МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

КУРСОВАЯ РАБОТА

по дисциплине

«Объектно-ориентированное программирование»

Тема: «Разработка приложений на основе принципов объектно-ориентированного подхода»

Студент гр. 1302 _____ Новиков Г.В.

Преподаватель: _____ Новакова Н.Е.

Санкт-Петербург
2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Новиков Г.В.

Группа 1302

Тема работы: Разработка приложений на основе принципов объектно-ориентированного подхода

Исходные данные:
Среда разработки: Microsoft VS 2022
Язык программирования: C#

Содержание пояснительной записки:
«Содержание», «Введение», «Цель работы», «Задание», «Теоретические сведения», «Формализация задачи», «Спецификация программы», «Руководство пользователя», «Руководство программиста», «Контрольный пример», «Листинг программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 40 страниц.

Дата выдачи задания: 05.04.2023

Дата сдачи курсовой работы: 07.06.2023

Дата защиты курсовой работы: 07.06.2023

Студент _____ Новиков Г.В.

Преподаватель _____ Новакова Н.Е.

АННОТАЦИЯ

Курсовая работа содержит в себе решение трех задач на основе объектно-ориентированного подхода. В первой задаче рассмотрена иерархия классов, во второй – реализация алгоритма Форда-Фалкерсона на графе, в третьей – разработка имитационной модели.

На основе этих моделей были разработаны приложения, включающие в себя различные пользовательские интерфейсы. Полученные результаты приведены в работе.

SUMMARY

Course work contains the solution of three problems based on an object-oriented approach. In the first task, the class hierarchy is considered, in the second, the implementation of the Ford-Fulkerson algorithm on a graph, in the third, the development of a simulation model.

Based on these models, applications have been developed that include various user interfaces. The results obtained are presented in the work.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Курсовая работа направлена на создание приложений на основе объектно-ориентированного подхода на языке C#. В ней рассматриваются иерархии классов и наследования, а также имитационные модели. Для первых двух задач реализовывается консольное приложение для взаимодействия с пользователем, для третьей используются WinForms.

ЦЕЛЬ РАБОТЫ

Целью курсовой работы является закрепление теоретических знаний и получение практических навыков разработки программного обеспечения на основе объектно-ориентированного подхода.

1. РАЗРАБОТКА ОБЪЕКТНОЙ МОДЕЛИ

1.1. Задание

Вариант 18

Разработать программу для представления склада лекарств.

1.2. Теоретические сведения

Склад лекарств представляет собой коммерческое заведение, в котором представлены различные товары для совершения заказа с соответствующими ценами и указанным количеством.

1.3. Формализация задачи

Program – основной класс, содержащий метод Main. Обеспечивает взаимодействие пользователя со складом.

IStoring – интерфейс для классов, способных хранить лекарства. Содержит поле Drugs, а также методы Add и Remove для добавления и удаления лекарств.

Класс DrugStorage представляет собой склад лекарств. Реализует IStoring. DrugStorage имеет поле Drugs, являющееся списком, хранящим лекарства типа Drug. Каждый объект в списке представляет определенное лекарство, причем новый объект добавляется только если на складе нет ни одного лекарства с таким названием, в противном случае увеличивается поле Drug.Quantity соответствующего лекарства. Для проверки наличия товара на складе используется метод Available.

Класс Order используется для создания заказа и его отправления клиенту. Реализует IStoring. Имеет поле Drugs – товары в заказе, добавление и удаление происходит таким же образом, как в DrugStorage. TotalPrice – поле, представляющее сумму заказа.

Для представления лекарств используются классы, наследующие Drug – Antidepressant, Barbirurate и Hormone. Они представляют тип лекарства,

конкретное лекарство должно быть представлено одним из этих классов с соответствующим именем, ценой и количеством, задаваемыми в конструкторе.

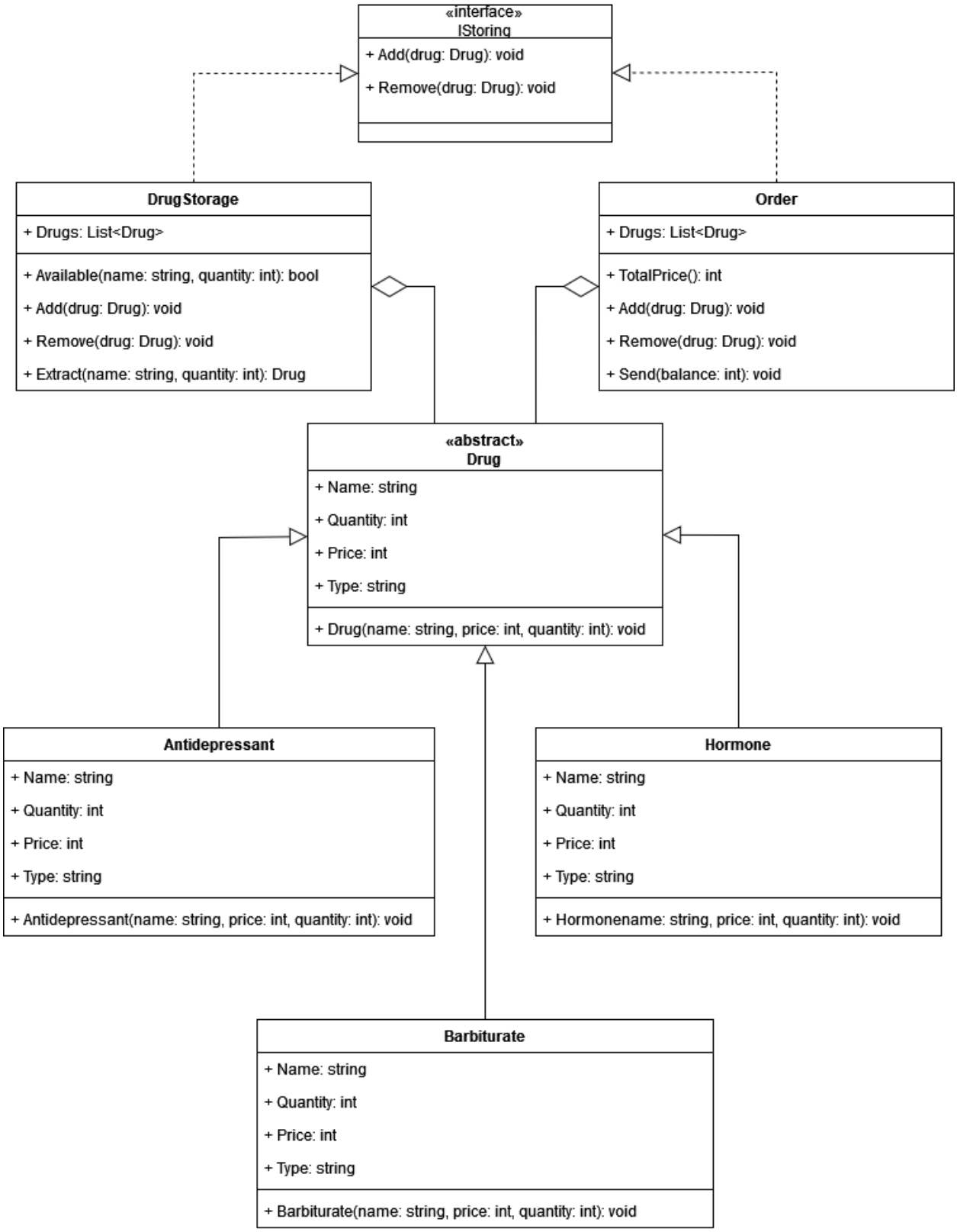

Рисунок 1.3.1. UML диаграмма классов

1.4. Спецификация программы

Таблица 1.3.1 Описание методов класса DrugStorage

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Available	bool	public	string name, int quantity	Проверка наличия товаров на складе
Add	void	public	Drug drug	Добавление товаров на склад
Remove	void	public	Drug drug	Удаление товаров со склада
Extract	Drug	public	string name, int quantity	Извлечение товаров со склада

Таблица 1.3.2 Описание методов класса Order

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Send	void	public	int balance	Отправление заказа
Add	void	public	Drug drug	Добавление товаров в заказ
Remove	void	public	Drug drug	Удаление товаров в заказ

1.5. Руководство пользователя

1.5.1. Назначение программы

Заказ лекарств со склада.

1.5.2. Условие выполнения программы:

Для запуска программы необходима установленная операционная система Windows, программа Microsoft Visual Studio, а также установлена платформа .NET Framework не ниже версии 4.5.2.

1.5.3. Выполнение программы:

1. Пользователь вводит баланс;
2. Выводится список товаров, доступных на складе для добавления в заказ;
3. Пользователь вводит выбор товара;
4. Пользователь вводит количество единиц товара;
5. Товар добавляется в заказ и удаляется со склада;
6. Выводится информация о заказе;
7. Выводится список доступных действий (добавить в заказ, удалить из заказа или отправить заказ);
8. Пользователь выбирает действие:
   - 8.1. Добавить лекарство в заказ – выполняются пункты 2 – 8;
   - 8.2. Удалить лекарство из заказа:
     - 8.2.1. Выводится список товаров в заказе;
     - 8.2.2. Пользователь выбирает товар;
     - 8.2.3. Пользователь вводит количество;
     - 8.2.4. Товар удаляется из заказа и возвращается на склад;
     - 8.2.5. Выполняются пункты 6 – 8;
   - 8.3. Отправить заказ:
     - 8.3.1. Выводится информация об отправленном заказе;
     - 8.3.2. Выводится новый баланс;
     - 8.3.3. Программа завершается;

1.6. Руководство программиста

1.6.1. Запуск программы:

Необходима операционная система Windows. При разработке использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2.

1.6.2. Характеристика программы

Программа имеет список команд для функционирования, позволяя продемонстрировать работу склада.

1.6.3. Входные и выходные данные

Программа не предполагает взаимодействие с пользователем напрямую. Новые данные об объектах вносит программист. Результатом служит информация об отправленном заказе.

1.7. Контрольный пример

На рисунке 1.7.1 было произведено добавление 3 товаров под номером 4 в заказ и отправка заказа:

