пункт меню Game -> Load можно восстановить сохраненный прогресс, процесс показан на рисунках 4.6 и 4.7.

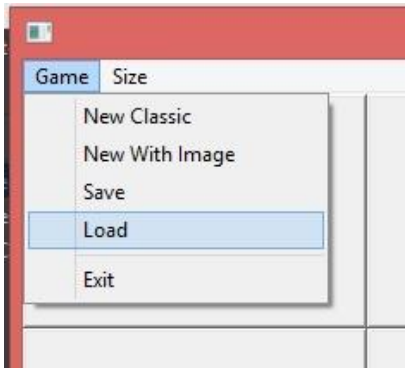

Рисунок 4.6 – Пункт меню для загрузки прогресса

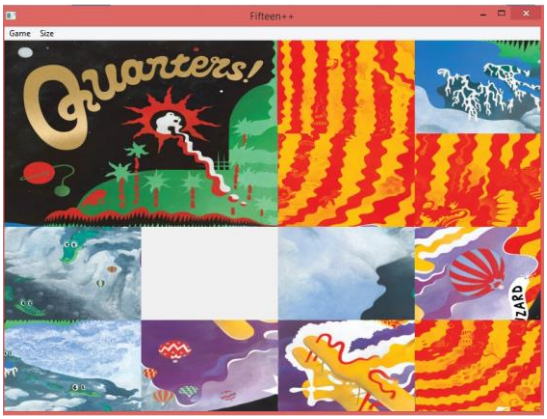

Рисунок 4.7 – Результат загрузки прогресса

Для изменения размеров игрового поля нужно воспользоваться пунктом меню Size и выбрать там подходящий размер из представленных, как показано на рисунке 4.8. В результате игровое поле перестроится, сформируется в соответствии с указанными размерами, к примеру, как на рисунке 4.9.

Рисунок 4.8 – Пункт меню для изменения размера поля

Рисунок 4.9 – Результат изменения размера поля

ЗАКЛЮЧЕНИЕ

С каждым годом популярность компьютерных игр растет. Результатом технологического прогресса и также массового распространения персональных компьютеров и мобильных устройств, стал все более расширяющийся рынок развлечений, что является веским аргументом в пользу проектов по созданию новых игр. Актуальность разработки игр становится все более неоспоримой, рынок игр увеличивается в геометрической прогрессии. За почти 50 лет индустрия компьютерных игр развилась до таких масштабов, что уже во многом опережает своих ближайших конкурентов: киноиндустрию, музыкальную индустрию, шоу-бизнес.

В процессе выполнения курсового проекта мной было разработано полностью рабочее игровое приложение, позволяющее с легкостью провести игру в Пятнашки с компьютера. Программа содержит два режима игры и поддерживает все функции и логику оригинальной головоломки, даёт возможность любому с легкостью освоить её. Приложение отличается стабильностью, плавностью, хорошим временем отклика, интуитивно понятным интерфейсом и поддержкой изображений в высоком разрешении.

Я получил большой опыт работы с языком программирования C++, набором функций WinAPI и библиотекой для работы с графикой GDI+.

Данный проект будет активно дорабатываться мной, вот некоторые из функций, которые я бы хотел добавить:

– Таймер показывающий время сборки головоломки;
– Таблицу рекордов по времени сборки;
– Звуковое сопровождение;
– Функцию автоматического решения с использованием алгоритма A*;
– Подсказки ходов на основе сгенерированной сборки;
– Поддержку нескольких сохранений.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] www.wikipedia.org [Электронный портал]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/15_puzzle

[2] lorecioni.github.io [Электронный портал]. – Электронные данные. – Режим доступа: https://lorecioni.github.io/fifteen-puzzle-game

[3] docs.microsoft.com [Электронный портал]. – Электронные данные. – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/

[4] www.wikipedia.org [Электронный портал]. – Электронные данные. – Режим доступа: https://en.wikipedia.org/wiki/Multiple_buffering

[5] Адитья Бхаргава. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих [Текст] / Адитья Бхаргава. – Санкт – Петербург: Издательство Питер, 2019. – 288с.

