ЗАДАНИЕ

Цель проекта: написать игру "Шашки" для двух человек.

Терминология.

Игрок – пользователь компьютерной программы.

Игровое поле – клетчатый квадрат 8*8 клетки равного размера.

Главная форма – форма приложения, содержащая игровое поле и все необходимые элементы для игры.

Модель игры.

1. Игра ведется двумя лицами на шашечной доске, разбитой на 64 квадрата, окрашенных в белый и черный цвета, 12-ью шашками белыми, принадлежащими одному игроку, и 12-ью шашками черными, принадлежащими другому игроку.

2. Доска между играющими кладется так, чтобы большая дорога шла от играющего слева направо.

3. Шашки с каждой стороны расставляются по черным квадратам на первых трех рядах от играющего.

4. Ходы играющими делаются поочередно.

5. За ход считается передвижение шашки вперед на соседний черный квадрат, а также взятие неприятельских шашек.

6. Если соседний квадрат занят вражеской, допустим, черной шашкой, а следующий за ней черный квадрат свободен, то черная шашка «бьется», т. е. белая шашка перескакивает через черную на следом за ней находящийся свободный черный квадрат, и черную шашку «съедают» — снимают с доски.

7. За один прием «бьется» столько шашек, сколько их стоит на пути на указанных выше условиях.

8. В случае, если возможно взять шашки противника одновременно по двум направлением, выбор, вне зависимости от количества, предоставляется усмотрению берущего.

9. При взятии шашки снимаются с доски только по окончании хода.

10. Два раза в один прием при ходе брать шашку (перекрещивать ее бьющей) не допускается.

11. Если шашка одного из играющих во время игры проникнет до последнего ряда, то она превращается в дамку.

Функционал программы.

Программа реализует игру в шашки между двумя пользователями в соответствии с описанной моделью.

По окончании партии программа выводит сообщение о конце игры и победителе (черные или белые), затем происходит автоматический переход в главное меню.

Также пользователи могут в главном меню прочитать правила игры.

Спроектировать, реализовать и описать в отчете программу объектно-ориентированного имитационного моделирования предметной области в соответствии с заданием.

Структура данных.

Координаты шашек (X, Y).

Матрица 8*8 – игровое поле.

Очередь хода.

Аннотация

Данная работа посвящена разработке игры «Шашки» для двух человек.

Работа выполнена на 35 страницах с использованием 4 источников, содержит 11 рисунков.

Содержание

Введение

Концепция объектно-ориентированного программирования подразумевает, что основой управления процессом реализации программы является передача сообщений объектам. Поэтому объекты должны определяться совместно с сообщениями, на которые они должны реагировать при выполнении программы. В этом состоит главное отличие ООП от процедурного программирования, где отдельно определённые структуры данных передаются в процедуры (функции) в качестве параметров. Таким образом, объектно-ориентированная программа состоит из объектов - отдельных фрагментов кода, обрабатывающего данные, которые взаимодействуют друг с другом через определённые интерфейсы.

Цель работы: закрепить и расширить знания и навыки по объектно-ориентированному проектированию и программированию, составлению технических заданий, тестированию и документированию ПО путем создания приложения, представляющего собой программную реализацию известной логической игры «Шашки».

Техническое задание

1. Терминология.

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе и скрыть детали реализации от пользователя.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс - потомком, наследником или производным классом.

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Класс является описываемой на языке терминологии (пространства имён) исходного кода моделью ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это экземпляр класса. При этом в некоторых исполняющих системах класс также может представляться

некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

Прототип — это объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка.

2. Цель создания приложения.

Данная программа предназначена для пользователей, которые любят логические игры. Игра предназначена для игры двух человек.

3. Требования к приложению.

Приложение должно выполнять следующий функционал:

– очистка и генерация поля;

– смена ходов;

– удаление фишек противника (как отдельно, так и подряд, при возможности);

– не королевскими шашками делать ход назад нельзя;

– замена обычных шашек на королев при достижении королевского ряда.

4. Описание экранов.

На рисунке 3 в приложении А изображена главная форма приложения. С ее помощью можно либо начать игру, либо перейти в окно «Правила игры», либо выйти из игры.

На рисунке 4 в приложении А изображено окно «Правила игры», которое описывает правила игры.

На рисунке 5 в приложении А изображено окно с самой игрой. На нем расположено игровое поле.

