**Форма № Н-6.01у**

Кременчуцький національний університет імені Михайла Остроградського

(повне найменування вищого навчального закладу)

Кафедра автоматизації та інформаційних систем

(повна назва кафедри, циклової комісії)

**КУРСОВИЙ ПРОЄКТ**

**(РОБОТА)**

з дисципліни «Сучасні мови об’єктно-орієнтованого програмування»

(назва дисципліни)

на тему Розробка WPF додатку “Memory”

Студента \_2\_ курсу КН–22–1 групи

Ступінь вищої освіти «Бакалавр»

(бакалавр, магістр)

Спеціальність 122 – «Комп’ютерні науки»

Освітньо-професійна програма

«Комп’ютерні науки»

Радіонов Д.О.

(прізвище та ініціали)

Керівник старший викладач кафедри АІС

\_\_\_\_\_\_\_ Бельська В. Ю.\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_. Оцінка: ЄКTС \_\_\_\_

Члени комісії  . В. Ю. Бельська

(підпис) (ініціали та прізвище)

І. Г. Оксанич

(підпис) (ініціали та прізвище)

В. В. Найда

(підпис) (ініціали та прізвище)

м. Кременчук 2023 рік

**Форма № Н-9.01у**

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра автоматизації та інформаційних систем

Дисципліна «Сучасні мови об’єктно-орієнтованого програмувння»

Освітній ступінь \_\_ «Бакалавр»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Спеціальність 122 – «Комп’ютерні науки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Освітня програма «Комп’ютерні науки»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс 2 група \_\_КН-22-1 семестр \_\_3\_\_

**ЗАВДАННЯ**

**НА КУРСОВИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Радіонову Дмитру Олеговичу \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_

(прізвище, ім’я, по-батькові)

1. Тема роботи: Розробка WPF додатку “Memory”
2. Термін здачі студентом роботи 1 грудня 2023 р
3. Вихідні дані до роботи:
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

постановка завдання, аналіз технічного завдання, розробка алгоритму роботи програми, опис моделі даних, структура програмного забезпечення, функціональна схема та інтерфейс програми.

1. Перелік графічного матеріалу:
2. Дата видачі завдання: 1 жовтня 2023 р.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| №  пор. | Назва етапів курсового проекту | Терміни виконання етапів проекту | Вказівки та  зауваження викладача (з зазначенням дати консультації) | Оцінювання етапів проекту | | |
| за націо- нальною шкалою | за шкалою ЄКТС | кількість балів |
| 1 | Етап 1  Аналіз предметної області | 01.10.23–  25.10.23 |  |  | | |
| 2 | Етап 2  Створення моделі даних | 16.10.23–  28.10.23 |  |  | | |
| 3 | Етап 3  Розробка віконного інтерфейсу та створення основного програмного коду додатку | 29.10.23–  24.11.23 |  |  | | |
| 4 | Етап 4  Тестування програмного коду | 16.10.23  29.11.23 |  |  | | |
| 5 | Етап 5  Оформлення пояснювальної записки | 25.11.23–  31.11 .23 |  |  | | |
| 6 | Етап 9  Захист | 01.12.23 |  |  | | |
|  | Разом | 8 тижнів |  |  | | |

Студент Радіонов Дмитро Олегович

(підпис)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_ В. Ю. Бельська \_\_\_\_

(підпис) (ініціали та прізвище)

«1» жовтня 2023 р.

РЕФЕРАТ

Курсова робота містить 29 сторінок, 2 розділи, 10 рисунків, 1 таблиця, 5 використаних джерел.

Об’єкт розробки – WPF додаток «Memory».

Мета: створення віконного Wpf додатку, що реалізує гру «Memory».

Під час виконання завдання, поставленого на курсову роботу, було виконано аналіз алгоритму гри «Memory». Визначено основні функціональні та не функціональні вимоги до додатку, що створюється. Побудована модель даних, що дозволить вирішити наступні задачі: створення віконного інтерфейсу задля графічного відображення етапів гри тощо.