[6] mathworld.wolfram.com [Электронный портал]. – Электронные данные. – Режим доступа :https://mathworld.wolfram.com/15Puzzle.html

ПРИЛОЖЕНИЕ 1
(обязательное)
Исходный код программы

Файл main.cpp

```
#ifndef _UNICODE
#define _UNICODE
#endif
#ifndef UNICODE
#define UNICODE
#endif

#include "Game.h"

const COLORREF BACKGROUND_COLOR = RGB(255, 255, 255);
const SIZE FIRST_WINDOW_SIZE = SIZE{800, 600};
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

using namespace Gdiplus;

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR pCmdLine, int nCmdShow){
   const wchar_t WINDOW_CLASS[] = L"MAIN_WINDOW_CLASS";
   const wchar_t WINDOW_TITLE[] = L"Fifteen++";

   GdiplusStartupInput gdiplusStartupInput;
   ULONG_PTR gdiplusToken;
   GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, nullptr);

   WNDCLASSEXW wc;
   wc.cbSize = sizeof(wc);
   wc.style = 0;
   wc.lpfnWndProc = WindowProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIconW(nullptr, IDI_APPLICATION);
   wc.hCursor = LoadCursorW(nullptr, IDC_ARROW);
   wc.hbrBackground = CreateSolidBrush(BACKGROUND_COLOR);
   wc.lpszMenuName = nullptr;
   wc.lpszClassName = WINDOW_CLASS;
   wc.hIconSm = LoadIconW(nullptr, IDI_APPLICATION);

   RegisterClassExW(&wc);

   HWND hwnd = CreateWindowExW(
         0x0,
         WINDOW_CLASS,
         WINDOW_TITLE,
         WS_OVERLAPPEDWINDOW,
         CW_USEDEFAULT, CW_USEDEFAULT, FIRST_WINDOW_SIZE.cx, FIRST_WINDOW_SIZE.cy,
         nullptr,
         nullptr,
         hInstance,
         nullptr
   );

   if (hwnd == nullptr) {
      MessageBoxW(nullptr, L"Error creating window", L"Attention", MB_OK);
      return 0;
```

```
    }

    ShowWindow(hwnd, nCmdShow);

    MSG msg = {};
    while (GetMessageW(&msg, nullptr, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessageW(&msg);
    }

    GdiplusShutdown(gdiplusToken);

    return 0;
}
```

Файл Game.h

```
#ifndef FIFTEEN_GAME_H
#define FIFTEEN_GAME_H

#include <windows.h>
#include <ctime>
#include <string>
#include <iostream>
#include <vector>
#include <gdiplus.h>
#include <cmath>
#include <fstream>
#include "Shlwapi.h"

#define GAME_MENU_NEW_CLASSIC 1
#define GAME_MENU_NEW_IMAGE 2
#define GAME_MENU_SAVE 3
#define GAME_MENU_LOAD 4
#define GAME_MENU_EXIT 5
#define GAME_SIZE_THREE 33
#define GAME_SIZE_FOUR 44
#define GAME_SIZE_FIVE 55
#define GAME_SIZE_SIX 66
#define GAME_SIZE_SEVEN 77

#define MENU_HEIGHT 55;

struct Cell{
    int id;
    HWND btn;
    boolean isEmpty;
    Gdiplus::Bitmap *bitmap;
};

struct Scale{
    int x;
    int y;
};

Scale GetWindowSteps(HWND);
boolean CheckSolution();
void RandomizeButtons();
void DrawGameField(HWND, Scale steps);
void Resize(HWND);
void CreateImageGameField(HWND, Scale, int, int);
```

```
void CreateClassicGameField(HWND, Scale, int, int);
void ClearGameField();
boolean CheckWinSituation();
void CheckMove(HWND, HWND);
void ScaleImage(HWND, Gdiplus::Bitmap*);
void OnNewClassicGame(HWND);
void OnNewImageGame(HWND);
void OnSave(HWND);
void OnLoad(HWND);
void RestoreImageGame(HWND, Scale);
void RestoreClassicGame(HWND, Scale);
void ResizeGameField(HWND, int);
void CutImage(LPARAM);
void DrawBitmap(HDC, RECT, Gdiplus::Image);
std::wstring OpenFile(HWND);
void AddMenus(HWND);
void Initialize(HWND);
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

#endif
```