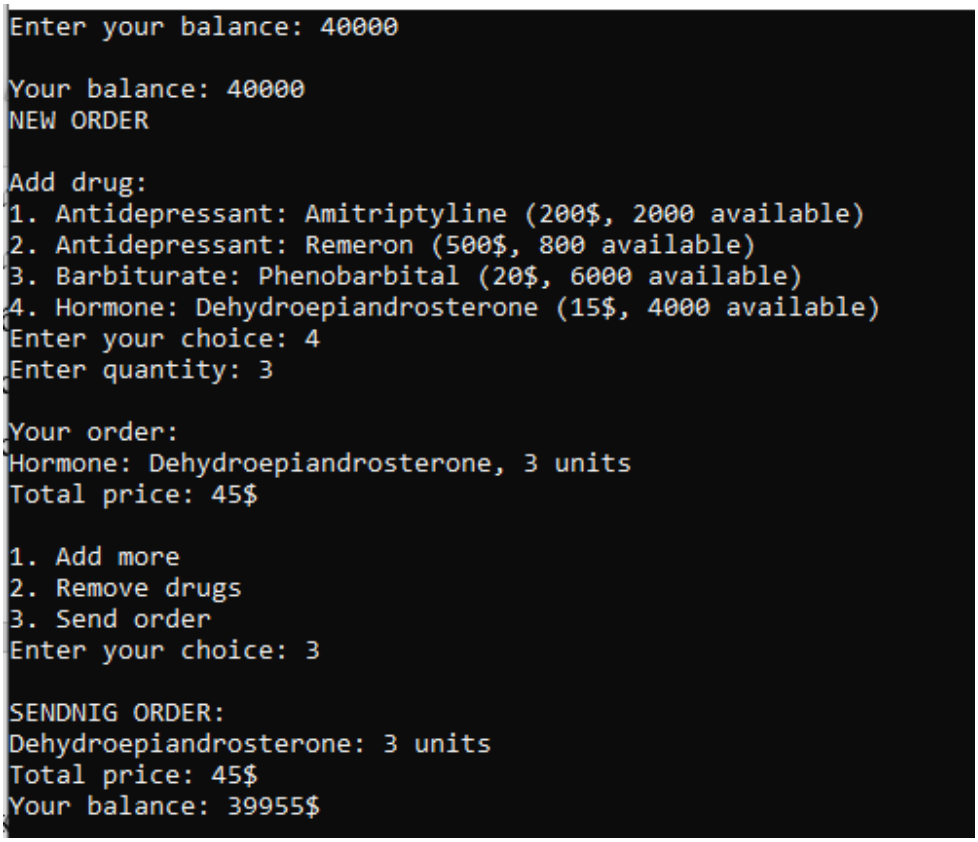

Рисунок 1.7.1 Работа программы

На рисунке 1.7.2 в заказ добавляются товары, затем часть товаров удаляется, добавляются новые и производится отправление заказа:

```
Enter your balance: 2000

Your balance: 2000
NEW ORDER

Add drug:
1. Antidepressant: Amitriptyline (200$, 2000 available)
2. Antidepressant: Remeron (500$, 800 available)
3. Barbiturate: Phenobarbital (20$, 6000 available)
4. Hormone: Dehydroepiandrosterone (15$, 4000 available)
Enter your choice: 1
Enter quantity: 13

Your order:
Antidepressant: Amitriptyline, 13 units
Total price: 2600$

1. Add more
2. Remove drugs
3. Send order
Enter your choice: 2

Remove drugs:
1. Antidepressant: Amitriptyline (200$, 13 units)
Enter your choice: 1
Enter quantity: 9

Your order:
Antidepressant: Amitriptyline, 4 units
Total price: 800$

1. Add more
2. Remove drugs
3. Send order
Enter your choice: 1

Add drug:
1. Antidepressant: Amitriptyline (200$, 1996 available)
2. Antidepressant: Remeron (500$, 800 available)
3. Barbiturate: Phenobarbital (20$, 6000 available)
4. Hormone: Dehydroepiandrosterone (15$, 4000 available)
Enter your choice: 2
Enter quantity: 2

Your order:
Antidepressant: Amitriptyline, 4 units
Antidepressant: Remeron, 2 units
Total price: 1800$

1. Add more
2. Remove drugs
3. Send order
Enter your choice: 3

SENDNIG ORDER:
Amitriptyline: 4 units
Remeron: 2 units
Total price: 1800$
Your balance: 200$
```

Рисунок 1.7.2 Работа программы

На рисунке 1.7.3 приведен пример, в котором товара нет на складе в указанном пользователем количестве (программа не завершается):

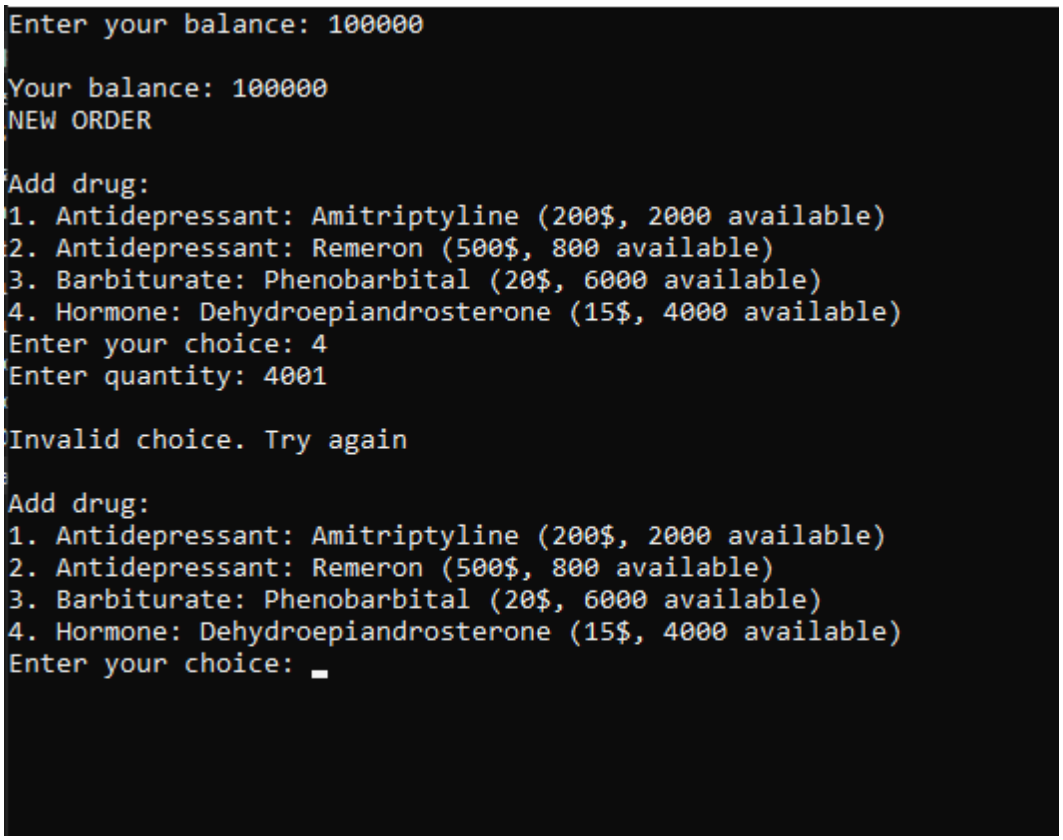

Рисунок 1.7.3 Работа программы

На рисунке 1.7.4 показан некорректный ввод:

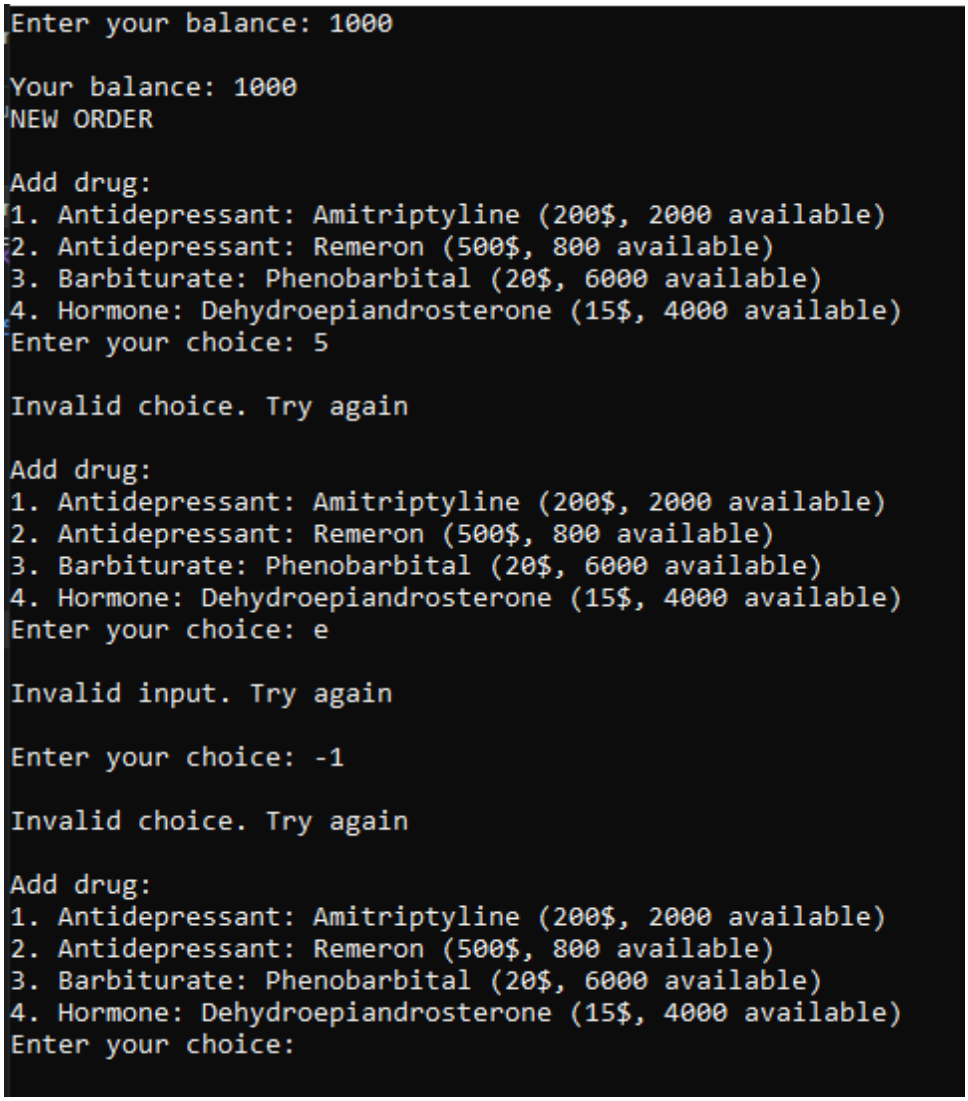

Рисунок 1.7.4 Работа программы

1.8. Листинг программы

DrugStorage.cs:

```
using System;

public class DrugStorage : IStoring
{
    private List<Drug> drugs = new List<Drug>();
    public List<Drug> Drugs
    {
        get { return drugs; }
        set { drugs = value; }
    }

    public bool Available(string name, int quantity = 1)
    {
        Drug? d = Drugs.Find(d => d.Name == name);
        if (d is null) return false;
        if (d.Quantity < quantity) return false;
        return true;
    }
```

```
    public void Add(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is not null) d.Quantity += drug.Quantity;
        else
        {
            Drug? new_drug = Activator.CreateInstance(drug.GetType(), new
object[] { drug.Name, drug.Price, drug.Quantity }) as Drug;
            if (new_drug is not null) Drugs.Add(new_drug);
        }
    }

    public void Remove(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < drug.Quantity) throw new ArgumentException("Trying to
remove more drugs than storage has");

        if (d.Quantity == drug.Quantity) Drugs.Remove(d);
        else d.Quantity -= drug.Quantity;
    }

    public Drug Extract(string name, int quantity)
    {
        Drug? d = Drugs.Find(d => d.Name == name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < quantity) throw new ArgumentException("Not enough
drugs in the storage");

        Drug? new_drug = Activator.CreateInstance(d.GetType(), new object[] {
d.Name, d.Price, quantity }) as Drug;
        if (new_drug is not null) Remove(new_drug);

        return new_drug;
    }
}
```

Order.cs:

```
using System;

public class Order : IStoring
{
    private List<Drug> drugs = new List<Drug>();
    public List<Drug> Drugs
    {
        get { return drugs; }
        set { drugs = value; }
    }

    public int TotalPrice
    {
        get
        {
            int total_price = 0;
            foreach (Drug drug in Drugs) total_price += drug.Price *
drug.Quantity;
            return total_price;
        }
    }

    public void Add(Drug drug)
```

```
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is not null) d.Quantity += drug.Quantity;
        else
        {
            Drug? new_drug = Activator.CreateInstance(drug.GetType(), new
object[] { drug.Name, drug.Price, drug.Quantity }) as Drug;
            if (new_drug is not null) Drugs.Add(new_drug);
        }
    }

    public void Remove(Drug drug)
    {
        Drug? d = Drugs.Find(d => d.Name == drug.Name);
        if (d is null) throw new ArgumentException("Drug is not found");
        if (d.Quantity < drug.Quantity) throw new ArgumentException("Trying to
remove more drugs than storage has");

        if (d.Quantity == drug.Quantity) Drugs.Remove(d);
        else d.Quantity -= drug.Quantity;
    }

    public void Clear()
    {
        Drugs = new List<Drug>();
    }

    public void Send(int balance)
    {
        if (Drugs.Count == 0)
        {
            Console.WriteLine("Cannot send order. Order is empty");
            return;
        }
        if (balance < TotalPrice) Console.WriteLine("Cannot send order. Balance
is too low");
        Console.WriteLine();
        Console.WriteLine("SENDNIG ORDER:");
        foreach (Drug drug in Drugs) Console.WriteLine($"{drug.Name}:
{drug.Quantity} units");
        Console.WriteLine($"Total price: {TotalPrice}$");
        Clear();
    }
}
```