В якості мови програмування для створення Wpf додатку використовувалася мова с# та середовище програмування Visual Studio 22

Результатом виконання всіх етапів є застосунок, який може проводити гру,  
можна вибрати будь-яку букву і всі аналогічні букви будуть відкриті

ОБ’ЄКТ, ІНКАПСУЛЯЦІЯ, ІНТЕРФЕЙС, КЛАС

**Зміст**

[Вступ 2](#_Toc151681430)

[1 Опис предметної областІ 3](#_Toc151681431)

[1.1 Делегати. Стандартні делегати. Для чого використовуються. 3](#_Toc151681432)

[1.2 Аналіз технічного завдання на роботу 4](#_Toc151681433)

[1.2.1 Функціональні вимоги 4](#_Toc151681434)

[1.2.2 Нефункціональні вимоги 4](#_Toc151681435)

[1.3 Опис алгоритму основних задач/підзадач у роботі 5](#_Toc151681436)

[1.3.1 Структура програмного забезпечення 5](#_Toc151681437)

[Висновки до розділу 5](#_Toc151681438)

[2 Опис розробки програмного забезпечення 5](#_Toc151681439)

[2.1 Структура програмного забезпечення 5](#_Toc151681440)

[2.2 Опис роботи програми 7](#_Toc151681441)

[2.3 Функціональна схема програми 8](#_Toc151681442)

[2.4 Опис інтерфейсу програми 8](#_Toc151681443)

[Висновки до розділу 9](#_Toc151681444)

[ВИСНОВКИ 10](#_Toc151681445)

[СПИСОК ЛІТЕРАТУРИ 11](#_Toc151681446)

[Додаток А XAML розмітки вікон 12](#_Toc151681447)

[Додаток Б C# коди класів вікон 15](#_Toc151681448)

[Додаток B C# основні класи 22](#_Toc151681449)

[Додаток Г Тестування додатку 24](#_Toc151681450)

ВСТУП

Технологія WPF (Windows Presentation Foundation) є частиною екосистеми платформи .NET і являє собою підсистему для побудови графічних інтерфейсів.

WPF є дуже поширеною платформою для розробки програмного забезпечення із графічним інтерфейсом на мові програмування C#.

Якщо при створені традиційних додатків на основі WinForms за візуалізацію елементів керування відповідали такі частини ОС Windows, як User32 та GDI+, то додатки WPF засновані на **DirectX**.В цьому і є головна відмінність рендерингу графіки у WPF: велика частина роботи по візуалізації графіки, як і звичайних кнопок, так і складних 3D-моделей, лежить на графічному процесорі на відеокарті, що також дає можливість користуватись апаратним прискоренням графіки.

Ще однією важливою особливістю WPF є використання мови декларативної розмітки XAML, заснованого на XML: ви можете створити гарний графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керуючих мовах, як C#.

Платформа WPF підтримує багато можливостей для створення різних додатків: створення анімацій, binding, стилі, шаблони, велика кількість елементів.

Тому WPF – непоганий варіант для створення ігри «Memory», де за допомогою анімацій, ресурсів, стилів та біндингу можна створити гарний та зрозумілий інтерфейс. А за допомогою вбудованих класів та інтерфейсів зручно робиться логіка гри.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Делегати. Стандартні делегати. Для чого використовуються.

Делегати є однією з ключових концепцій мови програмування C#. Вони дозволяють передавати метод як параметр в інший метод, що надає багато корисних можливостей.

Основне використання делегатів полягає в реалізації механізму зворотного виклику (callback). Вони дозволяють передавати виклик методу з одного об'єкта в інший, забезпечуючи зв'язок між цими об'єктами. Таким чином, вони дозволяють будь-якій частині програми визвати метод, який визначений в іншій частині програми.

Основні переваги використання делегатів:

1. Розривання купи залежностей: Використання делегатів дозволяє скласти код таким чином, що об'єкт, який виконує виклик методу, не залежить від конкретного об'єкта, на якому має бути викликано цей метод. Це сприяє зменшенню залежностей між об'єктами і полегшує тестування і розподілену роботу.

2. Зростання перевикористовування коду: Делегати дозволяють використовувати методи з об'єктів різних класів, сприяючи перевикористанню коду. Наприклад, ви можете передати делегату вкладений метод з одного класу і викликати його з іншого класу.

3. Підтримка асинхронного програмування: Делегати є ключовими компонентами для реалізації асинхронного програмування в C#. Вони дозволяють визначити методи зворотного виклику, які можна викликати після завершення виконання асинхронної операції.

4. Реалізація подій: Делегати використовуються для реалізації подій в C#. Події дозволяють класу або об'єкту повідомити інші класи або об'єкти про виникнення певної події.