Файл Game.cpp

```
#include "Game.h"

using namespace std;

vector<vector<Cell>> cells;
int n;
HMENU hMenu, hSizeMenu;
Gdiplus::Bitmap *Image = nullptr;
int idTimer = -1;
boolean needToRedraw = true;

const SIZE MIN_WINDOW_SIZE = SIZE{500, 500};

LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
   switch (uMsg) {
      case WM_PAINT: {
         PAINTSTRUCT ps;
         HDC hdc = BeginPaint(hWnd, &ps);
         EndPaint(hWnd, &ps);
         break;
      }
      case WM_COMMAND: {
         switch (wParam) {
            case GAME_MENU_NEW_CLASSIC: {
               OnNewClassicGame(hWnd);
               break;
            }
            case GAME_MENU_NEW_IMAGE: {
               OnNewImageGame(hWnd);
               break;
            }
            case GAME_MENU_SAVE: {
               OnSave(hWnd);
               break;
            }
            case GAME_MENU_LOAD: {
               OnLoad(hWnd);
               break;
```

```
            }
            case GAME_SIZE_THREE:{
               ResizeGameField(hWnd, 3);
               break;
            }
            case GAME_SIZE_FOUR:{
               ResizeGameField(hWnd, 4);
               break;
            }
            case GAME_SIZE_FIVE:{
               ResizeGameField(hWnd, 5);
               break;
            }
            case GAME_SIZE_SIX:{
               ResizeGameField(hWnd, 6);
               break;
            }
            case GAME_SIZE_SEVEN:{
               ResizeGameField(hWnd, 7);
               break;
            }
         }
         HWND buttonClicked = (HWND)lParam;
         CheckMove(hWnd, buttonClicked);
         break;
      }
      case WM_DRAWITEM:
      {
         CutImage(lParam);
         break;
      }
      case WM_TIMER: {
         if (needToRedraw){
            Resize(hWnd);
         }
      }
      case WM_ERASEBKGND:
         return (LRESULT)1;
      case WM_CREATE: {
         Initialize(hWnd);
         break;
      }
      case WM_DESTROY: {
         PostQuitMessage(0);
         break;
      }
      case WM_SIZE: {
         needToRedraw = true;
         break;
      }
      case WM_GETMINMAXINFO: {
         LPMINMAXINFO lpMMI = (LPMINMAXINFO) lParam;
         lpMMI->ptMinTrackSize.x = MIN_WINDOW_SIZE.cx;
         lpMMI->ptMinTrackSize.y = MIN_WINDOW_SIZE.cy;
         break;
      }
      default: {
         return DefWindowProc(hWnd, uMsg, wParam, lParam);
      }
   }
   return 0;
}
```

```
Scale GetWindowSteps(HWND hWnd){
    Scale result = {0};
    RECT rect;
    GetWindowRect(hWnd, &rect);

    int width = rect.right - rect.left;
    int height = rect.bottom - rect.top - 55;

    result.x = width / n;
    result.y = height / n;

    return result;
}

int getInversionsCount(){
    int count = 0;
    for (int i = 0; i < n * n - 1; i++){
        for (int j = i + 1; j < n * n; j++){
            if (cells[j/n][j%n].id && cells[i/n][i%n].id && cells[i/n][i%n].id > cells[j/n][j%n].id)
                count++;
        }
    }
    return count;
}

int findEmptyPosition(){
   for (int i = n - 1; i >= 0; i--)
        for (int j = n - 1; j >= 0; j--)
            if (cells[i][j].id == 0)
                return n - i;
    return 0;
}

boolean CheckSolution(){
    int invCount = getInversionsCount();

    if (n & 1)