IStoring.cs:

```
using System;
using System.ComponentModel.DataAnnotations;

public interface IStoring
{
    public List<Drug> Drugs { get; set; }
    void Add(Drug drug);
    void Remove(Drug drug);
}
```

Drug.cs:

```
using System;

public abstract class Drug
```

```
{
    public int Price;
    public int Quantity;
    public string Type = "";
    private string name = "";
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public Drug(string name, int price, int quantity)
    {
        Name = name;
        Price = price;
        Quantity = quantity;
    }


    public class Antidepressant : Drug
    {
        public Antidepressant(string name, int price, int quantity) :
base(name, price, quantity)
        {
            Type = "Antidepressant";
        }
    }


    public class Barbiturate : Drug
    {
        public Barbiturate(string name, int price, int quantity) : base(name,
price, quantity)
        {
            Type = "Barbiturate";
        }
    }


    public class Hormone : Drug
    {
        public Hormone(string name, int price, int quantity) : base(name,
price, quantity)
        {
            Type = "Hormone";
        }
    }
}
```

2. РАБОТА С ГРАФАМИ

2.1. Задание

Вариант Г-42-3

Найти максимальный поток в сети с помощью алгоритма Форда-Фалкерсона. Задачу решить в общем виде. В качестве контрольного примера использовать задание соответствующего варианта.

В качестве контрольного примера используются следующий граф:

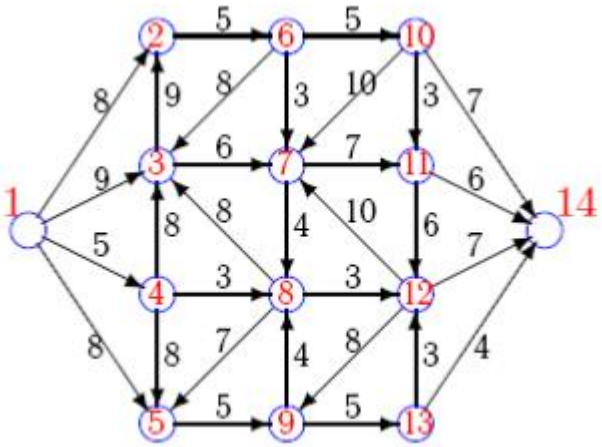

Рисунок 2.1.1 Граф для контрольного примера

2.2. Теоретические сведения

Граф (G(V,E)) – совокупность двух множеств — множества объектов V, называемого множеством вершин, и множества их парных связей E, называемого множеством рёбер. Элемент множества рёбер есть пара элементов множества вершин.

Взвешенный граф – граф, в котором каждому ребру поставлено в соответствие некоторое число, называемое весом ребра.

Матрица смежности – таблица, где как столбцы, так и строки соответствуют вершинам графа. В каждой ячейке этой матрицы записывается число, определяющее наличие связи от вершины-строки к вершине-столбцу (либо наоборот).

В теории графов транспортная сеть — ориентированный граф G = (V, E), в котором каждое ребро (u, v) ∈ E имеет неотрицательную пропускную способность c (u, v) ⩾ 0 и поток f(u,v). Выделяются две вершины: источник s и сток t такие, что любая другая вершина сети лежит на пути из s в t, при этом s ≠ t. Транспортная сеть может быть использована для моделирования, например, дорожного трафика.

Алгоритм Форда — Фалкерсона решает задачу нахождения максимального потока в транспортной сети. Идея алгоритма заключается в следующем. Изначально величине потока присваивается значение 0: f(u,v)=0 для всех u, v ∈ V. Затем величина потока итеративно увеличивается посредством поиска увеличивающего пути (путь от источника s к стоку t, вдоль которого можно послать больший поток). Процесс повторяется, пока можно найти увеличивающий путь.

2.3. Формализация задачи

FordFulkerson – класс, решающий задачу нахождени максимального потока с помощью алгоритма Форда-Фалкерсона

Класс Program содержит Main и обеспечивает вывод результатов.

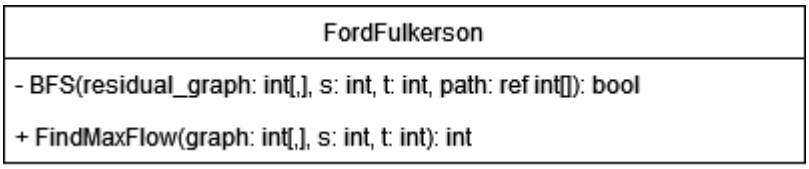

Рисунок 2.3.1 UML Диаграмма классов

2.4. Спецификация программы

Таблица 2.4.1 Описание методов класса FodrFulkerson

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
FindMaxFlow	int	public	int[,] graph, int s, int t	Нахождение максимального потока в сети
BFS	bool	private	int[,] residual_graph, int s, int t, ref int[] path	Поиск в ширину

2.5. Руководство пользователя

2.5.1 Назначение программы

Данная программа позволяет найти максимальный поток в заданной сети с заданным истоком и стоком.

2.5.2 Условие выполнения программы:

Для запуска программы необходима установленная операционная система Windows, программа Microsoft Visual Studio, а также установлена платформа .NET Framework не ниже версии 4.5.2.

2.5.3 Выполнение программы:

Программа разработана в виде консольного приложения. Она считывает матрицу смежности, исток и сток из метода Main и выводит максимальный поток в консоль.

2.6 Руководство программиста

2.6.1 Запуск программы:

Необходима операционная система Windows. При разработке использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2.

2.6.2 Характеристика программы

Программа решает задачу в общем виде, находит максимальный поток с помощью алгоритма Форда-Фалкерсона в сети по матрице смежности, заданной в методе Main.

2.6.3 Входные и выходные данные

Программа не предполагает взаимодействие с пользователем напрямую. Новые данные вносит программист (изменяет матрицу смежности и вершины s и t). Результатом служит вывод максимального потока в сети в консоль.

2.7 Контрольный пример

На рисунке 2.7.1 – работа программы для графа, данного в качестве контрольного примера:

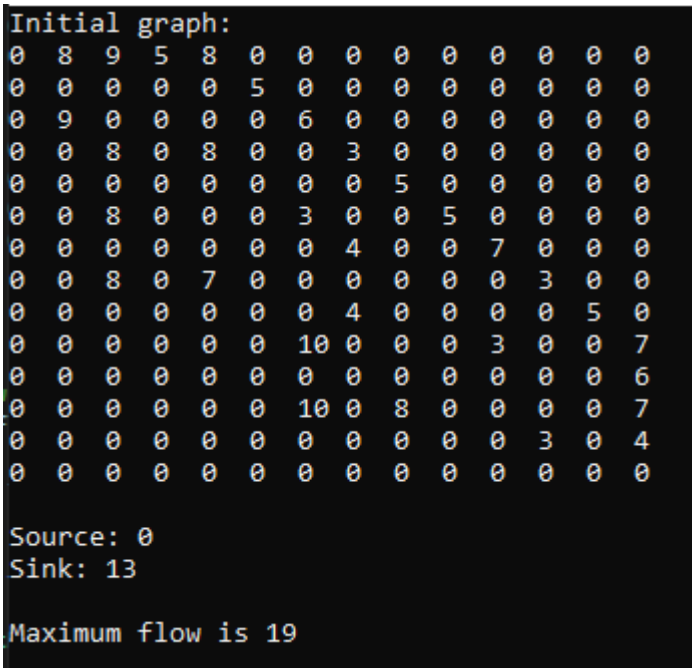

Рисунок 2.7.1 Работа программы

2.8 Листинг программы

Program.cs:

```
using System;
using System.Collections.Generic;


namespace part_2
{
    internal class Program
    {
        public static void Main()
        {
```

```
            // input
            int[,] graph = new int[,] {
                { 0, 8, 9, 5, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 9, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 8, 0, 8, 0, 0, 3, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0 },
                { 0, 0, 8, 0, 0, 0, 3, 0, 0, 5, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 7, 0, 0, 0 },
                { 0, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 3, 0, 0 },
                { 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 5, 0 },
                { 0, 0, 0, 0, 0, 0, 10, 0, 0, 0, 3, 0, 0, 7 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6 },
                { 0, 0, 0, 0, 0, 0, 10, 0, 8, 0, 0, 0, 0, 7 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 4 },
                { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
            };
            int t = 0;
            int s = 13;

            // finding flow
            FordFulkerson flow_finder = new FordFulkerson();

            int max_flow = -1;
            try
            {
                max_flow = flow_finder.FindMaxFlow(graph, t, s);
            }
            catch (ArgumentException e)
            {
                Console.Write("Error: ");
                Console.WriteLine(e.Message);
                Environment.Exit(1);
            }

            // output
            Console.WriteLine("Initial graph:");
            for (int u = 0; u < graph.GetLength(0); u++)
            {
                for (int v = 0; v < graph.GetLength(1); v++)
                    Console.Write("{0, -2} ", graph[u, v]);
                Console.WriteLine();
            }

            Console.WriteLine();
            Console.WriteLine($"Source: {t}");
            Console.WriteLine($"Sink: {s}");
            Console.WriteLine();
            Console.WriteLine($"Maximum flow is {max_flow}");
        }
    }
}
```

FordFulkerson.cs:

```
using System;

public class FordFulkerson
{
    private bool BFS(int[,] residual_graph, int s, int t, ref int[] path)
    {
        bool[] visited = new bool[residual_graph.GetLength(0)];
        for (int i = 0; i < residual_graph.GetLength(0); ++i)
            visited[i] = false;

        List<int> queue = new List<int>();
```

```
            queue.Add(s);
            visited[s] = true;
            path[s] = -1;

            while (queue.Count != 0)
            {
                int u = queue[0];
                queue.RemoveAt(0);

                for (int v = 0; v < residual_graph.GetLength(0); v++)
                {
                    if (visited[v] == false && residual_graph[u, v] > 0)
                    {
                        if (v == t)
                        {
                            path[v] = u;
                            return true;
                        }
                        queue.Add(v);
                        path[v] = u;
                        visited[v] = true;
                    }
                }
            }

            // didn't reach t
            return false;
        }

        public int FindMaxFlow(int[,] graph, int s, int t)
        {
            if (graph.GetLength(0) != graph.GetLength(1)) throw new
    ArgumentException("Graph adjacency matrix is not square");
            int len = graph.GetLength(0);

            if (s >= len || s < 0) throw new ArgumentException("Invalid s");
            if (t >= len || t < 0) throw new ArgumentException("Invalid t");

            int u, v;
            int[,] residual_graph = new int[len, len];

            for (u = 0; u < len; u++)
                for (v = 0; v < len; v++)
                    residual_graph[u, v] = graph[u, v];

            int[] path = new int[len];

            int max_flow = 0;

            // while there is path from source to sink
            while (BFS(residual_graph, s, t, ref path))
            {
                int path_flow = int.MaxValue;
                for (v = t; v != s; v = path[v])
                {
                    u = path[v];
                    path_flow = Math.Min(path_flow, residual_graph[u, v]);
                }

                for (v = t; v != s; v = path[v])
                {
                    u = path[v];
                    residual_graph[u, v] -= path_flow;
                    residual_graph[v, u] += path_flow;
                }
```

```
            max_flow += path_flow;
        }

        return max_flow;
    }
}
```

3. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

3.1 Задание

Вариант 43

Имитация игры в шашки «вслепую»

Создать программу, имитирующую игру в шашки «вслепую». Программа должна имитировать действия «противника». Противник должен отрабатывать результаты ваших ходов и наносить ответный удар случайным образом.