Оглядаючи всі ці використання делегатів, можно сказати, що вони є потужним інструментом для досягнення гнучкості і розширення функціональності коду.

1.2 Аналіз технічного завдання на роботу

Мета курсового проєкту – створення віконного Wpf додатку, що реалізує гру «Memory».

Під час розробки моделей даних та графічного інтерфейсу були висунуті наступні вимоги:

**1.2.1 Функціональні вимоги**

1. Гравець повинен мати змогу обрати дві карти на кожен хід.
2. Якщо обрані карти співпадають, вони повинні залишатися обличчям вгору, а гравець отримує очки.
3. Якщо обрані карти не співпадають, вони повинні повернутися обличчям донизу.
4. Гра повинна тривати до тих пір, поки всі пари карт не будуть відгадані.
5. Гра повинна мати можливість перезапустити гру або почати нову гру, коли гравець закінчив.

**1.2.2 Нефункціональні вимоги**

1. Зовнішній інтерфейс користувача має бути реалізованим за допомогою створення вікон в середовищі WPF додатку.
2. Додаток має бути віконним
3. Усі поля введення повинні бути захищені від некоректного введення.
4. Використання патерну MVVM тa прив’язки даних .

1.3 Опис алгоритму основних задач/підзадач у роботі

Розробка алгоритмів функціонування програми в контексті гри “Memory” у середовищі C# WPF передбачає створення моделі роботи та функціональності програми як єдиної системи. На даному етапі необхідно передбачити обмеження, які можуть у майбутньому вплинути на коректне розширення функціоналу програми.

Програма розробляється на мові програмування C# з використанням технології WPF. Основна мета програми – відображення гри “Memory”.

Основні кроки роботи програми:

* Користувачу відображається головний екран програми в якому є закриті картинки на які можна натиснути.
* При відкриванні двох однакових картинок гравцю нараховується очко.
* У разі якщо гравець помиляється, то картинки перевертаються знову і хід переходить іншому гравцю
* Коли всі картинки відкриті гра закінчується і перемагає гравець в якого найбільше очок.

Висновки до розділу

У першому розділі розглянуті базові принципи ООП, створена розширена постановка задачі, проаналізована предметна область, сформовано вимоги до функціональних можливостей програми та розглянуто технічне завдання. Складений план розробки програми та функції, що повинні бути реалізовані у програмі. Також, розроблений алгоритм роботи програми.

2 ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Структура програмного забезпечення

Структура програмного забезпечення є поєднанням усіх застосованих модулів програми, що утворюють працюючу систему. Кожний модуль повинен реалізовувати функції програми, що вирішують основні завдання проєкту.

Застосунок створений за допомою модулю Windows Presentation Foundation (WPF) на базі мови с#.

На рис. 2.1 зображено оглядач рішень програмного застосунку. В табл.2.1 представлено опис модулів(класів) проекту курсової роботи

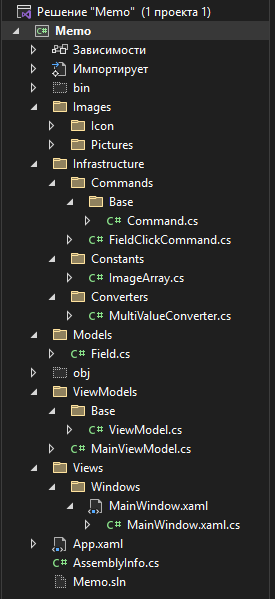


Рисунок 2.1 – Оглядач рішень

Таблиця 2.1 – Призначення модулів програми

|  |  |
| --- | --- |
| **Назва модуля** | **Призначення модуля** |
| Command.cs | Клас, який містить базовий функціонал для створення команд |
| FieldClickCommands.cs | Є командою зв’язаною з натисканням на картинку у грі |
| ImageArray.cs | Клас, який містить усі картинки що використовуються у грі |
| MultiValueConverter.cs | Дозволяє перетворити декілька значень в одне |
|  |  |
| ViewModel.cs | Клас який служить для зв’язку між візуальним відображенням інтерфейса та моделю даних |
| Field.cs | Клас який реалізує можливість взаємодії між полями. |
|  |  |
|  |  |
| MainWindow.xaml | Є файлом розмітки, який описує вигляд та розташування елементів на головному вікні програми. |
| MainWindow.xaml.cs | Файл який відповідає за логіку та функціонування головного вікна в програмі |

2.2 Опис роботи програми

Після запуску програми відкривається головне вікно, в якому відображаються закриті поля на які можна натиснути. При натисканні на поле відображається картинка. Якщо дві картинки співпадають то гравцю нараховується очко, якщо не співпадають то поля перевертаються знову та хід переходить іншому гравцю. Ціль гри перегорнути усі поля. Перемагає той гравець у якого найбілше очок .