        return !(invCount & 1);
    else {
        int pos = findEmptyPosition();
        if (pos & 1)
            return !(invCount & 1);
        else
            return invCount & 1;
    }
}

void RandomizeButtons(){
    int col1, col2, row1, row2;
    do {
        for (int i = 0; i < 100; i++) {
            col1 = rand() % n;
            col2 = rand() % n;
            row1 = rand() % n;
            row2 = rand() % n;
            std::swap(cells[row1][col1], cells[row2][col2]);
        }
    }while (!CheckSolution());
}

void DrawGameField(HWND hWnd, Scale steps){
```

```
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            MoveWindow(cells[i][j].btn, j * steps.x, i * steps.y, steps.x, steps.y, 0);
        }
    }
    InvalidateRect(hWnd, nullptr, false);
}

void Resize(HWND hWnd){
    if (!cells.empty()){
        Scale steps = GetWindowSteps(hWnd);
        DrawGameField(hWnd, steps);
        InvalidateRect(hWnd, nullptr, true);
        needToRedraw = false;
    }
}

void CreateImageGameField(HWND hWnd, Scale steps, int emptyI, int emptyJ){
    int id = 1;
    cells.resize(n);
    for(int i = 0; i < n; i++){
        cells[i].resize(n);
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cells[i][j].btn = CreateWindowW(L"Button", L"", WS_VISIBLE | WS_CHILD | BS_OWNERDRAW, steps.x
* j, steps.y * i, steps.x, steps.y, hWnd, nullptr, nullptr, nullptr);
            if( (i != emptyI) || (j != emptyJ)){
                cells[i][j].id = id++;
                cells[i][j].isEmpty = false;
                cells[i][j].bitmap = Image->Clone(steps.x * j, steps.y * i, steps.x, steps.y, PixelFormat24bppRGB);
            } else {
                cells[i][j].isEmpty = true;
                cells[i][j].id = 0;
            }
        }
    }
    RandomizeButtons();
}

void CreateClassicGameField(HWND hWnd, Scale steps, int emptyI, int emptyJ){
    int id = 1;
    cells.resize(n);
    for(int i = 0; i < n; i++){
        cells[i].resize(n);
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cells[i][j].btn = CreateWindowW(L"Button", nullptr, WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
steps.x * j, steps.y * i, steps.x, steps.y, hWnd, nullptr, nullptr, nullptr);
            if( (i != emptyI) || (j != emptyJ)){
                HFONT font = CreateFontW(35, 0, 0, 0, FW_SEMIBOLD, false, false, false, DEFAULT_CHARSET,
OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH, NULL);
                cells[i][j].id = id++;
                cells[i][j].isEmpty = false;
                wchar_t text[3] = L"";
                swprintf_s(text, L"%d", cells[i][j].id);
                SetWindowText(cells[i][j].btn, text);
                SendMessageW(cells[i][j].btn, WM_SETFONT, (WPARAM)font, (LPARAM)false);
            } else {
                cells[i][j].id = 0;
                cells[i][j].isEmpty = true;
```

```
            wchar_t text[3] = L"";
            SetWindowText(cells[i][j].btn, text);
        }
    }
  }
  RandomizeButtons();
}

void ClearGameField(){
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      DestroyWindow(cells[i][j].btn);
    }
  }
}

boolean CheckWinSituation(){
  for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
      if ((cells[i][j].id != i * n + j + 1) && (cells[i][j].id != 0)){