Разработать объектную структуру программы. Пользовательский интерфейс должен обеспечивать ввод ходов и команд. Необходимо сформировать протокол игры с выдачей его по завершении и запоминанием его «для истории» и с возможностью дальнейшей обработки статистики. Обеспечить вывод результатов данных в текстовый файл.

3.2 Теоретические сведения

Имитационное моделирование заключается в создании процессов близких к реальным.

Правила игры в шашки (русские):

Игра происходит на шахматной доске. В игре принимают участие 2 игрока. Игроки располагаются на противоположных сторонах доски. Игровое поле (доска) располагается таким образом, чтобы угловое темное поле было расположено с левой стороны игрока. Шашки расставляются на трех, ближних к игроку, рядах на темных клетках. Право первого хода принадлежит игроку, который играет белым (светлым) цветом. Ходы осуществляются соперниками поочередно. Взятие шашки соперника производится, переносом через неё своей, в том случае, если она находится на соседней с шашкой диагональной клетке и за ней имеется свободное поле. Если после этого хода имеется продолжение для взятия других шашек соперника, ход продолжается. Шашка (шашки) соперника снимается с доски.

Взятие шашки соперника может производиться, как вперед, так и назад, и является обязательным. Если шашка дошла до последней горизонтали, она становится «дамкой» и обозначается переворачиванием. Дамка может перемещаться по диагоналям на любое количество свободных клеток. Взятие дамкой шашек соперника может осуществляться через любое количество диагональных клеток, при наличии свободного пространства за «жертвой». Если она снова оказывается на одной диагонали рядом или на расстоянии от шашки соперника, за которой находится одно или несколько свободных полей, дамка обязательно должна продолжить взятие последующих и занять любое свободное поле на той же диагонали за последней взятой шашкой. Побеждает игрок, который взял или заблокировал все шашки соперника.

В реализованной игре противника представляет бот, ход которого определяется случайным образом.

3.3 Формализация задачи

Класс ChessBoard представляет доску, на которой происходит игра. Он содержит методы для обработки ходов.

Класс Bot – бот-соперник, совершающий случайный ход.

Piece – абстрактный класс для шашек и дамок.

Checker – шашка

King – дамка

Checkers.Color – enum для цвета. Значения: White, Black

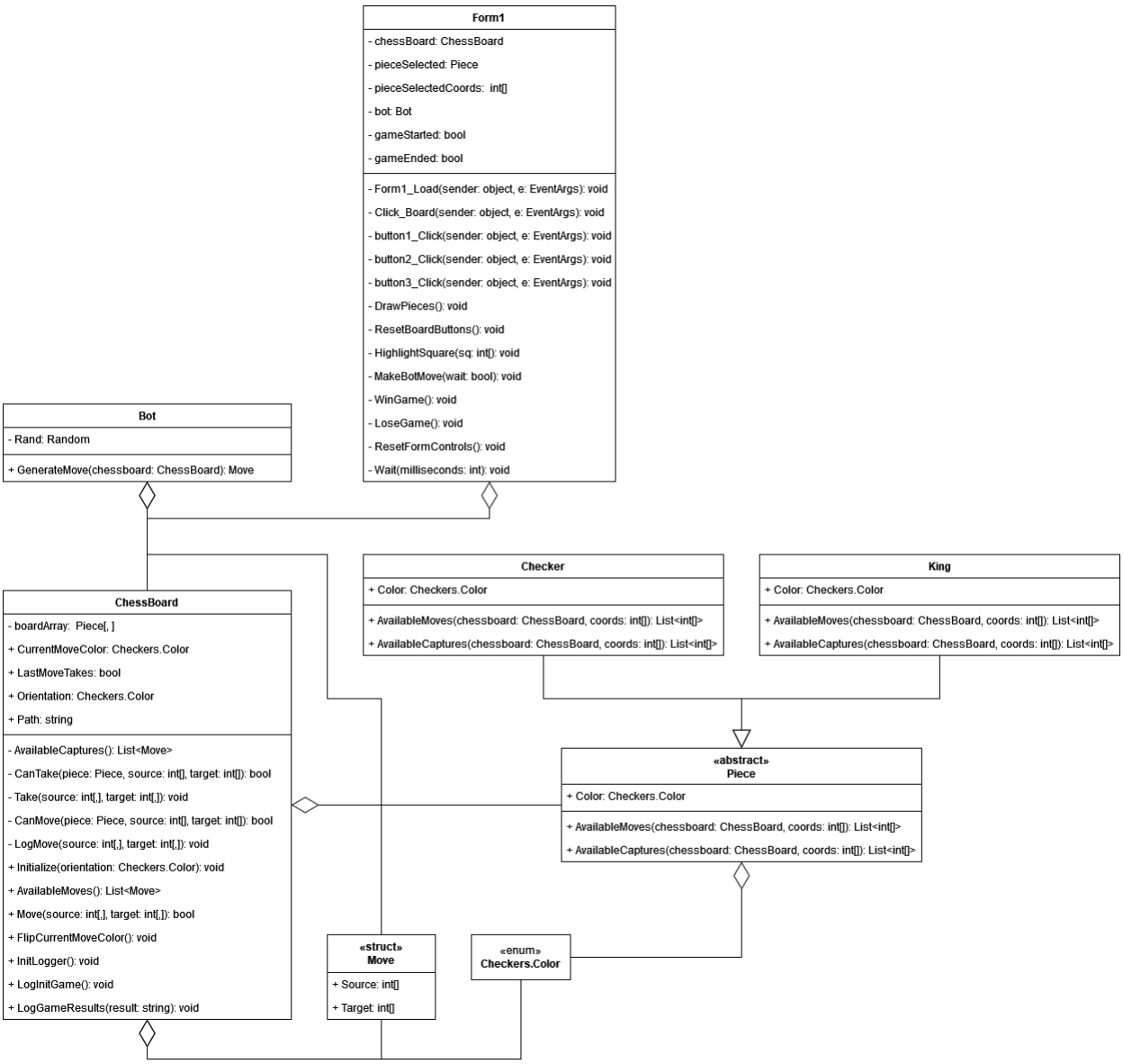

Рисунок 3.3.1 UML Диаграмма классов

3.4 Спецификация программы

Таблица 3.4.1 Описание методов класса ChessBoard

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
Initialize	void	public	Checkers.Color orientation	Инициализация игры. Вызывается перед началом игры

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
AvailableMoves	List<Move>	public	-	Все доступные ходы для текущего игрока
AvailableCaptures	List<Move>	private	-	Все доступные взятия для текущего игрока
CanMove	bool	private	Piece piece, int[] source, int[] target	Шашка может походить на заданное поле
CanTake	bool	private	Piece piece, int[] source, int[] target	Шашка может взять, встав на заданное поле
Take	void	private	int[] source, in int[] target	Метод для взятия другой шашки
Move	bool	public	int[] source, in int[] target	Метод для совершения хода
FlipCurrentMoveColor	void	public	-	Поменять текущего игрока
InitLogger	void	public	-	Готовит файл для вывода
LogInitGame	void	public	-	Выводит информацию об игре в файл
LogMove	void	private	int[] source, int[] target	Выводит ход в файл
LogGameResults	void	public	string result	Выводит результат игры в файл

Таблица 3.4.2 Описание методов классов Checker, King

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
AvailableMoves	List<int[]>	public	ChessBoard chessboard, int[] coords	Доступные ходы для шашки
AvailableCaptures	List<int[]>	public	ChessBoard chessboard, int[] coords	Доступные взятия для шашки

Таблица 3.4.3 Описание методов класса Bot

Методы	Возвращаемый тип	Модификатор доступа	Входные параметры	Назначение
GenerateMove	Move	public	ChessBoard chessboard	Генерирует случайный ход

3.5 Руководство пользователя

3.5.1 Назначение программы

Данная программа позволяет моделировать игру в шашки против компьютера, а также создавать новую игру с возможностью выбора цвета.

3.5.2 Условие выполнения программы:

Для запуска программы необходима установленная операционная система Windows, программа Microsoft Visual Studio, а также установлена платформа .NET Framework не ниже версии 4.5.2.

3.5.3 Выполнение программы:

Программа разработана с использованием WinForms. Пользователю доступна форма, на которой есть доска и 3 кнопки. Кнопка “Play as black” меняет цвет, за который играет игрок, при нажатии текст меняется на “Play as white” или “Play as black” в зависимости от текущего цвета. Кнопка “Start” запускает игру. Если она не нажата, доска не будет реагировать на действия

пользователя. Если пользователь играет за черных, бот совершает первый ход. Кнопка “Reset” возвращает доску к начальному положению (белые фигуры снизу, доска недоступна). После нажатия на нее для запуска игры нужно нажать кнопку “Start”. Также в форме есть текстовое поле, выводящее сообщения при нажатии кнопок или по мере игры.

На рисунке 3.5.1 показано как выглядит форма:

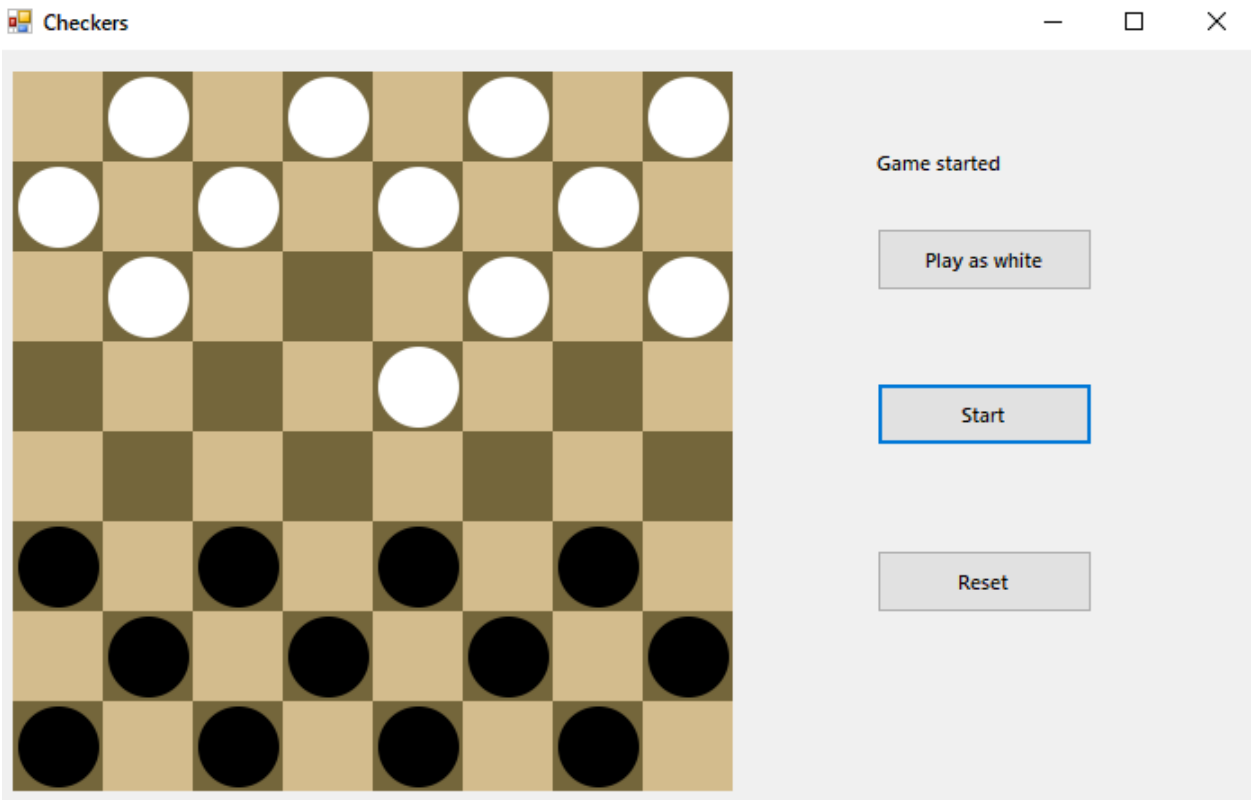

Рисунок 3.5.1 WinForms

На рисунках 3.5.2 – 3.5.5 показано, как выглядят шашки и дамки:

Рисунок 3.5.2 Шашка белая

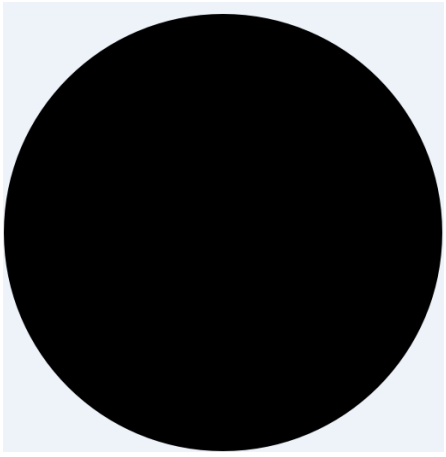

Рисунок 3.5.3 Шашка черная

Рисунок 3.5.4 Дамка белая

Рисунок 3.5.5 Дамка черная

3.6 Руководство программиста

3.6.1 Запуск программы:

Необходима операционная система Windows. При разработке использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2.