Проект программного продукта

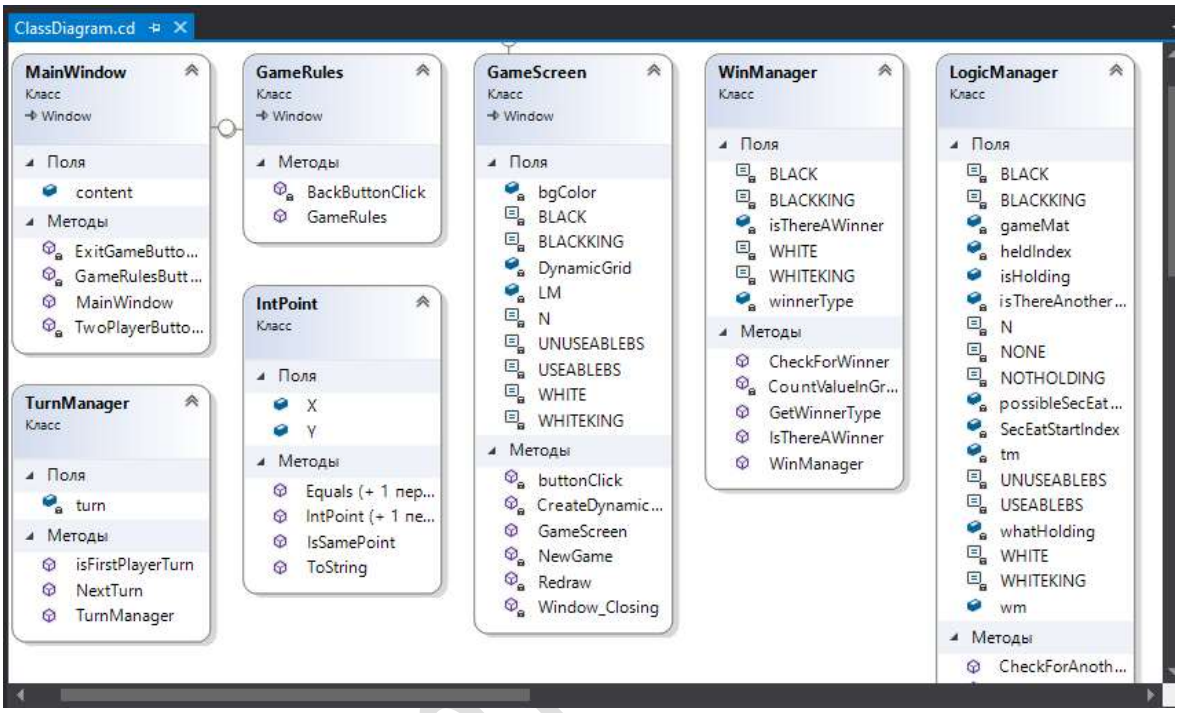

Рисунок 1 – Диаграмма классов

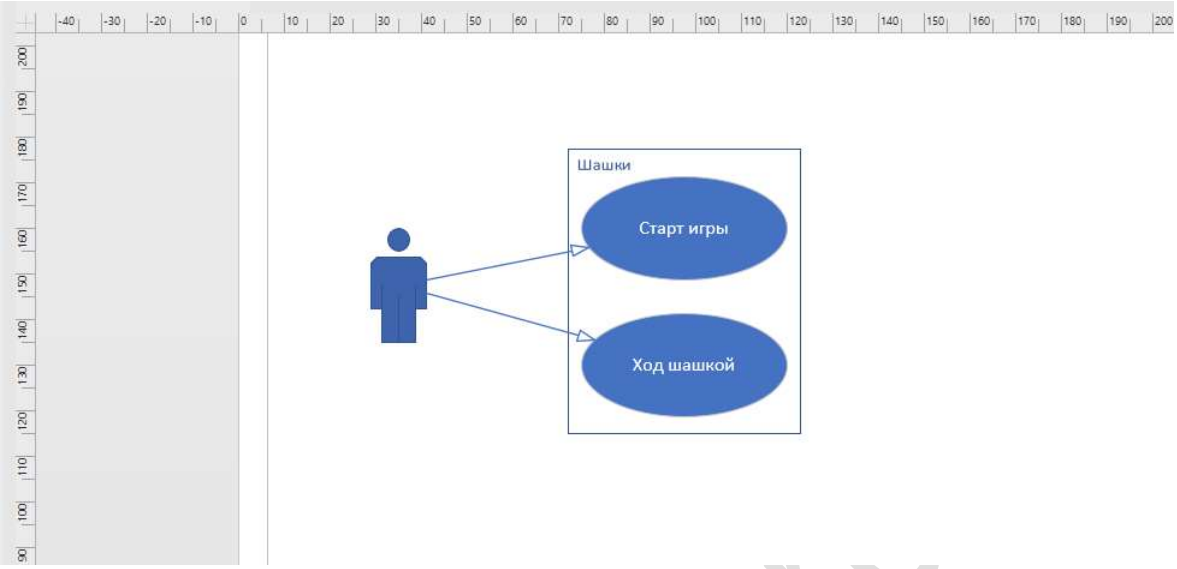

Рисунок 2 – Диаграмма вариантов использования

Класс «GameRules» реализует окно с правилами игры. На нем расположено лишь текстовое поле и кнопка для возврата в меню.

Класс «GameScreen» реализует окно с игровым полем. Здесь происходит генерация поля в методе «CreateDynamicGrid».

Класс «IntPoint» содержит поля для хранения координат шашек.

Класс «LogicManager» содержит всю логику игры, то есть ее алгоритм:

– удержание шашки при ее выборе;

– отпускание шашки при отмене ее выбора;

– ход шашки, который контролируется согласно правилам игры;

– удаление шашек (единичное или несколько за ход);

– замена шашек на королев.

Класс «MainWindow» реализует главное окно приложения, которое содержит три кнопки и текстовое поле.

Класс «TurnManager» контролирует очередность хода игроков.

Класс «WinManager» подсчитывает количество оставшихся шашек на поле и определяет кто есть победитель партии.

Описание программного продукта

При запуске игры необходимо выбрать «Начать игру». Появится окно с игровым полем и расположенными на нем игровыми шашками по 12 на каждого игрока. По правилам игры первым делает ход игрок с белыми шашками. Далее ход переходит к противнику. При возникновении случая удаления шашки противника, программа не будет вынуждать обязательно это сделать, что будет означать невнимательность игрока, если он этого не сделает. По окончании игры программа выведет сообщение о победителе и переведет пользователя на главное окно.

Заключение

В рамках данной работы было разработано программное обеспечение, реализующее собой игру «Шашки». В ходе написания проекта была освоена и закреплена работа с классами и объектами.

С помощью разработанной программы пользователи могут овладеть искусством игры в «Шашки», а также развить свое логическое мышление.

Пользовательское приложение разработано в среде программирования Visual Studio на платформе WPF.

Список использованных источников

1. Сайт о программировании Metanit [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf – 26.08.19.

2. Мэтью Мак-Дональд - Windows Presentation Foundation в .NET 4.0 с примерами на C# 2010.

3. Официальный сайт Википедия [Электронный ресурс]. – Режим доступа:https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D1%88%D0%BA%D0%B8 – 24.08.19.

4. Ишкова, Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.