        return false;
      }
    }
  }
  return true;
}

void CheckMove(HWND hWnd, HWND buttonClicked){
  boolean swapped = false;
  int i = 0;
  int j = 0;
  while (i < n && !swapped) {
    j = 0;
    while (j < n && !swapped) {
      if (buttonClicked == cells[i][j].btn){
        int emptyI;
        int emptyJ;

        for(int k = 0; k < n; k++){
          for(int q = 0; q < n; q++){
            if (cells[k][q].isEmpty){
              emptyI = k;
              emptyJ = q;
            }
          }
        }

        Scale steps = GetWindowSteps(hWnd);
        float delta = sqrt((i - emptyI) * (i - emptyI) + (j - emptyJ) * (j - emptyJ));
        if (delta <= 1.01){
          swapped = true;
          std::swap(cells[i][j], cells[emptyI][emptyJ]);
          MoveWindow(cells[i][j].btn, j * steps.x, i * steps.y, steps.x, steps.y, 0);
          MoveWindow(cells[emptyI][emptyJ].btn, emptyJ * steps.x, emptyI * steps.y, steps.x, steps.y, 1);

          if (CheckWinSituation()){
            if (MessageBoxW(hWnd, L"Congratulations! You won the video game", L"Start new game?",
MB_YESNO | MB_ICONQUESTION) == IDYES){
              ClearGameField();
              Image == nullptr ? CreateClassicGameField(hWnd, steps, n - 1, n - 1) :
CreateImageGameField(hWnd, steps, n - 1, n - 1);
              DrawGameField(hWnd, steps);
```

```
                }
            }
        }
    }
    j++;
  }
  i++;
 }
}

void ScaleImage(HWND hWnd, Gdiplus::Bitmap *bitmap){
  RECT rect;
  GetWindowRect(hWnd, &rect);

  int width = rect.right - rect.left;
  int height = rect.bottom - rect.top - MENU_HEIGHT;

  float horizontalScalingFactor = (float) width / (float) bitmap->GetWidth();
  float verticalScalingFactor = (float) height / (float) bitmap->GetHeight();

  Image = new Gdiplus::Bitmap(width, height);
  Image->SetResolution(bitmap->GetHorizontalResolution(), bitmap->GetVerticalResolution());
  Gdiplus::Graphics g(Image);
  g.ScaleTransform(horizontalScalingFactor, verticalScalingFactor);
  g.DrawImage(bitmap, 0, 0);
}

void OnNewClassicGame(HWND hWnd){
  if (MessageBoxW(hWnd, L"Your progress will be lost?", L"Start new game", MB_YESNO |
MB_ICONQUESTION) == IDNO)
    return;
  ClearGameField();
  Image = nullptr;
  n = 4;
  Scale steps = GetWindowSteps(hWnd);

  CreateClassicGameField(hWnd, steps, n - 1, n - 1);
  DrawGameField(hWnd, steps);
}

void OnNewImageGame(HWND hWnd){
  if (MessageBoxW(hWnd, L"Your progress will be lost?", L"Start new game", MB_YESNO |
MB_ICONQUESTION) == IDNO)
    return;
  wchar_t buffer[256];
  GetCurrentDirectoryW(256, buffer);
  std::wstring name = OpenFile(hWnd);
  SetCurrentDirectoryW(buffer);
  Gdiplus::Bitmap bitmap(name.c_str());

  if(bitmap.GetLastStatus() != Gdiplus::Ok){
    MessageBoxW(nullptr, L"Can't create bitmap", L"Attention", MB_OK);
    return;
  }

  ScaleImage(hWnd, &bitmap);

  ClearGameField();
  n = 4;
  Scale steps = GetWindowSteps(hWnd);

  cells.resize(n);
```

```
    for(int i = 0; i < n; i++){
        cells[i].resize(n);
    }
    CreateImageGameField(hWnd, steps, n - 1, n - 1);
    DrawGameField(hWnd, steps);
}


void OnSave(HWND hWnd){
    if(Image != nullptr){
        CLSID pngClsid;
        CLSIDFromString(L"{557CF406-1A04-11D3-9A73-0000F81EF32E}", &pngClsid);