**2.3 Функціональна схема програми**

Детальний опис й пояснення процесів окремих функцій розробляємої системи утворюють функціональну схему програми(рис. 2.4).

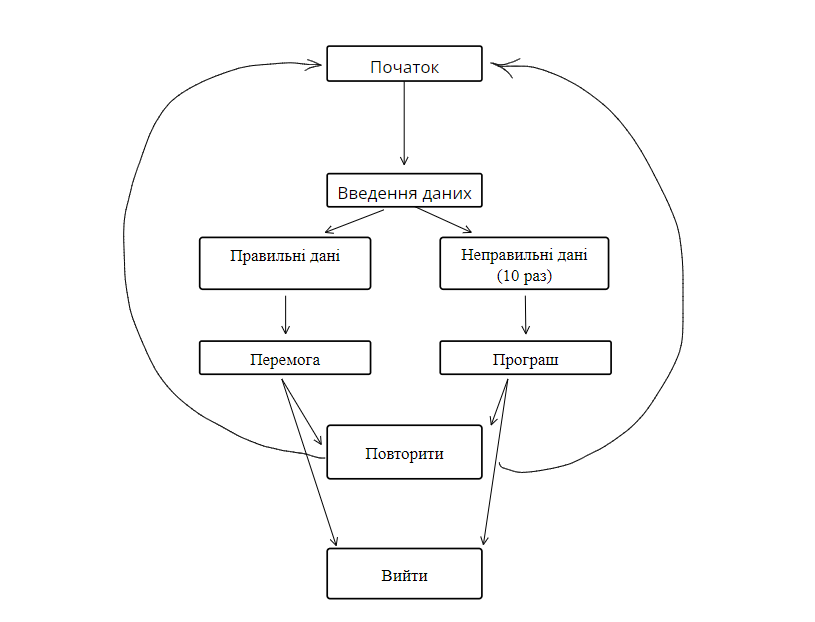


Рисунок 2.4 – Функціональна схема програмного коду

Як видно з рис.2.4 після запуску за стосунку першим відображається вікно з алфавітом і загаданим словом.

2.4 Опис інтерфейсу програми

При розробці програми створено зручний інтерфейс користувача. Опишемо основні вікна програми:

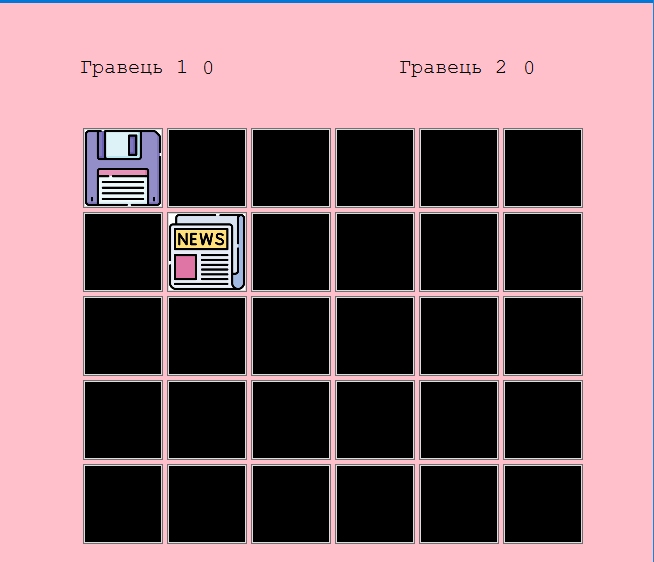


Рисунок 2.5 – Інтерфейс вікна MainWindow

Висновки до розділу

У другому розділі розглянуті: ієрархія класів, ініціалізація даних, функціональна схема програми та опис основних функціональних вікон програми, їх призначення та властивості.

Крім того, розроблено інтерфейс програми і детально описано керівництво користувача системи.

ВИСНОВКИ

Курсовий проєкт спрямований на створення WPF додатку «Memory». Розглянуті базові принципи ООП, проаналізована предметна область та розглянуто технічне завдання, реалізовано можливість реєстрації та авторизації, можливість створити платіж за книгу та оплатити його, можливість переглянути свої платежі.