Приложение А. Снимки экранных форм пользовательского интерфейса

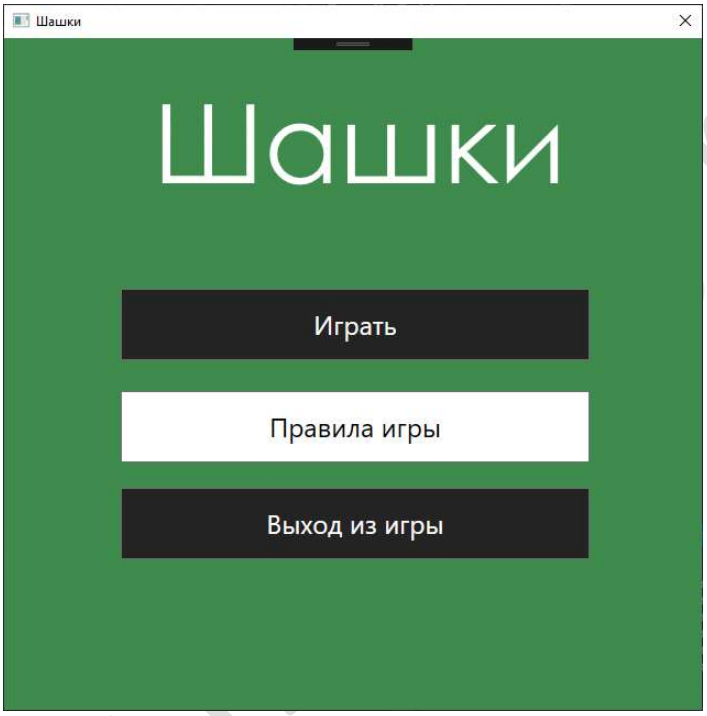

Рисунок 3 – Главное меню игры

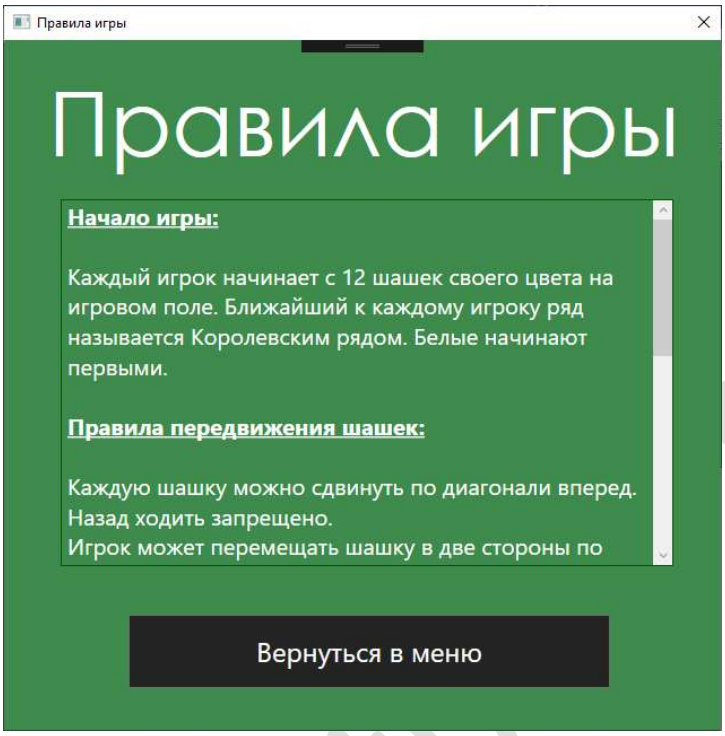

Рисунок 4 – Окно «Правила игры»