3.6.2 Характеристика программы

Программа выполняется с использованием WinForms. Программа имеет интуитивно понятный интерфейс, проста и доступна в использовании.

3.6.3 Входные и выходные данные

Входными данными служат ходы, совершенные игроком. Выходными данными служат доска с шашками, ход компьютера и сообщения об игре. Также протокол игры выводится в текстовый файл.

3.7 Контрольный пример

На рисунке 3.7.1 показана форма при запуске программы:

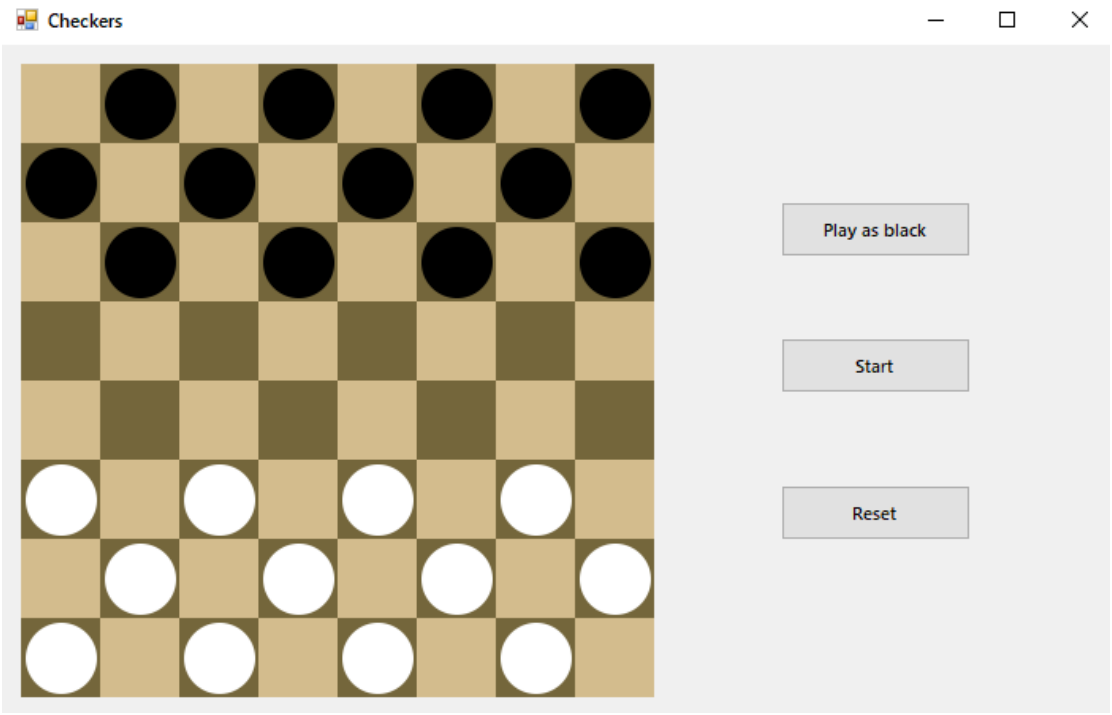

Рисунок 3.7.1 Работа программы

На рисунке 3.7.2 показана форма после нажатия пользователем кнопки “Play as black”:

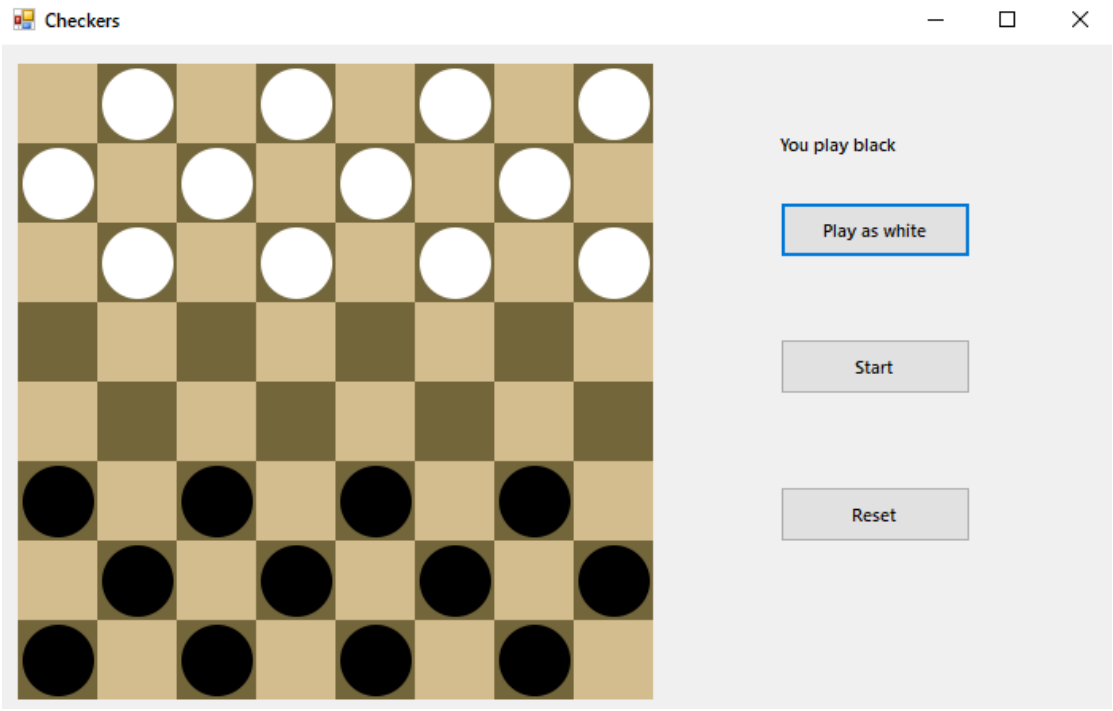

Рисунок 3.7.2 Работа программы

На рисунке 3.7.3 показана форма после нажатия пользователем кнопки “Start”:

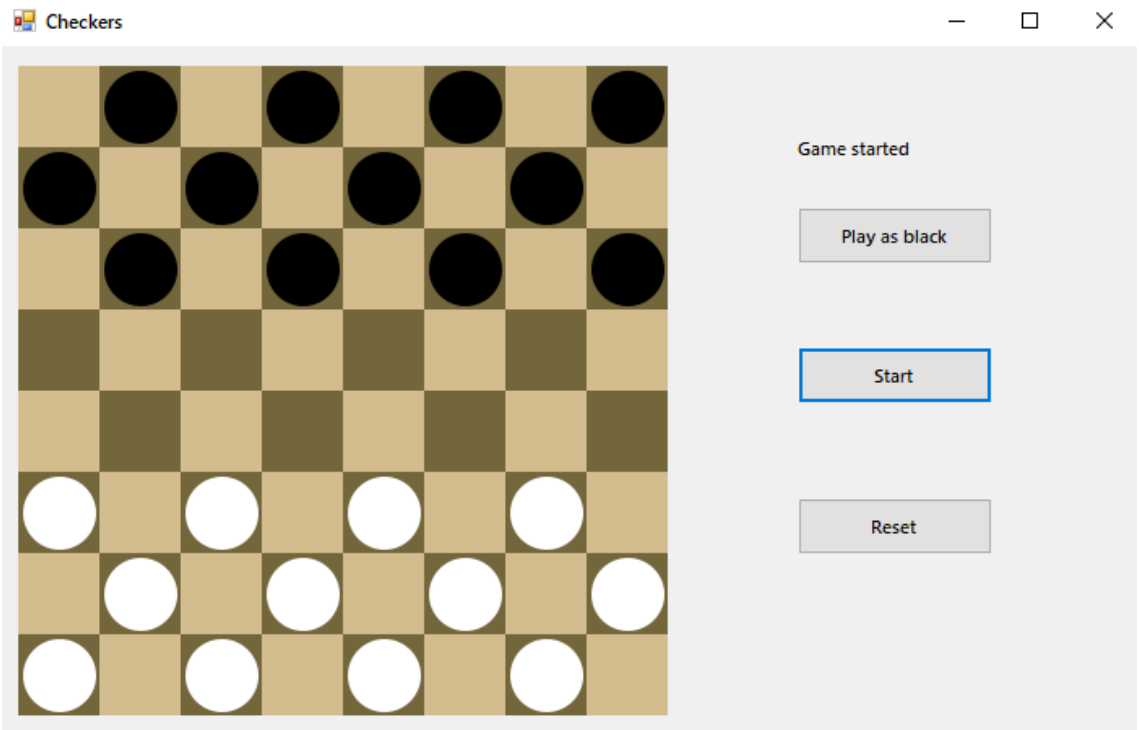

Рисунок 3.7.3 Работа программы

На рисунке 3.7.4 показана форма после нажатия пользователем кнопки “Start” и совершения 2 ходов:

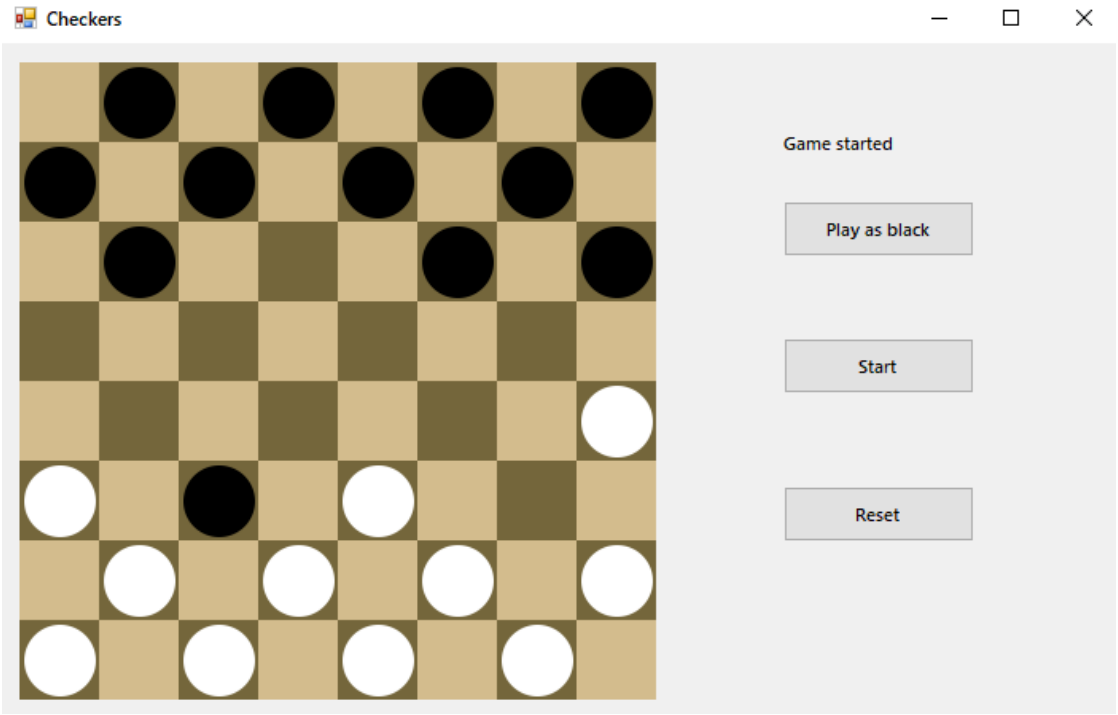

Рисунок 3.7.4 Работа программы

На рисунке 3.7.5 показана форма после нажатия кнопки “Play as black” и кнопки “Start”:

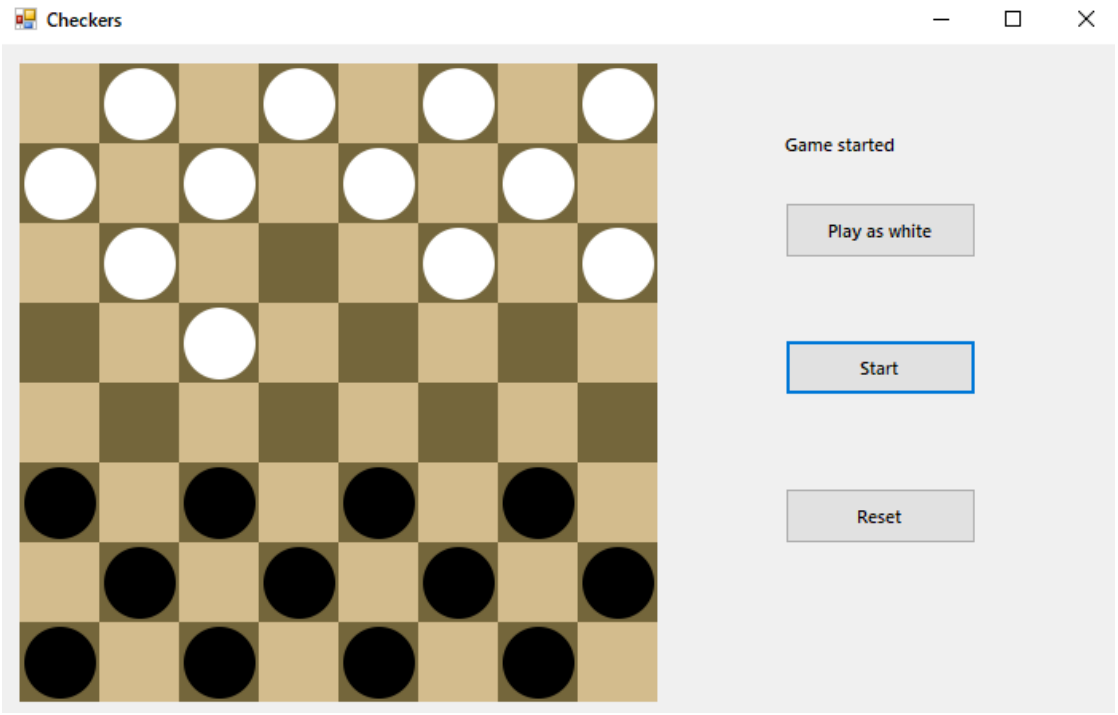

Рисунок 3.7.5 Работа программы

На рисунке 3.7.6 показана форма после превращения белой шашки в дамку:

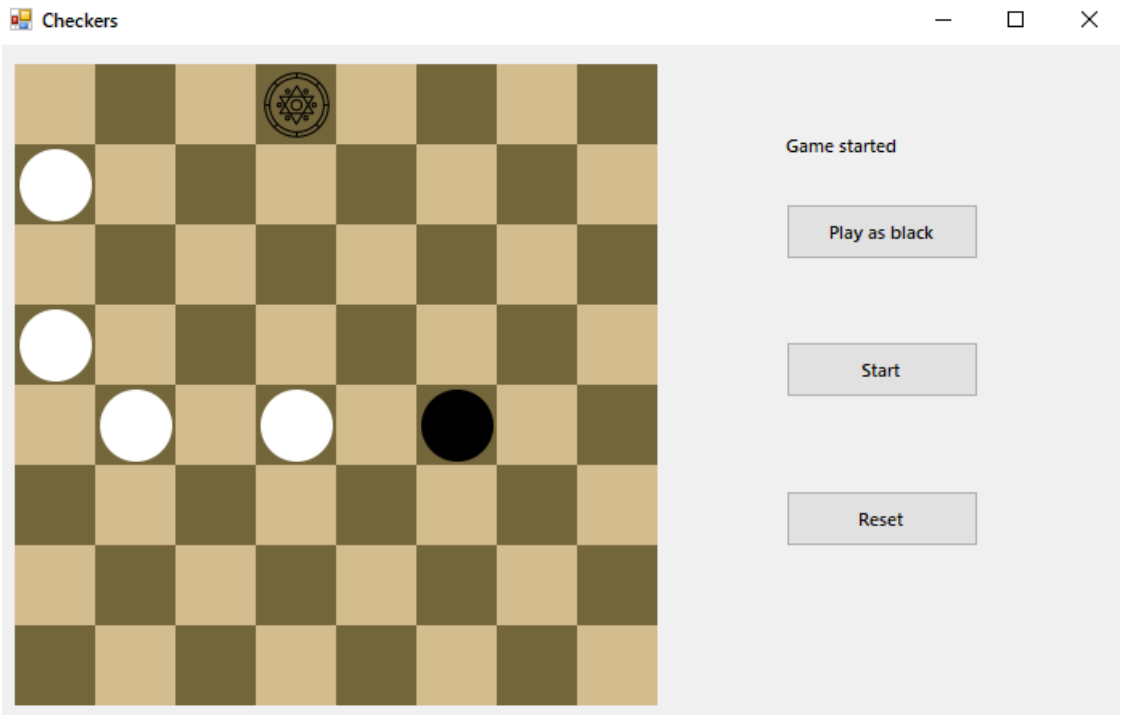

Рисунок 3.7.6 Работа программы

На рисунке 3.7.7 показана форма после победы:

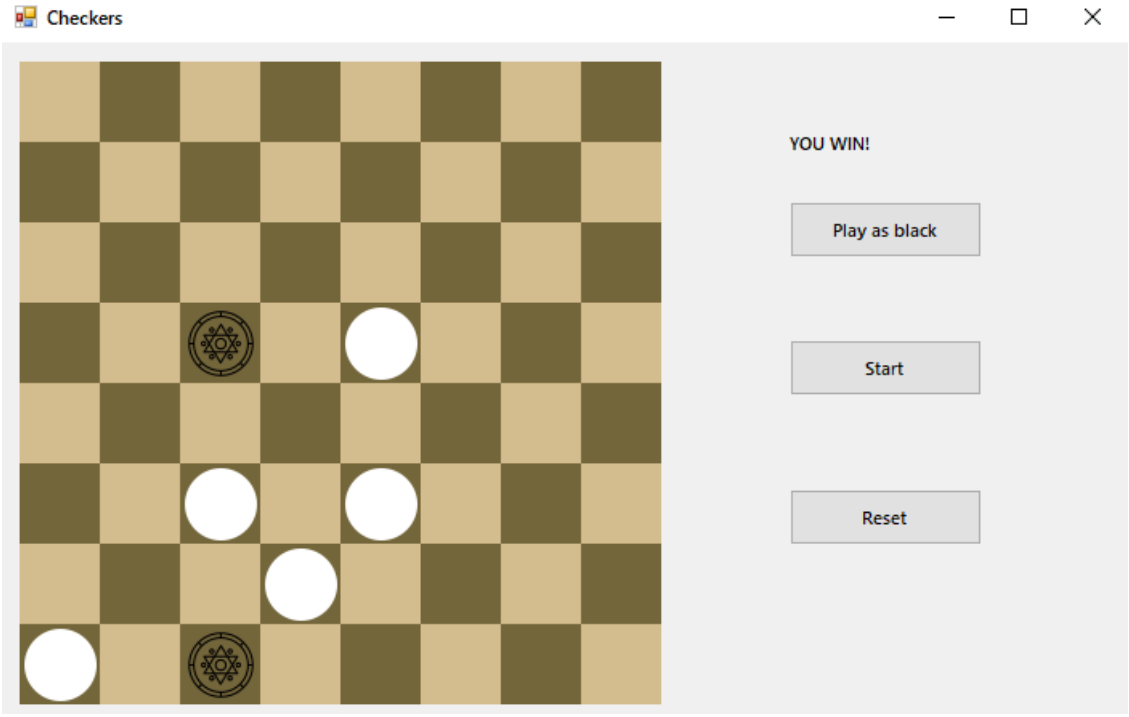

Рисунок 3.7.7 Работа программы

На рисунке 3.7.8 показана форма после нажатия кнопки “Reset” в форме на рисунке 3.7.6:

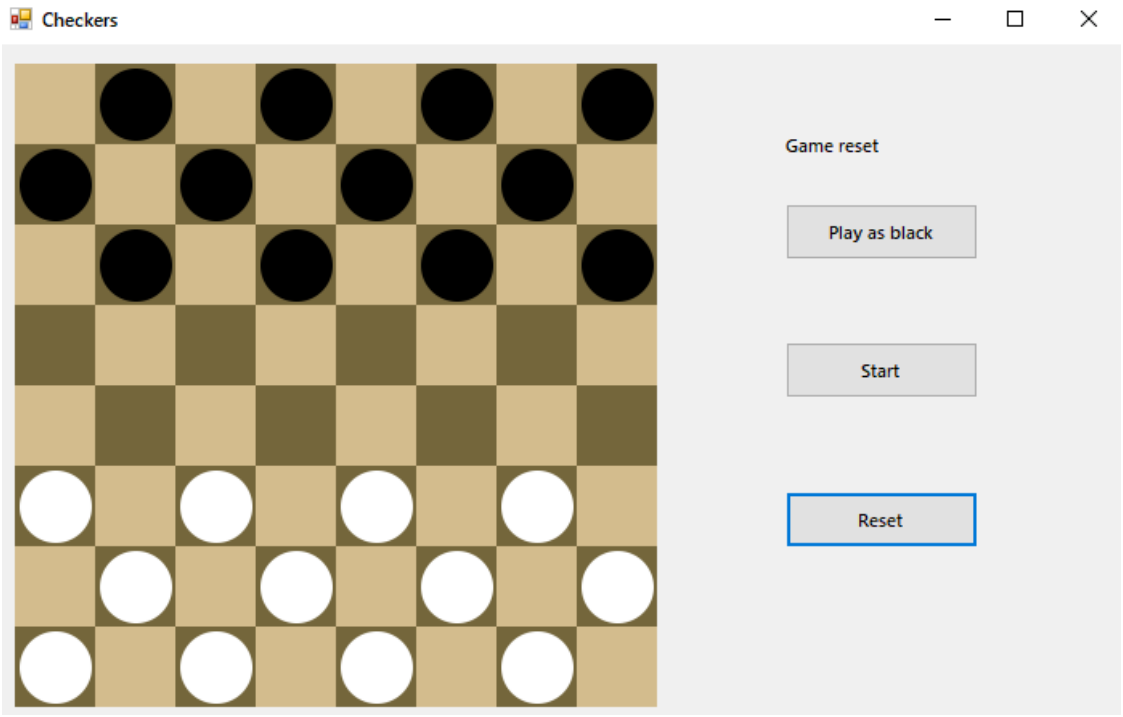

Рисунок 3.7.8 Работа программы

На рисунках 3.7.9-3.7.10 показана небольшая часть вывода протокола игры в текстовый файл:

```
Starting logger <06.06.2023 21:36:43>


New game <06.06.2023 21:36:49>
Human plays white
Bot plays black

player: white, move: g3-f4
player: black, move: d6-e5
player: white, move: f4:d6
player: black, move: e7:c5
player: white, move: e3-d4
player: black, move: c5:e3
player: white, move: d2:f4
player: black, move: b6-c5
player: white, move: f4-e5
```