Для виконання даного проєкту була використана мова програмування C#, яка добре підходить для розробки програм з використанням принципів ООП.

Важливим етапом розробки стали планування й реалізація користувальницького інтерфейсу. Основною метою було задоволення ряду потреб: комфортність роботи з інтерфейсом, швидкодія при обміні даних, ефективність управління, реалізація необхідних функцій. Використання приємних для більшості людей кольорів, а також захист від усіляких способів некоректного введення.

Таким чином, у курсового проєкту продемонстровані найважливіші методи роботи з програмним середовищем WPF, його інструментами та функціями.

СПИСОК ЛІТЕРАТУРИ

1. Полное руководство по языку программирования С# 11 и платформе .NET 7. URL: <https://metanit.com/sharp/tutorial/>
2. Джейсон Шрейкер. Кровь пот и пиксели. Форс, 2019. 368 с.
3. Тайнан Сильвестр. Геймдизайн. Санкт-Петербург: Питер, 2021. 448 с.
4. Вайсфельд М. Об’єктно-ориентированное мышление. Санкт-Петербург: Питер, 2014. 304 с.
5. Уроки створення додатку Windows Forms C#. URL: <https://itproger.com/course/csharp-app>

Додаток А  
XAML розмітки вікон

**Файл MainWindow.xaml**

<Window x:Class="Memo.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Memo"

xmlns:com="clr-namespace:Memo.Infrastructure.Commands"

xmlns:vm="clr-namespace:Memo.ViewModels"

xmlns:con="clr-namespace:Memo.Infrastructure.Converters"

Icon="../../Images/Icon/icon.png"

mc:Ignorable="d"

Title="MEMORY" Height="600" Width="670" Background="Pink" ResizeMode="NoResize" WindowStartupLocation="CenterScreen">

<Window.Resources>

<vm:MainViewModel x:Key="MainViewModel"/>

<com:FieldClickCommand x:Key="FieldClickCommand"/>

<con:MultiValueConverter x:Key="MultiValueConverter"/>

<DataTemplate x:Key="OneButton">

<Button Command ="{StaticResource FieldClickCommand}" Width="80" Height="80" Margin="2" IsEnabled="{Binding Enable}">

<Button.CommandParameter>

<MultiBinding Converter="{StaticResource MultiValueConverter}">

<Binding/>

<Binding Source="{StaticResource MainViewModel}"/>

</MultiBinding>

</Button.CommandParameter>

<Image Source="{Binding ImagePath}"/>

</Button>

</DataTemplate>

<DataTemplate x:Key="OneRow">

<ItemsControl ItemsSource="{Binding}" ItemTemplate="{DynamicResource OneButton}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<StackPanel Orientation="Horizontal"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

</ItemsControl>

</DataTemplate>

</Window.Resources>

<Grid DataContext="{StaticResource MainViewModel}" Margin="5,0,3,-16">

<Grid.RowDefinitions>

<RowDefinition Height="551\*"/>

<RowDefinition Height="49\*"/>

</Grid.RowDefinitions>

<ItemsControl ItemTemplate="{DynamicResource OneRow}" ItemsSource="{Binding A}" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="77,123,0,0" Height="442" Width="505" Grid.RowSpan="2"></ItemsControl>

<TextBlock HorizontalAlignment="Left" Margin="76, 53,0 ,0" VerticalAlignment="Top" Foreground="Black">

<Run Text="Гравець 1" FontSize="20" FontFamily="Courier New" />

<LineBreak/>

</TextBlock>

<TextBlock HorizontalAlignment="Left" Margin="395, 53, 0, 0" VerticalAlignment="Top" Foreground="Black" Height="36" >

<Run Text="Гравець 2" FontSize="20" FontFamily="Courier New" />

</TextBlock>

<Label Content="{Binding PointPlayer1}" Foreground="Black" HorizontalAlignment="Left" Margin="193,49,0,0" VerticalAlignment="Top" FontSize="20" FontFamily="Courier New"/>

<Label Content="{Binding PointPlayer2}" Foreground="Black" HorizontalAlignment="Left" Margin="514,49,0,0" VerticalAlignment="Top" FontSize="20" FontFamily="Courier New"/>