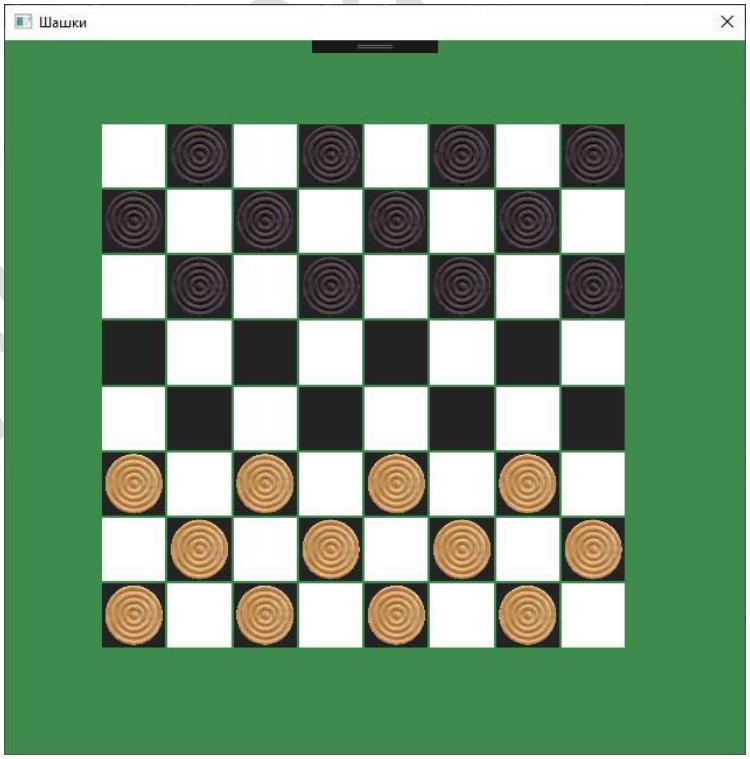

Рисунок 5 – Окно игры

Приложение Б. Тесты

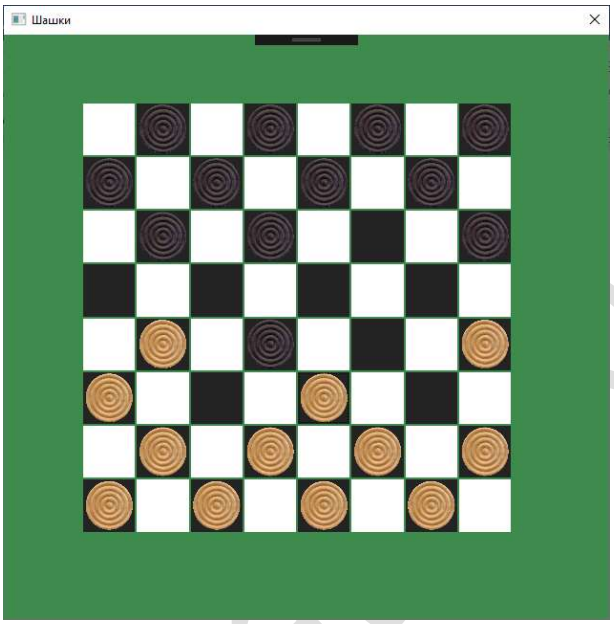

Рисунок 6 – Передвижение шашек на игровом поле

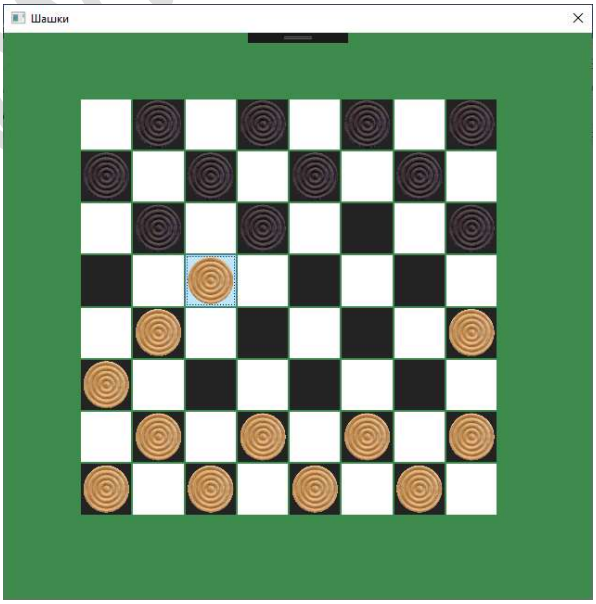

Рисунок 7 – Удаление черной шашки путем перескакивания через нее белой шашкой

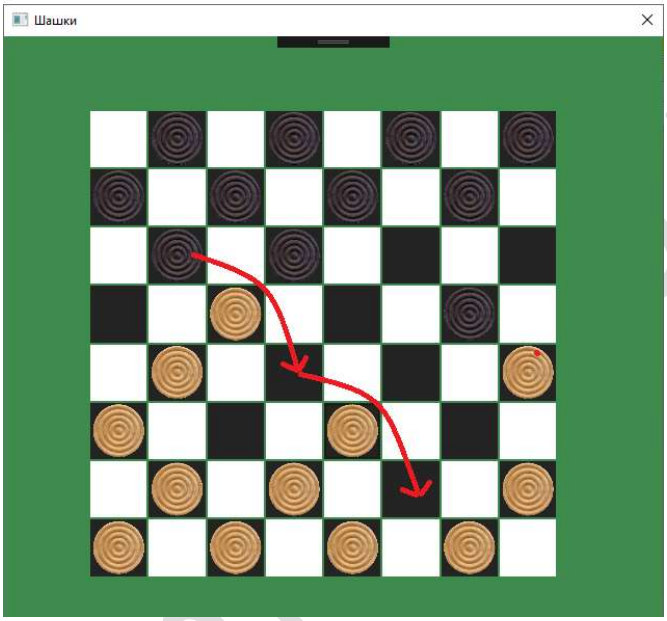

Рисунок 8 – Ситуация для удаления двух белых шашек подряд

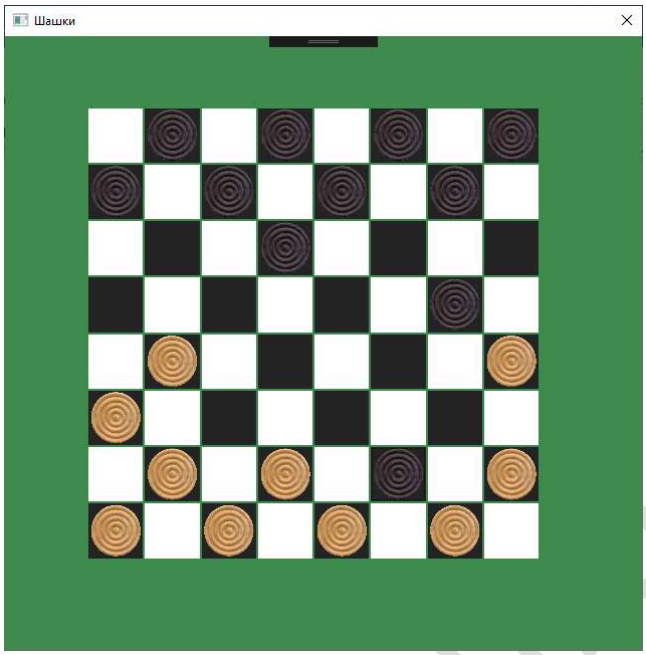

Рисунок 9 – Удаление двух белых шашек за один ход

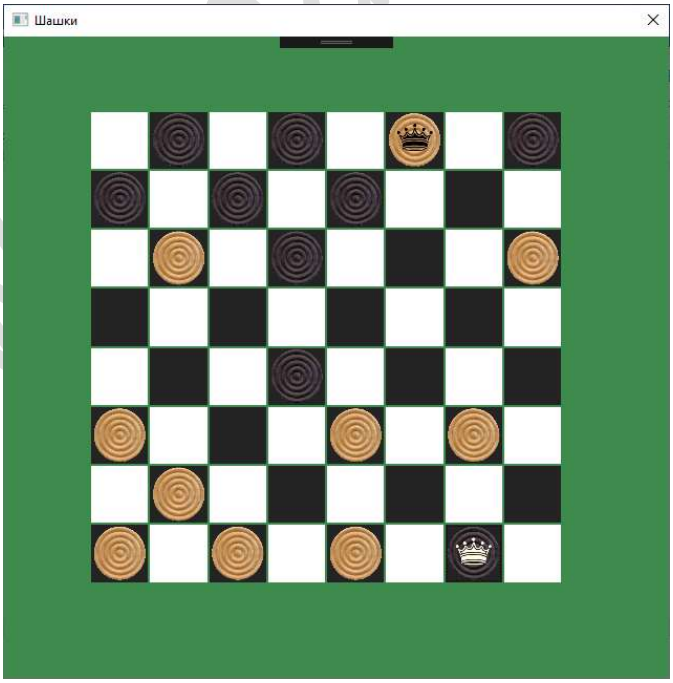

Рисунок 10 – Достижение королевского ряда

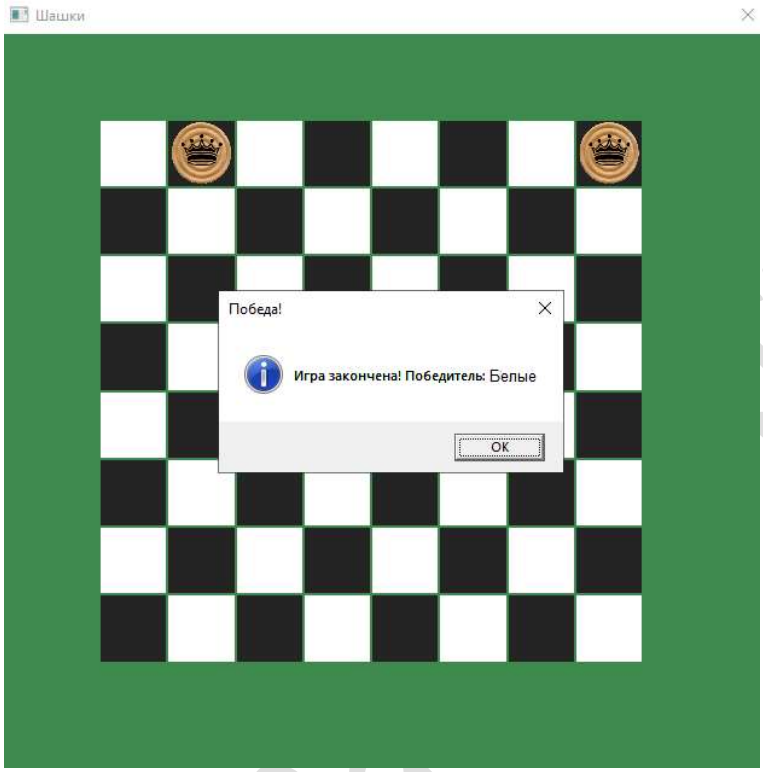

Рисунок 11 – Победа белыми шашками

Приложение В. Исходный код программы

Класс «GameRules.xaml.cs»

```
public partial class GameScreen : Window
    {

        #region Переменные
        private Grid DynamicGrid;
        private const int N = 8;
        private Brush bgColor = new SolidColorBrush(Color.FromRgb(36, 36, 36));
        private LogicManager LM;

        private const int BLACK = 1;
        private const int WHITE = 2;
        private const int BLACKKING = 3;
        private const int WHITEKING = 4;
        private const int USEABLEBS = 5;
        private const int UNUSEABLEBS = 6;
        #endregion

        public GameScreen()
        {
            InitializeComponent();
            CreateDynamicWPFGrid();
            LM = new LogicManager();
            NewGame();
        }

        private void CreateDynamicWPFGrid()
        {
            Application.Current.MainWindow.Height = 650;
            Application.Current.MainWindow.Width = 650;
            this.Title = "Шашки";
            this.Height = 650;
            this.Width = 650;
            ResizeMode = ResizeMode.NoResize;
            this.WindowStartupLocation = WindowStartupLocation.CenterScreen;
            this.Background = new SolidColorBrush(Color.FromRgb(63, 139, 77));
            DynamicGrid = new Grid();//создаем Grid где будут все создаваемые клетки и
шашки
            DynamicGrid.Width = 450;
            DynamicGrid.Height = 450;
            DynamicGrid.HorizontalAlignment = HorizontalAlignment.Left;
            DynamicGrid.VerticalAlignment = VerticalAlignment.Top;
            DynamicGrid.ShowGridLines = false;
            DynamicGrid.Background = new SolidColorBrush(Color.FromRgb(63, 139, 77));

            Button b;

            for (int i = 0; i < N; i++)
            {
                //создаем в Grid столбцы и строки
                ColumnDefinition gridCol = new ColumnDefinition();
                RowDefinition gridRow = new RowDefinition();
```

```
            gridCol.Width = new GridLength(10, GridUnitType.Star);
            gridRow.Height = new GridLength(10, GridUnitType.Star);
            DynamicGrid.ColumnDefinitions.Add(gridCol);
            DynamicGrid.RowDefinitions.Add(gridRow);
        }

        for (int i = 0; i < N * N; i++)//создаем кнопки на поле
        {
            b = new Button();
            b.BorderBrush = Brushes.Transparent;
            b.FontSize = 12;
            b.FontWeight = FontWeights.Bold;
            b.Click += new RoutedEventHandler(buttonClick);
            Grid.SetColumn(b, i % N);//уставаливаем столбец для клетки
            Grid.SetRow(b, i / N);//устанавливаем строку для клетки
            b.Tag = i;
            b.Foreground = Brushes.Red;
            DynamicGrid.Children.Add(b);//удочеряем клетку элементом Grid
        }

        Canvas cnvas = new Canvas();//сооздаем элемент Canvas для размещения на нем
созданный Grid
        cnvas.Height = DynamicGrid.Height + 20;
        cnvas.Width = DynamicGrid.Width + 20;
        cnvas.Children.Add(DynamicGrid);//удочеряем весь Grid элементом Canvas

        Content = cnvas;
    }

    private void Redraw()//перерисовка поля
    {
        Button button;
        for (int i = 0; i < N * N; i++)
        {
            button =
DynamicGrid.Children.Cast<Button>().ToList().ElementAt(i);//получаем каждую из клеток в
коллекции Grid
            button.Background = bgColor;

            switch (LM.GetValueAtIndex(i))
            {
                case BLACK://если фишка черная то ставим картинку черной фишки
                    button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\CheckerBlack.png", UriKind.RelativeOrAbsolute)), };