Рисунок 3.7.9 Работа программы

```
player: black, move: c1-a3
player: white, move: f8-c5
player: black, move: g5-f4
player: white, move: g3:e5
player: black, move: a3-c1
player: white, move: b4-a3
player: black, move: c1-b2
player: white, move: a3:c1
Human wins
```

Рисунок 3.7.10 Работа программы

3.8 Листинг программы

ChessBoard.cs:

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;

public struct Move
{
    public int[] Source;
    public int[] Target;
}


public class ChessBoard
{
    public string Path;
    private Piece[,] boardArray = new Piece[8, 8];
    public Checkers.Color Orientation;
    public Checkers.Color CurrentMoveColor;
    public bool LastMoveTakes = false;

    public ChessBoard()
    {
        Path = Directory.GetParent(Environment.CurrentDirectory).Parent.FullName +
"\\protocol.txt";
    }

    public void Initialize(Checkers.Color orientation)
    {
        CurrentMoveColor = Checkers.Color.White;
        Orientation = orientation;
        for (int y = 0; y < 8; y++)
            for (int x = 0; x < 8; x++)
            {
                if ((x + y) % 2 == 1 && y != 3 && y != 4)
                {
                    Checkers.Color color = Checkers.Color.White;
                    if (y < 3) color = Checkers.Color.Black;
                    boardArray[y, x] = new Checker(color);
                }
                else boardArray[y, x] = null;
            }
    }
```

```
    public Piece this[int x, int y]
    {
        get { return boardArray[x, y]; }
    }

     public List<Move> AvailableMoves()
    {
        List<Move> moves = AvailableCaptures();
        if (moves.Count > 0) return moves;

        Move move;
        moves = new List<Move>();
        for (int y = 0; y < 8; y++)
        {
            for (int x = 0; x < 8; x++)
            {
                if (boardArray[y, x] != null && boardArray[y, x].Color ==
CurrentMoveColor)
                {
                    int[] source = new int[2] { y, x };
                    List<int[]> targets = boardArray[source[0],
source[1]].AvailableMoves(this, source);
                    for (int i = 0; i < targets.Count; i++)
                    {
                        move = new Move() { Source = source, Target = targets[i] };
                        moves.Add(move);
                    }
                }
            }
        }
        return moves;
    }

    private List<Move> AvailableCaptures()
    {
        Move move;
        List<Move> moves = new List<Move>();
        for (int y = 0; y < 8; y++)
        {
            for (int x = 0; x < 8; x++)
            {
                if (boardArray[y, x] != null && boardArray[y, x].Color ==
CurrentMoveColor)
                {
                    int[] source = new int[2] { y, x };
                    List<int[]> targets = boardArray[source[0],
source[1]].AvailableCaptures(this, source);
                    for (int i = 0; i < targets.Count; i++)
                    {
                        move = new Move() { Source = source, Target = targets[i] };
                        moves.Add(move);
                    }
                }
            }
        }
        return moves;
    }

    private bool CanMove(Piece piece, int[] source, int[] target)
    {
        List<Move> moves = AvailableMoves()
            .Where(move => move.Source[0] == source[0] && move.Source[1] ==
source[1])
```

```
            .Where(move => move.Target[0] == target[0] && move.Target[1] ==
target[1])
            .ToList<Move>();

        if (moves.Any()) return true;
        else return false;
    }

    private bool CanTake(Piece piece, int[] source, int[] target)
    {
        List<Move> moves = AvailableCaptures()
            .Where(move => move.Source[0] == source[0] && move.Source[1] ==
source[1])
            .Where(move => move.Target[0] == target[0] && move.Target[1] ==
target[1])
            .ToList<Move>();

        if (moves.Any()) return true;
        else return false;
    }

    private void Take(int[] source, in int[] target)
    {
        int y0 = target[0];
        int x0 = target[1];
        int y_sign = Math.Sign(source[0] - target[0]);
        int x_sign = Math.Sign(source[1] - target[1]);
        for(int i = 0; i < Math.Abs(source[0] - target[0]); i++)
        {
            boardArray[y0 + i * y_sign, x0 + i * x_sign] = null;
        }
    }

    public bool Move(int[] source, int[] target)
    {
        Piece piece = boardArray[source[0], source[1]];
        if (CanMove(piece, source, target))
        {
            LastMoveTakes = false;
            if (CanTake(piece, source, target))
            {
                Take(source, target);
                LastMoveTakes = true;
            }
            boardArray[source[0], source[1]] = null;
            boardArray[target[0], target[1]] = piece;

            if (target[0] == 0 && CurrentMoveColor == Checkers.Color.White)
boardArray[target[0], target[1]] = new King(Checkers.Color.White);
            else if (target[0] == 7 && CurrentMoveColor == Checkers.Color.Black)
boardArray[target[0], target[1]] = new King(Checkers.Color.Black);

            LogMove(source, target);

            return true;
        }
        // invalid move
        return false;
    }

    public void FlipCurrentMoveColor()
    {
        if (CurrentMoveColor == Checkers.Color.White) CurrentMoveColor =
Checkers.Color.Black;
        else CurrentMoveColor = Checkers.Color.White;
```

```
    }

    public void InitLogger()
    {
        using (StreamWriter sw = new StreamWriter(Path, append: false))
        {
            sw.Write("Starting logger <");
            sw.WriteLine($"{DateTime.Now}>");
            sw.WriteLine();
        }
    }

    public void LogInitGame()
    {
        using (StreamWriter sw = new StreamWriter(Path, append: true))
        {
            sw.Write("\nNew game <");
            sw.WriteLine($"{DateTime.Now}>");
            sw.WriteLine("Human plays {0}", (Orientation == Checkers.Color.White) ?
"white" : "black");
            sw.WriteLine("Bot plays {0}", (Orientation == Checkers.Color.Black) ?
"white" : "black");
        }
    }

    private void LogMove(int[] source, int[] target)
    {
        using (StreamWriter sw = new StreamWriter(Path, append: true))
        {
            sw.Write("\nplayer: {0}, move: ", (CurrentMoveColor ==
Checkers.Color.White) ? "white" : "black");
            sw.Write("{0}{1}", "abcdefgh"[source[1]], 8 - source[0]);
            if (LastMoveTakes) sw.Write(':');
            else sw.Write('-');
            sw.Write("{0}{1}", "abcdefgh"[target[1]], 8 - target[0]);
        }
    }

    public void LogGameResults(string result)
    {
        using (StreamWriter sw = new StreamWriter(Path, append: true))
        {
            sw.WriteLine();
            switch (result.ToLower())
            {
                case "win":
                    sw.Write("Human wins");
                    break;
                case "lose":
                    sw.Write("Bot wins");
                    break;
                default:
                    throw new InvalidDataException();
            }
            sw.WriteLine();
        }
    }

}
```

Color.cs:

```
using System;

namespace Checkers
```

```
{
    public enum Color
    {
        White,
        Black
    }
}
```

Bot.cs:

```
using System;
using System.Collections.Generic;

public class Bot
{
    private Random Rand;

    public Bot()
    {
        Rand = new Random(DateTime.Now.Millisecond);
    }


    public Move GenerateMove(ChessBoard chessboard)
    {
        List<Move> moves = chessboard.AvailableMoves();
        return moves[Rand.Next(moves.Count)];
    }
}
```

Piece.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;

public abstract class Piece
{
    public Checkers.Color Color;
    public Image Img { get; set; }
    public abstract List<int[]> AvailableMoves(ChessBoard chessboard, int[] coords);
    public abstract List<int[]> AvailableCaptures(ChessBoard chessboard, int[]
coords);
}
```

Checker.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;

public class Checker : Piece
{
    public Checker(Checkers.Color color)
    {
        Color = color;

        string path =
Directory.GetParent(Environment.CurrentDirectory).Parent.FullName +
"\\images\\checker_";
        if (color == Checkers.Color.White) path += "w.png";
        else path += "b.png";

        Img = Image.FromFile(path);
```

```
    }

    public override List<int[]> AvailableMoves(ChessBoard chessboard, int[] coords)
    {
        List<int[]> moves = new List<int[]>();

        int y = coords[0];
        int x = coords[1];
        if (chessboard.CurrentMoveColor == Checkers.Color.Black && y < 7)
        {
            if (x > 0 && chessboard[y + 1, x - 1] == null) moves.Add(new int[] { y +
1, x - 1 });
            if (x < 7 && chessboard[y + 1, x + 1] == null) moves.Add(new int[] { y +
1, x + 1 });
        }
        else if (chessboard.CurrentMoveColor == Checkers.Color.White && y > 0)
        {
            if (x > 0 && chessboard[y - 1, x - 1] == null) moves.Add(new int[] { y -
1, x - 1 });
            if (x < 7 && chessboard[y - 1, x + 1] == null) moves.Add(new int[] { y -
1, x + 1 });
        }

        moves.AddRange(AvailableCaptures(chessboard, coords));

        return moves;
    }

    public override List<int[]> AvailableCaptures(ChessBoard chessboard, int[]
coords)
    {
        List<int[]> moves = new List<int[]>();
        Piece piece_taken;

        int y = coords[0];
        int x = coords[1];
        if (y < 6)
        {
            if (x > 1)
            {
                piece_taken = chessboard[y + 1, x - 1];
                if (piece_taken != null && piece_taken.Color != Color &&
chessboard[y + 2, x - 2] == null)
                    moves.Add(new int[] { y + 2, x - 2 });
            }
            if (x < 6)
            {
                piece_taken = chessboard[y + 1, x + 1];
                if (piece_taken != null && piece_taken.Color != Color &&
chessboard[y + 2, x + 2] == null)
                    moves.Add(new int[] { y + 2, x + 2 });
            }
        }

        if (y > 1)
        {
            if (x > 1)
            {
                piece_taken = chessboard[y - 1, x - 1];
                if (piece_taken != null && piece_taken.Color != Color &&
chessboard[y - 2, x - 2] == null)
                    moves.Add(new int[] { y - 2, x - 2 });
            }
            if (x < 6)
            {
```