</Grid>

</Window>

Додаток Б  
C# коди класів вікон

**Клас Command.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Input;

namespace Memo.Infrastructure.Commands.Base

{

public abstract class Command : ICommand

{

public event EventHandler CanExecuteChanged

{

add => CommandManager.RequerySuggested += value;

remove => CommandManager.RequerySuggested -= value;

}

public bool CanExecute(object parameter) => true;

public abstract void Execute(object parameter);

}

}

**Клас FieldClickCommand.cs**

using Memo.ViewModels;

using Memo.Models;

using Memo.Infrastructure.Commands.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memo.Infrastructure.Commands

{

public class FieldClickCommand : Command

{

public override void Execute(object parameters)

{

object[] param = parameters as object[];

Field field = param[0] as Field;

MainViewModel model = param[1] as MainViewModel;

model.ClickField(field);

}

}

}

**Клас MultiValueConverter.cs**

using System;

using System.Collections.Generic;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Data;

namespace Memo.Infrastructure.Converters

{

public class MultiValueConverter : IMultiValueConverter

{

public object Convert(object[] values, Type targetType, object parameter, CultureInfo culture)

{

return values.Clone();

}

public object[] ConvertBack(object value, Type[] targetType, object paremeter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

**Клас ImageArray.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memo.Infrastructure.Constants

{

class ImageArray

{

public static string ImageFolder = "../../Images/Pictures/";

public static string Black = ImageFolder + "Black.png";

public static string[] images = new string[]

{

ImageFolder + "camera.png",

ImageFolder + "cpu.png",

ImageFolder + "dom.png",

ImageFolder + "disketka.png",

ImageFolder + "gazeta.png",

ImageFolder + "clava.png",

ImageFolder + "instr.png",

ImageFolder + "lampochka.png",

ImageFolder + "map.png",

ImageFolder + "money.png",

ImageFolder + "pechenie.png",

ImageFolder + "pesok-chasi.png",

ImageFolder + "pirojin.png",

ImageFolder + "sova.png",

ImageFolder + "zamok.png",

};

}

}

**Клас Field.cs**

using Memo.Infrastructure.Constants;

using Memo.ViewModels.Base;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Memo.Models

{

internal class Field : ViewModel

{

private int i;

public int I { get => i; set => Set(ref i, value); }

private int j;

public int J { get => j; set => Set(ref j, value); }

public string image = ImageArray.Black;

private string imagePath = ImageArray.Black;

public string ImagePath { get => imagePath; set => Set(ref imagePath, value); }

private bool enable = true;

public bool Enable { get => enable; set => Set(ref enable, value); }

private bool selected = false;

public bool Selected { get => selected;set => Set(ref selected, value); }

}

}

**КласViewModel.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace Memo.ViewModels.Base

{

public abstract class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

protected virtual bool Set<T>(ref T field, T value, [CallerMemberName] string propertyName = null)

{

if (Equals(field,value))

return false;

field = value;

OnPropertyChanged(propertyName);

return true;

}

}

}

**Файл MainViewModel.cs**

using Memo.Infrastructure.Constants;

using Memo.Models;

using Memo.ViewModels.Base;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.DirectoryServices.ActiveDirectory;

using System.IO.IsolatedStorage;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace Memo.ViewModels

{

internal class MainViewModel : ViewModel

{

public bool StepFirst = true;

public Field SelectedField = null;

private ObservableCollection<ObservableCollection<Field>> a = new();

public ObservableCollection<ObservableCollection<Field>> A { get => a; set => Set(ref a, value); }

private int pointPlayer1 = 0;

public int PointPlayer1 { get => pointPlayer1; set => Set(ref pointPlayer1, value); }

private int pointPlayer2 = 0;

public int PointPlayer2 { get => pointPlayer2; set => Set(ref pointPlayer2, value); }

readonly Random rnd = new();

public void StartGame()

{

StepFirst = true;

GenerateField();

GenerateImage();

PointPlayer1 = 0;

PointPlayer2 = 0;

}

public MainViewModel()

{

StartGame();

}

public void RestartGame()

{

MessageBoxResult results = MessageBox.Show("Розпочати знову", "Кінець гри", MessageBoxButton.YesNo);

if (results == MessageBoxResult.Yes)

{

StartGame();

}

else

{

Application.Current.Shutdown();

}

}

public void GenerateField()

{

a.Clear();

for (int i = 0; i < 5;i++)

{

ObservableCollection<Field> row = new();

for (int j = 0; j < 6; j++)