                    break;
                case WHITE://аналогично с белыми
                    button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\CheckerWhite.png", UriKind.RelativeOrAbsolute)), };
                    break;
                case USEABLEBS://используемая черная клетка
                    button.Content = null;
                    break;
                case UNUSEABLEBS://не используемая белая клетка
                    button.Background = Brushes.White;
                    break;
                case WHITEKING://если белая королева, ставим картинку белой королевы
                    button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\WhiteCrown.png", UriKind.RelativeOrAbsolute)), };
```

```
                        break;
                    case BLACKKING://аналогично с черной королевой
                        button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\BlackCrown.png", UriKind.RelativeOrAbsolute)), };
                        break;
                }
            }

            if (LM.wm.IsThereAWinner())//если игра закончена
            {
                //выводим сообщение
                MessageBox.Show("Игра закончена! Победитель: " +
LM.GetNameOfValue(LM.wm.GetWinnerType()), "Победа!", MessageBoxButton.OK,
MessageBoxImage.Information);
                MainWindow mw = new MainWindow();
                mw.Show();
                this.Close();//закрываем окно
            }
        }

        private void NewGame()//новая игра
        {
            LM.ResetLogicManager();
            Redraw();
        }

        private void buttonClick(object sender, RoutedEventArgs e)
        {
            Button pressedButton = sender as Button;
            int buttonIndex = int.Parse(pressedButton.Tag.ToString());
            if (LM.isHolding)
                LM.Release(buttonIndex);
            else
                LM.Hold(buttonIndex);
            Redraw();
        }

        private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            MessageBoxResult result = MessageBox.Show("Вы действительно хотите прервать
игру?", "Предупреждение", MessageBoxButton.YesNo);
            if (result != MessageBoxResult.Yes)
            {
                e.Cancel = true;
            }
        }
    }
```

Класс «GameScreen.xaml.cs»

```
public partial class GameScreen : Window
    {

        #region Переменные
```

```
        private Grid DynamicGrid;
        private const int N = 8;
        private Brush bgColor = new SolidColorBrush(Color.FromRgb(36, 36, 36));
        private LogicManager LM;

        private const int BLACK = 1;
        private const int WHITE = 2;
        private const int BLACKKING = 3;
        private const int WHITEKING = 4;
        private const int USEABLEBS = 5;
        private const int UNUSEABLEBS = 6;
        #endregion

        public GameScreen()
        {
            InitializeComponent();
            CreateDynamicWPFGrid();
            LM = new LogicManager();
            NewGame();
        }

        private void CreateDynamicWPFGrid()
        {
            Application.Current.MainWindow.Height = 650;
            Application.Current.MainWindow.Width = 650;
            this.Title = "Шашки";
            this.Height = 650;
            this.Width = 650;
            ResizeMode = ResizeMode.NoResize;
            this.WindowStartupLocation = WindowStartupLocation.CenterScreen;
            this.Background = new SolidColorBrush(Color.FromRgb(63, 139, 77));
            DynamicGrid = new Grid();//создаем Grid где будут все создаваемые клетки и
шашки
            DynamicGrid.Width = 450;
            DynamicGrid.Height = 450;
            DynamicGrid.HorizontalAlignment = HorizontalAlignment.Left;
            DynamicGrid.VerticalAlignment = VerticalAlignment.Top;
            DynamicGrid.ShowGridLines = false;
            DynamicGrid.Background = new SolidColorBrush(Color.FromRgb(63, 139, 77));

            Button b;

            for (int i = 0; i < N; i++)
            {
                //создаем в Grid столбцы и строки
                ColumnDefinition gridCol = new ColumnDefinition();
                RowDefinition gridRow = new RowDefinition();