        Image->Save(L"save.png", &pngClsid, nullptr);
    } else {
        wchar_t buffer[] = L"save.png";
        wchar_t *lpStr;
        lpStr = buffer;

        if(PathFileExistsW(lpStr))
            DeleteFileW(lpStr);
    }

    std::fstream fout("save.dat", std::ios::out | std::ios::binary);
    if (!fout)
    {
        MessageBoxW(hWnd, L"Can't open file for writing.", L"Error", MB_OK | MB_ICONERROR);
        return;
    }
    if (MessageBoxW(hWnd, L"Last save will be deleted", L"Are you sure?", MB_YESNO | MB_ICONQUESTION)
== IDNO)
        return;

    fout << n << ' ';

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (cells[i][j].isEmpty)
                fout << i << ' ' << j << ' ';


    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            fout << cells[i][j].id << ' ';

    fout.close();
    MessageBoxW(hWnd, L"Game saved.", L"Success!", MB_OK | MB_ICONASTERISK);
}

void RestoreImageGame(HWND hWnd, Scale steps){
    int x0, y0;
    for(int i = 0; i<n; i++){
        for(int j = 0; j<n; j++){
            y0 = (cells[i][j].id - 1) / n;
            x0 = (cells[i][j].id - 1) % n;
            cells[i][j].btn = CreateWindowW(L"Button", L"", WS_VISIBLE | WS_CHILD | BS_OWNERDRAW, steps.x
* x0, steps.y * y0, steps.x, steps.y, hWnd, nullptr, nullptr, nullptr);
            if( !cells[i][j].isEmpty ) {
                cells[i][j].bitmap = Image->Clone(steps.x * x0, steps.y * y0, steps.x, steps.y, PixelFormat32bppRGB);
            }
        }
    }
}
```

```
void RestoreClassicGame(HWND hWnd, Scale steps){
   int x0, y0;
   for(int i = 0; i<n; i++){
      for(int j = 0; j<n; j++){
         y0 = (cells[i][j].id - 1) / n;
         x0 = (cells[i][j].id - 1) % n;
         cells[i][j].btn = CreateWindowW(L"Button", L"", WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON, steps.x
* x0, steps.y * y0, steps.x, steps.y, hWnd, nullptr, nullptr, nullptr);
         if( !cells[i][j].isEmpty ){
            wchar_t text[3] = L"";
            swprintf_s(text, L"%d", cells[i][j].id);
            SetWindowText(cells[i][j].btn, text);
         } else {
            wchar_t text[3] = L"";
            SetWindowText(cells[i][j].btn, text);
         }
      }
   }
}

void OnLoad(HWND hWnd){
   boolean isImageOpen = false;
   wchar_t buffer[] = L"save.png";
   wchar_t *lpStr;
   lpStr = buffer;

   std::fstream fin("save.dat", std::ios::in | std::ios::binary);
   if (MessageBoxW(hWnd, L"Load another game?", L"Are you sure?", MB_YESNO | MB_ICONQUESTION) ==
IDNO)
      return;
   if (!fin){
      MessageBoxW(hWnd, L"No loadings available.", L"Error", MB_OK | MB_ICONERROR);
      return;
   }

   if(PathFileExistsW(lpStr)){
      Gdiplus::Bitmap bitmap(L"save.png");
      isImageOpen = true;
      ScaleImage(hWnd, &bitmap);
   }

   ClearGameField();

   cells.clear();

   fin >> n;
   cells.resize(n);
   for(int i = 0; i < n; i++){
      cells[i].resize(n);
   }

   Scale steps  = GetWindowSteps(hWnd);

   int emptyI, emptyJ;
   fin >> emptyI;
   fin >> emptyJ;

   for (int i = 0; i < n; i++){
      for (int j = 0; j < n; j++){
         fin >> cells[i][j].id;
         cells[i][j].isEmpty = false;
```

```
        }
    }

    cells[emptyI][emptyJ].isEmpty= true;

    fin.close();