{

row.Add(new Field() { I = i, J = j });

}

a.Add(row);

}

}

public void GenerateImage()

{

for (int i = 0; i < ImageArray.images.Length; i++)

{

int added = 0;

for (int j = 0; j < 2 + added; j++)

{

int w = rnd.Next(0, 5);

int h = rnd.Next(0, 6);

if (A[w][h].image != ImageArray.Black)

{

added++;

continue;

}

A[w][h].image = ImageArray.images[i];

}

}

}

public bool FindOpenedField()

{

List<Field> opened = new();

for (int i = 0; i < 5;i++)

{

for (int j =0; j < 6; j++)

{

if (A[i][j].image == A[i][j].ImagePath && A[i][j].Enable)

{

opened.Add(A[i][j]);

}

}

}

if (opened.Count == 2)

{

opened[0].ImagePath = ImageArray.Black;

opened[1].ImagePath = ImageArray.Black;

return true;

}

return false;

}

public void ClickField(Field field)

{

if (FindOpenedField()) return;

if (SelectedField == null)

{

SelectedField = field;

SelectedField.ImagePath = SelectedField.image;

}

else

{

if (field == SelectedField)

{

return;

}

if (SelectedField.ImagePath == field.image)

{

field.ImagePath = field.image;

SelectedField.Enable = false;

field.Enable = false;

if (StepFirst)

{

PointPlayer1 += 1;

}

else

{

PointPlayer2 += 1;

}

}

else

{

field.ImagePath = field.image;

StepFirst = !StepFirst;

}

SelectedField = null;

CheckPoint();

}

}

public void CheckPoint()

{

int count = 0;

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 6; j++)

{

if (!A[i][j].Enable)

{

count++;

}

}

}

if (count == 30)

{

if (pointPlayer1 > pointPlayer2)

{

MessageBox.Show("Перший гравець виграв!");

RestartGame();

}

else if (pointPlayer1 < pointPlayer2)

{

MessageBox.Show("Другий гравець виграв!");

RestartGame();