                gridCol.Width = new GridLength(10, GridUnitType.Star);
                gridRow.Height = new GridLength(10, GridUnitType.Star);
                DynamicGrid.ColumnDefinitions.Add(gridCol);
                DynamicGrid.RowDefinitions.Add(gridRow);
            }

            for (int i = 0; i < N * N; i++)//создаем кнопки на поле
            {
                b = new Button();
                b.BorderBrush = Brushes.Transparent;
                b.FontSize = 12;
```

```
                b.FontWeight = FontWeights.Bold;
                b.Click += new RoutedEventHandler(buttonClick);
                Grid.SetColumn(b, i % N);//уставаливаем столбец для клетки
                Grid.SetRow(b, i / N);//устанавливаем строку для клетки
                b.Tag = i;
                b.Foreground = Brushes.Red;
                DynamicGrid.Children.Add(b);//удочеряем клетку элементом Grid
            }

            Canvas cnvas = new Canvas();//сооздаем элемент Canvas для размещения на нем
созданный Grid
            cnvas.Height = DynamicGrid.Height + 20;
            cnvas.Width = DynamicGrid.Width + 20;
            cnvas.Children.Add(DynamicGrid);//удочеряем весь Grid элементом Canvas

            Content = cnvas;
        }

        private void Redraw()//перерисовка поля
        {
            Button button;
            for (int i = 0; i < N * N; i++)
            {
                button =
DynamicGrid.Children.Cast<Button>().ToList().ElementAt(i);//получаем каждую из клеток в
коллекции Grid
                button.Background = bgColor;

                switch (LM.GetValueAtIndex(i))
                {
                    case BLACK://если фишка черная то ставим картинку черной фишки
                        button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\CheckerBlack.png", UriKind.RelativeOrAbsolute)), };
                        break;
                    case WHITE://аналогично с белыми
                        button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\CheckerWhite.png", UriKind.RelativeOrAbsolute)), };
                        break;
                    case USEABLEBS://используемая черная клетка
                        button.Content = null;
                        break;
                    case UNUSEABLEBS://не используемая белая клетка
                        button.Background = Brushes.White;
                        break;
                    case WHITEKING://если белая королева, ставим картинку белой королевы
                        button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\WhiteCrown.png", UriKind.RelativeOrAbsolute)), };
                        break;
                    case BLACKKING://аналогично с черной королевой
                        button.Content = new Image { Source = new BitmapImage(new
Uri("Images\\BlackCrown.png", UriKind.RelativeOrAbsolute)), };
                        break;
                }
            }

            if (LM.wm.IsThereAWinner())//если игра закончена
            {
                //выводим сообщение
```

```
                MessageBox.Show("Игра закончена! Победитель: " +
LM.GetNameOfValue(LM.wm.GetWinnerType()), "Победа!", MessageBoxButton.OK,
MessageBoxImage.Information);
                MainWindow mw = new MainWindow();
                mw.Show();
                this.Close();//закрываем окно
            }
        }

        private void NewGame()//новая игра
        {
            LM.ResetLogicManager();
            Redraw();
        }

        private void buttonClick(object sender, RoutedEventArgs e)
        {
            Button pressedButton = sender as Button;
            int buttonIndex = int.Parse(pressedButton.Tag.ToString());
            if (LM.isHolding)
                LM.Release(buttonIndex);
            else
                LM.Hold(buttonIndex);
            Redraw();
        }

        private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs
e)
        {
            MessageBoxResult result = MessageBox.Show("Вы действительно хотите прервать
игру?", "Предупреждение", MessageBoxButton.YesNo);
            if (result != MessageBoxResult.Yes)
            {
                e.Cancel = true;
            }
        }
    }
```