    isImageOpen ? RestoreImageGame(hWnd, steps) : RestoreClassicGame(hWnd, steps);
    DrawGameField(hWnd, steps);
}

void ResizeGameField(HWND hWnd, int size){
    if (n != size) {
        ClearGameField();
        n = size;
        Scale steps = GetWindowSteps(hWnd);
        if(Image != nullptr){
            Gdiplus::Bitmap *bitmap = Image->Clone();
            ScaleImage(hWnd, bitmap);
            CreateImageGameField(hWnd, steps, n - 1, n - 1);
        } else {
            CreateClassicGameField(hWnd, steps, n - 1, n - 1);
        }
        DrawGameField(hWnd, steps);
    }
}

void DrawBitmap(HDC hDC, RECT rc, Gdiplus::Image *bitmap){
    HDC memDC = CreateCompatibleDC(hDC);

    HBITMAP bmp = CreateCompatibleBitmap(hDC, rc.right - rc.left,rc.bottom - rc.top);
    auto oldBmp = (HBITMAP) SelectObject(memDC, bmp);

    Gdiplus::Graphics(memDC).DrawImage(bitmap, Gdiplus::Rect(rc.left, rc.top, rc.right - rc.left, rc.bottom - rc.top));

    BitBlt(hDC, 0, 0, rc.right - rc.left, rc.bottom - rc.top, memDC, 0, 0, SRCCOPY);

    SelectObject(memDC, oldBmp);
    DeleteObject(bmp);
    DeleteDC(memDC);
}

void CutImage(LPARAM lParam){
    auto lpdrawstLogon = (LPDRAWITEMSTRUCT) lParam;
    PAINTSTRUCT ps;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            if ((lpdrawstLogon->hwndItem == cells[i][j].btn) && !cells[i][j].isEmpty)
            {
                DrawBitmap(lpdrawstLogon->hDC, lpdrawstLogon->rcItem, cells[i][j].bitmap);
            }
        }
    }
}

std::wstring OpenFile(HWND hWnd){
    OPENFILENAME ofn;

    wchar_t path[256];

    ZeroMemory(&ofn, sizeof(OPENFILENAME));
```

```
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = hWnd;
    ofn.lpstrFile = path;
    ofn.lpstrFile[0] = '\0';
    ofn.nMaxFile = 256;
    ofn.lpstrFilter = L"Images\0*.jpg;*.jpeg;*.png;*.bmp\0";
    ofn.nFilterIndex = 1;

    if (GetOpenFileName(&ofn) == FALSE){
        MessageBoxW(nullptr, L"Error opening image", L"Attention", MB_OK);
    }

    return ofn.lpstrFile;
}

void AddMenus(HWND hWnd){
    hMenu = CreateMenu();
    HMENU hFileMenu = CreateMenu();
    AppendMenuA(hMenu, MF_POPUP, (UINT_PTR)hFileMenu, "Game");
    AppendMenuA(hFileMenu, MF_STRING, GAME_MENU_NEW_CLASSIC, "New Classic");
    AppendMenuA(hFileMenu, MF_STRING, GAME_MENU_NEW_IMAGE, "New With Image");
    AppendMenuA(hFileMenu, MF_STRING, GAME_MENU_SAVE, "Save");
    AppendMenuA(hFileMenu, MF_STRING, GAME_MENU_LOAD, "Load");
    AppendMenuA(hFileMenu, MF_SEPARATOR, (UINT_PTR)nullptr, nullptr);
    AppendMenuA(hFileMenu, MF_STRING, GAME_MENU_EXIT, "Exit");
    hSizeMenu = CreateMenu();
    AppendMenuA(hMenu, MF_POPUP, (UINT_PTR)hSizeMenu, "Size");
    AppendMenuA(hSizeMenu, MF_STRING, GAME_SIZE_THREE, "3 * 3");
    AppendMenuA(hSizeMenu, MF_STRING, GAME_SIZE_FOUR, "4 * 4");
    AppendMenuA(hSizeMenu, MF_STRING, GAME_SIZE_FIVE, "5 * 5");
    AppendMenuA(hSizeMenu, MF_STRING, GAME_SIZE_SIX, "6 * 6");
    AppendMenuA(hSizeMenu, MF_STRING, GAME_SIZE_SEVEN, "7 * 7");
    SetMenu(hWnd, hMenu);
}

void Initialize(HWND hWnd){
    srand(time(nullptr));
    SetTimer(hWnd, idTimer = 1, 1, nullptr);
    AddMenus(hWnd);

    n = 4;
    Scale steps = GetWindowSteps(hWnd);
    CreateClassicGameField(hWnd, steps, n - 1, n - 1);
    DrawGameField(hWnd, steps);
}
```