}

}

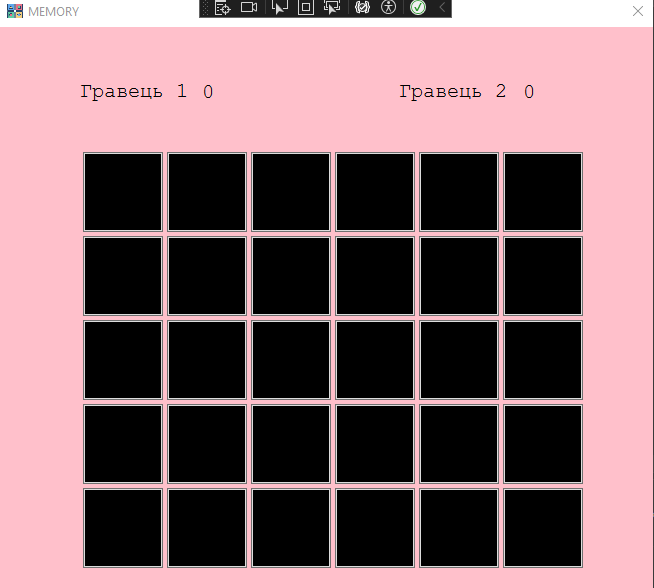
}

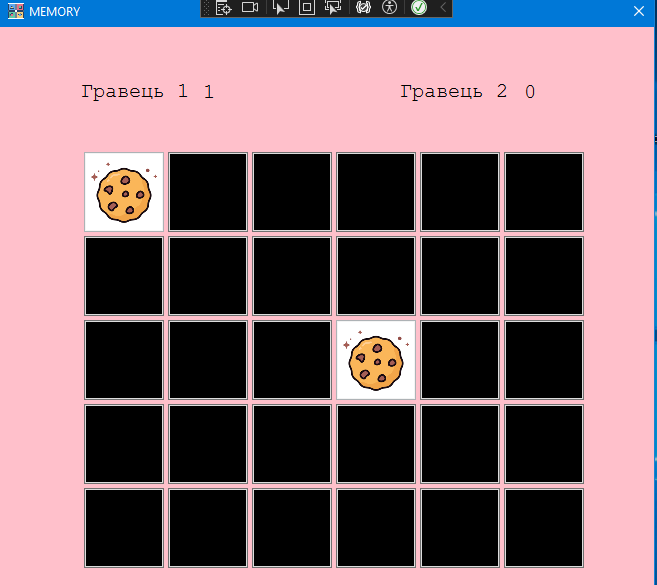
}

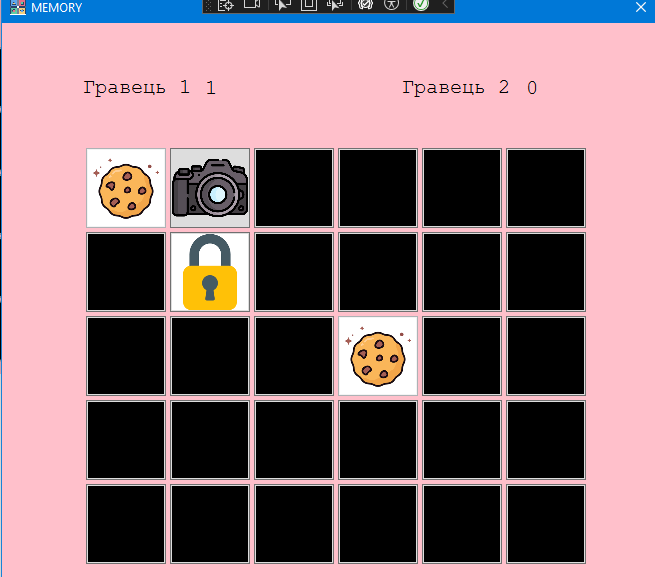
}

Додаток B  
C# основні класи

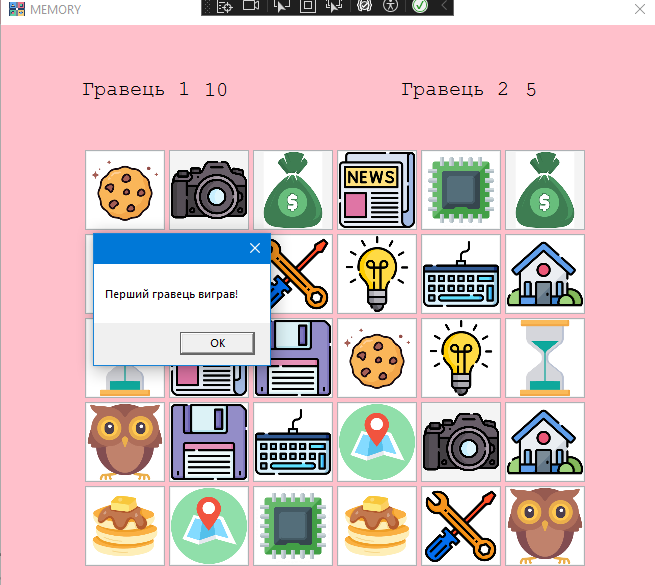
Додаток Г  
Тестування додатку

 **Рисунок Г.1 – початкове вікно**

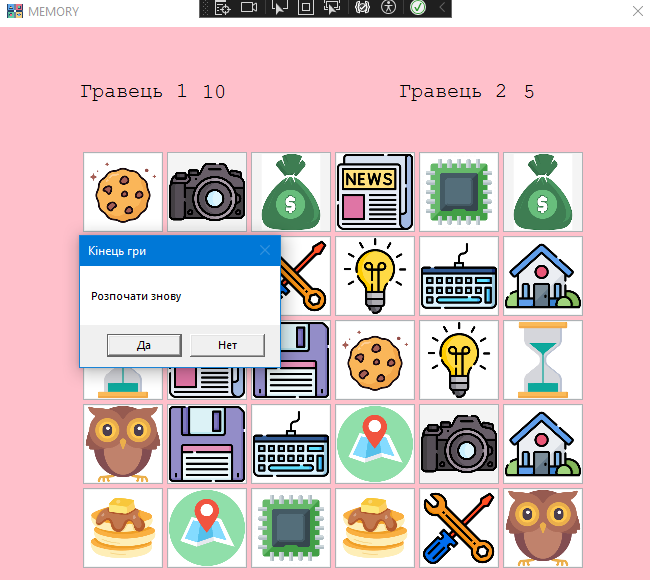
 **Рисунок Г.2 – Правильні поля**



**Рисунок Г.3 – Неправильні поля**



**Рисунок Г.4 – Закінчення гри**



**Рисунок Г.5**