Класс «IntPoint»

```
class IntPoint
{
    public int X;
    public int Y;

    public IntPoint()
    {
        this.X = 0;
        this.Y = 0;
    }

    public IntPoint(int x, int y)
    {
        this.X = x;
        this.Y = y;
    }

    public override string ToString()
```

```
    {
        return "(" + this.X + ", " + this.Y +")";
    }

    public bool IsSamePoint(IntPoint pointToCheck)
    {
        return this.X == pointToCheck.X && this.Y == pointToCheck.Y;
    }

    public bool Equals(IntPoint other)
    {
        if (other == null)
            return false;

        if (this.X == other.X && this.Y == other.Y)
            return true;
        else
            return false;
    }

    public override bool Equals(Object obj)
    {
        if (obj == null)
            return false;

        IntPoint personObj = obj as IntPoint;
        if (personObj == null)
            return false;
        else
            return Equals(personObj);
    }
}
```

Класс «MainWindow.xaml.cs»

```
public partial class MainWindow : Window
    {
        public object content;

        public MainWindow()
        {
            this.ResizeMode = System.Windows.ResizeMode.NoResize;
            InitializeComponent();
            this.WindowStartupLocation =
System.Windows.WindowStartupLocation.CenterScreen;
            content = this.Content;
        }

        private void TwoPlayerButton_Click_1(object sender, RoutedEventArgs e)
        {
            GameScreen gs = new GameScreen();
            gs.ShowDialog();
        }

        private void GameRulesButton_Click_1(object sender, RoutedEventArgs e)
```

```
        {
            GameRules gr = new GameRules();
            gr.ShowDialog();
        }

        private void ExitGameButton_Click_1(object sender, RoutedEventArgs e)
        {
            var result = MessageBox.Show("Вы действительно хотите выйти из игры?",
"Подтверждение", MessageBoxButton.YesNo, MessageBoxImage.Question);
            if (MessageBoxResult.Yes == result)
                Environment.Exit(0);
        }
    }
```

Класс «TurnManager.cs»

```
class TurnManager
{
    private bool turn;

    public TurnManager()
    {
        this.turn = true;
    }

    public void NextTurn()
    {
        this.turn = !this.turn;
    }

    public bool isFirstPlayerTurn()
    {
        //если true, значит ходит первый игрок
        return turn;
    }
}
```

Класс «WinManager.cs»

```
class WinManager
{
    private bool isThereAWinner;
    private int winnerType;

    private const int BLACK = 1;
    private const int WHITE = 2;
    private const int BLACKKING = 3;
    private const int WHITEKING = 4;

    public WinManager()
    {
```

```
        this.isThereAWinner = false;
        this.winnerType = -1;
    }

    private int CountValueInGrid(int[,] gameMat, int valueToCount)
    {
        int counter = 0;
        foreach (int cell in gameMat)
            if (cell == valueToCount)
                counter++;
        return counter;
    }

    public void CheckForWinner(int[,] gameMat)//проверка на конец игры и определяем
победителя
    {
        //если количество белых фишек и количество белых королев равно 0
        if (this.CountValueInGrid(gameMat, WHITE) == 0 && this.CountValueInGrid(gameMat,
WHITEKING) == 0)
        {
            //то победитель черные
            this.isThereAWinner = true;
            this.winnerType = BLACK;
        }

        else if (this.CountValueInGrid(gameMat, BLACK) == 0 &&
this.CountValueInGrid(gameMat, BLACKKING) == 0)
        {
            //иначе победитель белые
            this.isThereAWinner = true;
            this.winnerType = WHITE;
        }
    }

    public bool IsThereAWinner()//метод для доступа из других классов
    {
        return this.isThereAWinner;
    }

    public int GetWinnerType()//метод для доступа из других классов
    {
        return this.winnerType;
    }
}
```