```
                piece_taken = chessboard[y - 1, x + 1];
                if (piece_taken != null && piece_taken.Color != Color &&
chessboard[y - 2, x + 2] == null)
                    moves.Add(new int[] { y - 2, x + 2 });
            }
        }

        return moves;
    }
}
```

King.cs:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;

public class King : Piece
{
    public King(Checkers.Color color)
    {
        Color = color;

        string path =
Directory.GetParent(Environment.CurrentDirectory).Parent.FullName +
"\\images\\king_";
        if (color == Checkers.Color.White) path += "w.png";
        else path += "b.png";

        Img = Image.FromFile(path);
    }

    public override List<int[]> AvailableMoves(ChessBoard chessboard, int[] coords)
    {
        List<int[]> moves = new List<int[]>();

        int y = coords[0];
        int x = coords[1];
        for (int i = 1; i <= Math.Min(y, x); i++)
        {
            if (chessboard[y - i, x - i] == null) moves.Add(new int[2] { y - i, x -
i });
            else break;
        }
        for (int i = 1; i <= Math.Min(7 - y, 7 - x); i++)
        {
            if (chessboard[y + i, x + i] == null) moves.Add(new int[2] { y + i, x +
i });
            else break;
        }
        for (int i = 1; i <= Math.Min(y, 7 - x); i++)
        {
            if (chessboard[y - i, x + i] == null) moves.Add(new int[2] { y - i, x +
i });
            else break;
        }
        for (int i = 1; i <= Math.Min(7 - y, x); i++)
        {
            if (chessboard[y + i, x - i] == null) moves.Add(new int[2] { y + i, x -
i });
            else break;
        }

        moves.AddRange(AvailableCaptures(chessboard, coords));
```

```
        return moves;
    }

    public override List<int[]> AvailableCaptures(ChessBoard chessboard, int[]
coords)
    {
        List<int[]> moves = new List<int[]>();

        int y0 = coords[0];
        int x0 = coords[1];

        int y = y0 - 1;
        int x = x0 - 1;
        while (y > 0 && x > 0)
        {
            if (chessboard[y, x] == null)
            {
                y--; x--;
                continue;
            }
            if (chessboard[y, x].Color == Color) break;

            y--; x--;
            while (y >= 0 && x >= 0 && chessboard[y, x] == null)
            {
                moves.Add(new int[] { y, x });
                y--; x--;
            }

            y--; x--;
        }

        y = y0 + 1;
        x = x0 - 1;
        while (y < 7 && x > 0)
        {
            if (chessboard[y, x] == null)
            {
                y++; x--;
                continue;
            }
            if (chessboard[y, x].Color == Color) break;

            y++; x--;
            while (y <= 7 && x >= 0 && chessboard[y, x] == null)
            {
                moves.Add(new int[] { y, x });
                y++; x--;
            }

            y++; x--;
        }

        y = y0 - 1;
        x = x0 + 1;
        while (y > 0 && x < 7)
        {
            if (chessboard[y, x] == null)
            {
                y--; x++;
                continue;
            }
            if (chessboard[y, x].Color == Color) break;
```

```
            y--; x++;
            while (y >= 0 && x <= 7 && chessboard[y, x] == null)
            {
                moves.Add(new int[] { y, x });
                y--; x++;
            }

            y--; x++;
        }

        x = x0 + 1;
        y = y0 + 1;
        while (y < 7 && x < 7)
        {
            if (chessboard[y, x] == null)
            {
                y++; x++;
                continue;
            }
            if (chessboard[y, x].Color == Color) break;

            y++; x++;
            while (y <= 7 && x <= 7 && chessboard[y, x] == null)
            {
                moves.Add(new int[] { y, x });
                y++; x++;
            }

            y++; x++;
        }

        return moves;
    }
}
```

Form1.cs:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace part_3
{
    public partial class Form1 : Form
    {
        private ChessBoard chessBoard = new ChessBoard();
        private Piece pieceSelected = null;
        private int[] pieceSelectedCoords = new int[2];
        private Bot bot = new Bot();
        private bool gameStarted = false;
        private bool gameEnded = false;

        public Form1()
        {
            InitializeComponent();
            Text = "Checkers";
        }

        private void Form1_Load(object sender, EventArgs e)
```

```
        {
            chessBoard.Initialize(Checkers.Color.White);
            chessBoard.InitLogger();
            for (int x = 0; x < boardLayoutPanel.ColumnCount; x++)
            {
                for (int y = 0; y < boardLayoutPanel.RowCount; y++)
                {
                    Button button = new Button();
                    button.Dock = DockStyle.Fill;
                    button.Margin = new Padding(0);
                    button.FlatStyle = FlatStyle.Flat;
                    button.FlatAppearance.BorderSize = 0;
                    if ((x + y) % 2 == 1) button.BackColor =
System.Drawing.Color.FromArgb(116, 102, 59);  // dark
                    else button.BackColor = System.Drawing.Color.FromArgb(211, 188,
141);                   // light
                    boardLayoutPanel.Controls.Add(button);
                    button.Click += Click_Board;
                }
            }
            DrawPieces();
        }

        private void DrawPieces()
        {
            for (int i = 0; i < 8; i++)
            {
                for (int j = 0; j < 8; j++)
                {
                    int x;
                    int y;
                    if (chessBoard.Orientation == Checkers.Color.White)
                    {
                        y = i;
                        x = j;
                    }
                    else
                    {
                        y = 7 - i;
                        x = 7 - j;
                    }

                    Button button =
(Button)boardLayoutPanel.GetControlFromPosition(x, y);
                    button.FlatStyle = FlatStyle.Flat;
                    if (chessBoard[i, j] != null)
                    {
                        Piece checker = chessBoard[i, j];
                        button.Tag = checker;

                        Size size = button.Size;
                        size.Width -= size.Width / 10;
                        size.Height -= size.Height / 10;
                        button.Image = (Image)(new Bitmap(checker.Img, size));
                        button.ImageAlign = ContentAlignment.MiddleCenter;
                    }
                    else
                    {
                        button.Image = null;
                        button.Tag = null;
                    }
                }
            }
        }
```

```
        private void ResetBoardButtons(ChessBoard board)
        {
            for (int i = 0; i < 8; i++)
                for (int j = 0; j < 8; j++)
                {
                    int x;
                    int y;
                    if (chessBoard.Orientation == Checkers.Color.White)
                    {
                        y = i;
                        x = j;
                    }
                    else
                    {
                        y = 7 - i;
                        x = 7 - j;
                    }

                    Button button =
(Button)boardLayoutPanel.GetControlFromPosition(x, y);
                    button.FlatStyle = FlatStyle.Flat;
                }
        }

        private void Click_Board(object s, EventArgs e)
        {
            if (!gameStarted || gameEnded) return;

            ResetBoardButtons(chessBoard);
            if (!(s is Button)) return;
            Button button = (Button)s;
            TableLayoutPanelCellPosition pos =
boardLayoutPanel.GetPositionFromControl((Control)s);
            if (chessBoard.Orientation == Checkers.Color.Black)
            {
                pos.Row = 7 - pos.Row;
                pos.Column = 7 - pos.Column;
            }

            if (!(button.Tag is Piece)) // empty square
            {
                if (pieceSelected != null)
                {
                    // move
                    int[] target = new int[] { pos.Row, pos.Column };
                    bool move_is_valid = chessBoard.Move(pieceSelectedCoords,
target);
                    if (move_is_valid)
                    {
                        // capture chain
                        List<int[]> captures_available =
pieceSelected.AvailableCaptures(chessBoard, target);
                        if (chessBoard.LastMoveTakes && captures_available.Any())
                        {
                            DrawPieces();
                            pieceSelected = chessBoard[target[0], target[1]];
                            pieceSelectedCoords = target;
                            button.FlatStyle = FlatStyle.Standard;

                            foreach (int[] capture in captures_available)
HighlightSquare(capture);
                            return;
                        }
```

```
                        chessBoard.FlipCurrentMoveColor();
                        List<Move> moves = chessBoard.AvailableMoves();
                        if (moves.Count == 0)
                        {
                            WinGame();
                            return;
                        }
                        DrawPieces();

                        MakeBotMove();
                        chessBoard.FlipCurrentMoveColor();
                        moves = chessBoard.AvailableMoves();
                        if (moves.Count == 0)
                        {
                            LoseGame();
                            return;
                        }
                    }
                    DrawPieces();
                    pieceSelected = null;
                }
                return;
            }

            Piece piece = (Piece)button.Tag;

            if (pieceSelected != null && pieceSelected.Color != piece.Color)  //
piece is already selected and colors are different
            {
                pieceSelected = null;
            }
            else if (piece.Color == chessBoard.Orientation)
            {
                pieceSelected = piece;
                pieceSelectedCoords[0] = pos.Row;
                pieceSelectedCoords[1] = pos.Column;

                List<Move> moves = chessBoard.AvailableMoves()
                    .Where(move => move.Source[0] == pieceSelectedCoords[0] &&
move.Source[1] == pieceSelectedCoords[1])
                    .ToList<Move>();

                if (moves.Any()) button.FlatStyle = FlatStyle.Standard;
                foreach (Move move in moves) HighlightSquare(move.Target);
            }
        }

        private void HighlightSquare(int[] sq)
        {
            int y = (chessBoard.Orientation == Checkers.Color.White) ? sq[0] : 7 -
sq[0];
            int x = (chessBoard.Orientation == Checkers.Color.White) ? sq[1] : 7 -
sq[1];
            Button actionButton = (Button)boardLayoutPanel.GetControlFromPosition(x,
y);
            actionButton.FlatStyle = FlatStyle.Standard;
        }

        public void MakeBotMove(bool wait = true)
        {
            // bot move
            Move move = bot.GenerateMove(chessBoard);
            chessBoard.Move(move.Source, move.Target);
            if (wait) Wait(500);
            DrawPieces();
```

```
            // capture chain
            List<int[]> captures_available = chessBoard[move.Target[0],
move.Target[1]].AvailableCaptures(chessBoard, move.Target);
            while (chessBoard.LastMoveTakes && captures_available.Any())
            {
                move = bot.GenerateMove(chessBoard);
                chessBoard.Move(move.Source, move.Target);
                Wait(300);
                DrawPieces();
                captures_available = chessBoard[move.Target[0],
move.Target[1]].AvailableCaptures(chessBoard, move.Target);
            }
        }

        private void WinGame()
        {
            gameEnded = true;
            label1.Text = "YOU WIN!";
            DrawPieces();
            chessBoard.LogGameResults("win");
        }

        private void LoseGame()
        {
            gameEnded = true;
            label1.Text = "YOU LOSE!";
            DrawPieces();
            chessBoard.LogGameResults("lose");
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // change your color
            if (gameStarted) return;
            Checkers.Color orientation = (chessBoard.Orientation ==
Checkers.Color.White) ? Checkers.Color.Black : Checkers.Color.White;
            chessBoard.Initialize(orientation);
            DrawPieces();

            button1.Text = "Play as ";
            button1.Text += (chessBoard.Orientation == Checkers.Color.Black) ?
"white" : "black";

            label1.Text = "You play ";
            label1.Text += (chessBoard.Orientation == Checkers.Color.Black) ?
"black" : "white";
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // start game
            if (gameStarted) return;
            gameStarted = true;
            label1.Text = "Game started";
            chessBoard.LogInitGame();
            if (chessBoard.Orientation == Checkers.Color.Black)
            {
                MakeBotMove(false);
                chessBoard.FlipCurrentMoveColor();
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
```

```
            // reset
            chessBoard = new ChessBoard();
            chessBoard.Initialize(Checkers.Color.White);
            gameStarted = false;
            gameEnded = false;
            pieceSelected = null;
            DrawPieces();
            ResetFormControls();
            label1.Text = "Game reset";
        }

        private void ResetFormControls()
        {
            button1.Text = "Play as black";
            label1.Text = "";
        }

        public void Wait(int milliseconds)
        {
            var timer = new System.Windows.Forms.Timer();
            if (milliseconds == 0 || milliseconds < 0) return;

            timer.Interval = milliseconds;
            timer.Enabled = true;
            timer.Start();

            timer.Tick += (s, e) =>
            {
                timer.Enabled = false;
                timer.Stop();
            };

            while (timer.Enabled)
            {
                Application.DoEvents();
            }
        }
    }
}
```

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были разработаны 3 программы, основанные на принципах объектно-ориентированного подхода. Были закреплены навыки работы с файлами, WinForms, интерфейсами, абстрактными классами. Использовались принципы инкапсуляции, наследования, полиморфизма, агрегации. Получен опыт в разработке приложений, моделирующих непостоянные процессы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Поздняков, Рыбин – Дискретная математика (дата обращения: 01.06.2023)
2. Moodle – источник заданий URL: vec.etu.ru/moodle (дата обращения 10.05.2023)
3. Документация по языку C# URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 11.05.2023)
4. Статья с информацией о UML-диаграммах классах URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/
5. Документация по UML-диаграммам URL: https://www.uml-diagrams.org/