Класс «LogicManager.cs»

```
class LogicManager
{
    private TurnManager tm; //экземпляр класса TurnManager
    public WinManager wm; //экземпляр класса WinManager
    private int[,] gameMat; //игровая доска
    private const int N = 8; //размер доски

    //Константы
```

```
    private const int NOTHOLDING = 0;
    private const int BLACK = 1;
    private const int WHITE = 2;
    private const int BLACKKING = 3;
    private const int WHITEKING = 4;
    private const int USEABLEBS = 5;
    private const int UNUSEABLEBS = 6;
    private const int NONE = -1;

    public bool isHolding;
    private int whatHolding;
    private int heldIndex;

    private bool isThereAnotherPossibleEat;
    private List<IntPoint> possibleSecEatReleaseLocations;
    private int SecEatStartIndex;

    /// <summary>
    /// Логика создания новой игры
    /// </summary>
    public LogicManager()
    {
        this.gameMat = new int[N, N];//инициализируем массив
        this.ResetLogicManager();//сбрасываем, если что то было ранее
    }

    /// <summary>
    /// Сброс логики и всех переменных игры
    /// </summary>
    public void ResetLogicManager()
    {
        for (int i = 0; i < N; i++) //пробегаем по всему полю
            for (int j = 0; j < N; j++)
            {
                int index = (i * N) + j; //генерируем индекс клеток
                //теперь нам нужно определить черные клетки, избегая белые, так как они
не используются
                if (((index / N) % 2 == 0 && (index % N) % 2 != 0 || (index / N) % 2 != 0
&& (index % N) % 2 == 0))
                {
                    if (index < 24) //проверяем, если это позиция для черной шашки
                        this.InsertValueIntoGameMat(BLACK, i, j);
                    else if (index > 39) //иначе если позиция для белой шашки
                        this.InsertValueIntoGameMat(WHITE, i, j);
                    else //если ни то ни другое, то пустая клетка
                        this.InsertValueIntoGameMat(USEABLEBS, i, j);
                }
                else
                    this.InsertValueIntoGameMat(UNUSEABLEBS,  i,  j);  //не  белая  и  не
черная и не пустая, то это неиспользуемая клетка
            }

        //сброс переменных
        this.isHolding = false;
        this.whatHolding = NONE;
        this.heldIndex = NONE;
        this.ResetAnotherPossibleEat();
```

```
        tm = new TurnManager();
        wm = new WinManager();
    }

    /// Эта функция вставляет значение в заданное расположение точки
    private void InsertValueIntoGameMat(int value, IntPoint location)
    {
        gameMat[location.Y, location.X] = value;
    }

    /// Эта функция вставляет значение в заданное расположение i, j
    private void InsertValueIntoGameMat(int value, int i, int j)
    {
        this.gameMat[i, j] = value;
    }

    /// Эта функция возвращает расположение точки с заданным индексом
    private IntPoint GetLocationFromIndex(int index)
    {
        return new IntPoint(index % N, index / N);
    }

    /// Эта функция возвращает индекс местоположения точки на поле
    public int GetIndexFromLocation(IntPoint location)
    {
        return (location.Y * N) + location.X;
    }

    /// Эта функция будет генерировать "первый этап" для движения в соответствии с индексом и типом.
    private List<IntPoint> GetFirstFloorLocations(int checkerIndex, int holdingType)
    {
        List<IntPoint> availableLocations = new List<IntPoint>();
        switch (holdingType)
        {
            case WHITE:
                int[] whiteIndexes = { checkerIndex - 8 - 1, checkerIndex - 8 + 1, checkerIndex }; //индексы расположения
                foreach (int tempIndex in whiteIndexes) //пробегаем по всем индексам
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
//генерируем точку и добавляем ее в список
                break;
            case BLACK:
                int[] blackIndexes = { checkerIndex + 8 - 1, checkerIndex + 8 + 1, checkerIndex };
                foreach (int tempIndex in blackIndexes)
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
                break;
            case BLACKKING:
            case WHITEKING:
                int[] kingIndexes = { checkerIndex + 8 - 1, checkerIndex + 8 + 1, checkerIndex - 8 - 1, checkerIndex - 8 + 1, checkerIndex};
                foreach (int tempIndex in kingIndexes)
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
                break;
        }

        //возвращаем список точек
```

```
        return availableLocations;
    }

    // Эта функция будет генерировать "второй этап" для удаления шашки в соответствии с
индексом и типом
    private List<IntPoint> GetSecFloorLocations(int checkerIndex, int holdingType)
    {
        //Работает очень схоже с первым этапом
        //но с другими индексами
        List<IntPoint> availableLocations = new List<IntPoint>();
        switch (holdingType)
        {
            case WHITE:
                int[] whiteIndexes = { checkerIndex - 8 - 8 - 1 - 1, checkerIndex - 8 - 8
+ 1 + 1, checkerIndex };
                foreach (int tempIndex in whiteIndexes)
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
                break;
            case BLACK:
                int[] blackIndexes = { checkerIndex + 8 + 8 - 1 - 1, checkerIndex + 8 + 8
+ 1 + 1, checkerIndex};
                foreach (int tempIndex in blackIndexes)
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
                break;
            case BLACKKING:
            case WHITEKING:
                int[] kingIndexes = { checkerIndex + 8 + 8 -1 - 1, checkerIndex + 8 + 8 +
1 + 1, checkerIndex - 8 -8 - 1 - 1, checkerIndex - 8 - 8 + 1 + 1 };
                foreach (int tempIndex in kingIndexes)
                    availableLocations.Add(this.GetLocationFromIndex(tempIndex));
                break;
        }

        return availableLocations;
    }

    // Функция выполняет логическое удержание
    public void LogicalHold(int indexToHold)
    {
        //удержание проверки
        this.isHolding = true;
        this.whatHolding = this.GetValueAtIndex(indexToHold);
        this.heldIndex = indexToHold;

        //удаление шашки с доски
        this.InsertValueIntoGameMat(USEABLEBS, this.GetLocationFromIndex(indexToHold));
    }

    // удержание
    public void Hold(int index)
    {
        //удержание для второго удаления шашки
        if (this.isThereAnotherPossibleEat && this.SecEatStartIndex == index)
        {
            int valueAtIndex = this.GetValueAtIndex(index);
            //если было удержание
            if (!(this.isHolding) && valueAtIndex <= 4 && valueAtIndex >= 0)
            {
```

```
                //и держим белую шашку или белую королеву
                if   ((this.tm.isFirstPlayerTurn())   &&   (valueAtIndex   ==   WHITE   ||
valueAtIndex == WHITEKING))
                {
                    LogicalHold(index);
                }

                //и держим черную шашку или черную королеву
                else  if  ((!this.tm.isFirstPlayerTurn())  &&  (valueAtIndex  ==  BLACK  ||
valueAtIndex == BLACKKING))
                {
                    LogicalHold(index);
                }
            }
        }
        else
        {
            int valueAtIndex = this.GetValueAtIndex(index);
            //если было удержание
            if (!(this.isHolding) && valueAtIndex <= 4 && valueAtIndex >= 0)
            {
                //и держим белую шашку или белую королеву
                if   ((this.tm.isFirstPlayerTurn())   &&   (valueAtIndex   ==   WHITE   ||
valueAtIndex == WHITEKING))
                    LogicalHold(index);

                //и держим черную шашку или черную королеву
                else  if  ((!this.tm.isFirstPlayerTurn())  &&  (valueAtIndex  ==  BLACK  ||
valueAtIndex == BLACKKING))
                    LogicalHold(index);
            }
        }

    }

    // отпускание удержания
    public  void  LogicalRelease(IntPoint  releaseLocation,  int  valueToInsert,  IntPoint
possibleEnemyLocation = null)
    {
        if                    (this.isThereAnotherPossibleEat                          &&
this.possibleSecEatReleaseLocations.Contains(releaseLocation))
        {
            //вставка королев
            if (valueToInsert == WHITE && releaseLocation.Y == 0)
                this.InsertValueIntoGameMat(WHITEKING, releaseLocation);
            else if (valueToInsert == BLACK && releaseLocation.Y == 7)
                this.InsertValueIntoGameMat(BLACKKING, releaseLocation);
            else
                this.InsertValueIntoGameMat(valueToInsert, releaseLocation);

            //удаление согласно значениям
            if (possibleEnemyLocation != null)
            {
                this.InsertValueIntoGameMat(USEABLEBS, possibleEnemyLocation);
                this.CheckForAnotherPossibleEat(releaseLocation);
            }

            //Наступающий ход
```

```
            if    ((!(releaseLocation.Equals(this.GetLocationFromIndex(heldIndex))))    &&
(!(this.isThereAnotherPossibleEat)))
                this.tm.NextTurn();

            //логические отпускание
            this.heldIndex = NONE;
            this.whatHolding = NONE;
            this.isHolding = false;

            wm.CheckForWinner(this.gameMat);//проверяем на конец игры
        }

        else
        {
            //вставка королев
            if (valueToInsert == WHITE && releaseLocation.Y == 0)
                this.InsertValueIntoGameMat(WHITEKING, releaseLocation);
            else if (valueToInsert == BLACK && releaseLocation.Y == 7)
                this.InsertValueIntoGameMat(BLACKKING, releaseLocation);
            else
                this.InsertValueIntoGameMat(valueToInsert, releaseLocation);

            //удаление согласно значениям
            if (possibleEnemyLocation != null)
            {
                this.InsertValueIntoGameMat(USEABLEBS, possibleEnemyLocation);
                this.CheckForAnotherPossibleEat(releaseLocation);
            }

            //Наступающий ход
            if    ((!(releaseLocation.Equals(this.GetLocationFromIndex(heldIndex))))    &&
(!(this.isThereAnotherPossibleEat)))
                this.tm.NextTurn();

            //логические отпускание
            this.heldIndex = NONE;
            this.whatHolding = NONE;
            this.isHolding = false;

            wm.CheckForWinner(this.gameMat);
        }

    }

    // Проверяет, можно ли удалить еще
    public void CheckForAnotherPossibleEat(IntPoint startLocation)
    {
        this.ResetAnotherPossibleEat();
        int   startIndex   =   this.GetIndexFromLocation(startLocation),   valueAtIndex   =
this.GetValueAtIndex(startIndex);
        List<IntPoint>   firstFloorLocation   =   this.GetFirstFloorLocations(startIndex,
valueAtIndex),     secoundFloorLocations     =     this.GetSecFloorLocations(startIndex,
valueAtIndex);
        for (int i = 0; i < secoundFloorLocations.Count; i++)
        {
            if
(this.IsIndexOkToMove(this.GetIndexFromLocation(secoundFloorLocations.ElementAt(i))))
            {
```

```
                int                             valueAtFirstFloor                               =
this.GetValueAtIndex(this.GetIndexFromLocation(firstFloorLocation.ElementAt(i)));
                if   ((valueAtIndex   ==   WHITE   ||   valueAtIndex   ==   WHITEKING)   &&
(valueAtFirstFloor == BLACK || valueAtFirstFloor == BLACKKING))
                    this.SetAnotherEatPossible(startIndex,
secoundFloorLocations.ElementAt(i));
                else  if  ((valueAtIndex  ==  BLACK  ||  valueAtIndex  ==  BLACKKING)  &&
(valueAtFirstFloor == WHITE || valueAtFirstFloor == WHITEKING))
                    this.SetAnotherEatPossible(startIndex,
secoundFloorLocations.ElementAt(i));
            }
        }
    }

    // Устанавливаем необходимые значения, если можно удалить еще
    private void SetAnotherEatPossible(int startIndex, IntPoint locationToAdd)
    {
        this.isThereAnotherPossibleEat = true;
        this.SecEatStartIndex = startIndex;
        this.possibleSecEatReleaseLocations.Add(locationToAdd);
    }

    // Сбрасывает необходимые значения, если когда есть возможность удалить еще
    private void ResetAnotherPossibleEat()
    {
        this.isThereAnotherPossibleEat = false;
        this.possibleSecEatReleaseLocations = new List<IntPoint>();
        this.SecEatStartIndex = NONE;
    }

    // Отпускание
    public void Release(int index)
    {
        if (this.isHolding && this.IsIndexOkToMove(index))
        {
            List<IntPoint>                          firstFloorLocations                          =
this.GetFirstFloorLocations(this.heldIndex, this.whatHolding);
            IntPoint releaseLocation = this.GetLocationFromIndex(index);

            if                (firstFloorLocations.Contains(releaseLocation)                &&
!this.isThereAnotherPossibleEat)
            {
                LogicalRelease(releaseLocation, this.whatHolding);
            }

            else
            {
                List<IntPoint>                      secoundFloorLocations                       =
this.GetSecFloorLocations(this.heldIndex, this.whatHolding);

                if (secoundFloorLocations.Contains(releaseLocation))
                {
                    IntPoint                      possibleEnemyLocation                         =
firstFloorLocations.ElementAt(secoundFloorLocations.IndexOf(releaseLocation));
                    int                     valueAtPossibleEnemyLocation                        =
this.GetValueAtIndex(this.GetIndexFromLocation(possibleEnemyLocation));
                    if ((this.whatHolding == WHITE || this.whatHolding == WHITEKING) &&
(valueAtPossibleEnemyLocation == BLACK || valueAtPossibleEnemyLocation == BLACKKING))
```

```
                    {
                        LogicalRelease(releaseLocation,                 this.whatHolding,
possibleEnemyLocation);
                    }

                    else if ((this.whatHolding == BLACK || this.whatHolding == BLACKKING)
&& (valueAtPossibleEnemyLocation == WHITE || valueAtPossibleEnemyLocation == WHITEKING))
                    {
                        LogicalRelease(releaseLocation,                 this.whatHolding,
possibleEnemyLocation);
                    }
                }
            }
        }
    }

    /// Эта функция будет проверять, является ли индекс пригодным для использования -
можем ли мы переместить к нему шашку.
    private bool IsIndexOkToMove(int index){
        return this.GetValueAtIndex(index) == USEABLEBS;
    }

    /// Эта функция возвращает значение индекса на доске
    public int GetValueAtIndex(int index){
        int tempIndex;
        for (int i = 0; i < N; i++){
            for (int j = 0; j < N; j++){
                tempIndex = (i * N) + j;
                if (index == tempIndex)
                    return this.gameMat[i, j];
            }
        }
        return NONE;
    }
    /// Эта функция возвращает имя заданного значения.
    public string GetNameOfValue(int value){
        if (value == 1)
            return "Black";
        else if (value == 2)
            return "White";
        return "";}}
```