Черкаський національний університет імені Богдана Хмельницького

(повне найменування вищого навчального закладу)

Кафедра програмного забезпечення автоматизованих систем

(повна назва кафедри)

КУРСОВА РОБОТА

з дисципліни "Об'єктно-орієнтоване програмування"

НА ТЕМУ «Узагальнення гри «Columns»»

Студента 2 курсу, групи КН-19

спеціальності «Комп'ютерні науки»

Поліщук Д.С

(прізвище та ініціали)

Керівник

Викл. Порубльов Ілля Миколайович

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

Національна шкала: _____

Кількість балів: _____ Оцінка: ECTS _____

Члени комісії _____ _____

(підпис) **(прізвище та ініціали)**

_____ _____

(підпис) **(прізвище та ініціали)**

_____ _____

(підпис) **(прізвище та ініціали)**

м. Черкаси – 2021 рік

Зміст

Вступ

Columns – відеогра жанру «puzzle», яка вийшла в 1990 році для платформи « Sega Mega Drive». Після неодноразово була перенесена на інші платформи. У гри є продовження «Columns II», яке було доступне тільки на території Японії. З 2010 року гра доступна майже на всіх популярних платформах. Колони були одні з багатьох головоломок, що з'явилися після успіху теріса наприкінці 1980-х років. Ігрова зона це високий прямокутник з наступними розмірами: 13 клітинок по вертикалі та 6 клітинок по горизонталі. Колони з трьома різними кольорами або різнокольоровими коштовностями з'являються у верхній частині ігрової зони та падають вниз призимляючись або на дно, або на фігурки, шо впали раніше. Під час того як фігурка падає, гравець може рухати її вліво та вправо, а також може циклічно прокручувати кольори ні цій фігурці. Після приземлення колони якщо три або більше однакових кольорів з'єднані вертикальною, горизонтальною або діагональною лінією, ці кольори зникають. Потім колони осідають під дією сили тяжіння. Якщо після осідання з'явиться лінія з кольорів, то вони теж зникають і цикл повторюється. Іноді з'являється чарівна колона, яка знищує всі кольори того самого кольору що знаходиться під нею. Фігурки падають швидше по мірі того як гравець знищує блоки. Мета гри – грати якомога довше поки нова колона не зможе з'явитися на ігровому полі, що закінчує гру. Гравці можуть набрати до 99 999 999 очок. Джерело[2]

Мета курсової роботи полягає в відтворені особливостей оригінальної гри та вдосконалення її за допомогою знань об'єктно-орієнтованого програмування

Найпростішою для розуміння буде розробка в середовищі «Visual Studio» в «Windows Forms», для реалізації ООП в даній грі буде використовуватись мова програмування C#.

Розділ 1. Огляд алгоритмів гри

1.1 Пошук інформації для створення гри

Так як завдання полягає в узагальненні гри то пошук інформації не складав якихось проблем. Не потрібно було придумувати гру з самого початку більшість інформації можливо знайти в інтернеті. Ідею логіки ігри продемонструвало відео, що представлене в документі з описом завдання, що виклав керівник курсової роботи. З відео я запозичив розмір ігрового поля та вибір кольорів. Реалізацію ігрового поля вирішив виконувати в виді двовимірного масива. Джерело [3]

1.2 Алгоритми та дизайн гри

Ігрове поле – має бути розміром 6 по горизонталі та 13 по вертикалі. Реалізацію ігрового поля можливо представити як двовимірний масив що заповнений нулями, тоді це буде ніби карта по якій будуть рухатись колони. Розмітка карти буде виконана за допомогою певних методів середовища таких як Graphics.DrawLine та Graphics.FillRectangle. Таким чином на формі з’являться квадрати та сітка, яка їх розділяє.

Потрібно буде реалізувати такий функціонал для гри:

- Кнопка «Play/Pause»;
- Вибір випадкового кольору для колон, кількість кольрів : 5(Синій, жовтий, зелений, червоний та оранжевий);
- Колона має падати з плином часу;
- Можливість руху колони вправо та вліво в межах ігрового поля;
- Можливість прокрутки кольорів колони;
- Закінчення гри після неможливості з’явиться новій колоні;

- Знищення квадратиків одного кольору, що можуть створити пряму лінію в трьох напрямках: вертикаль, горизонталь та по діагоналі.;
- Начислення та виведення балів, які рахуються прямо пропорційно до знищених квадратиків;
- Віконце, що буде відображати наступну колону;

Для демонстрації поліморфізму буде три види колон які: можуть тільки прокручуватися, можуть прокручуватися та повертатися на 180° та ті, які можуть прокручуватися та повертатися на 90°.

1.3 Пошук оптимального середовища для розробки

Для створення подібної гри можна було б скористатися спеціальним середовищем таким як «Godot Engine» але для вивчення самого середовища пішло б дуже багато часу. Тому оптимальним рішенням було розробляти гру в «Visual Studio», а саме в «Windows Forms». Оскільки я маю певні навички роботи з формами це спрощує задачу. Форми дають змогу реалізувати зрозумілий та в той же час простий інтерфейс.

1.4 Висновок до першого розділу

Найскладнішим для реалізації буде метод що буде знаходити лінії з однакових кольорів. Оскільки ігрове поле представлене масивом нулів для кожного кольору можна присвоїти свою цифру, щоб ідентифікувати кольорові квадрати на візуальний ігровій мапі. Знаходження однакових кольорів на одній лінії потребує проходження масиву мапи по діагоналі, що не так просто зробити. Неможливість знищення лінії відразу після її виявлення так як лінії одного кольору можуть

перекриватися змушує творити двовимірний масив булевих значень. Цей масив буде вмістити в собі інформацію про те які клітини на ігровому полі потрібно буде видалити після приземлення фігурки.

Розділ 2 Проектування логіки гри

2.1 Проектування реалізації об'єктно-орієнтованого програмування

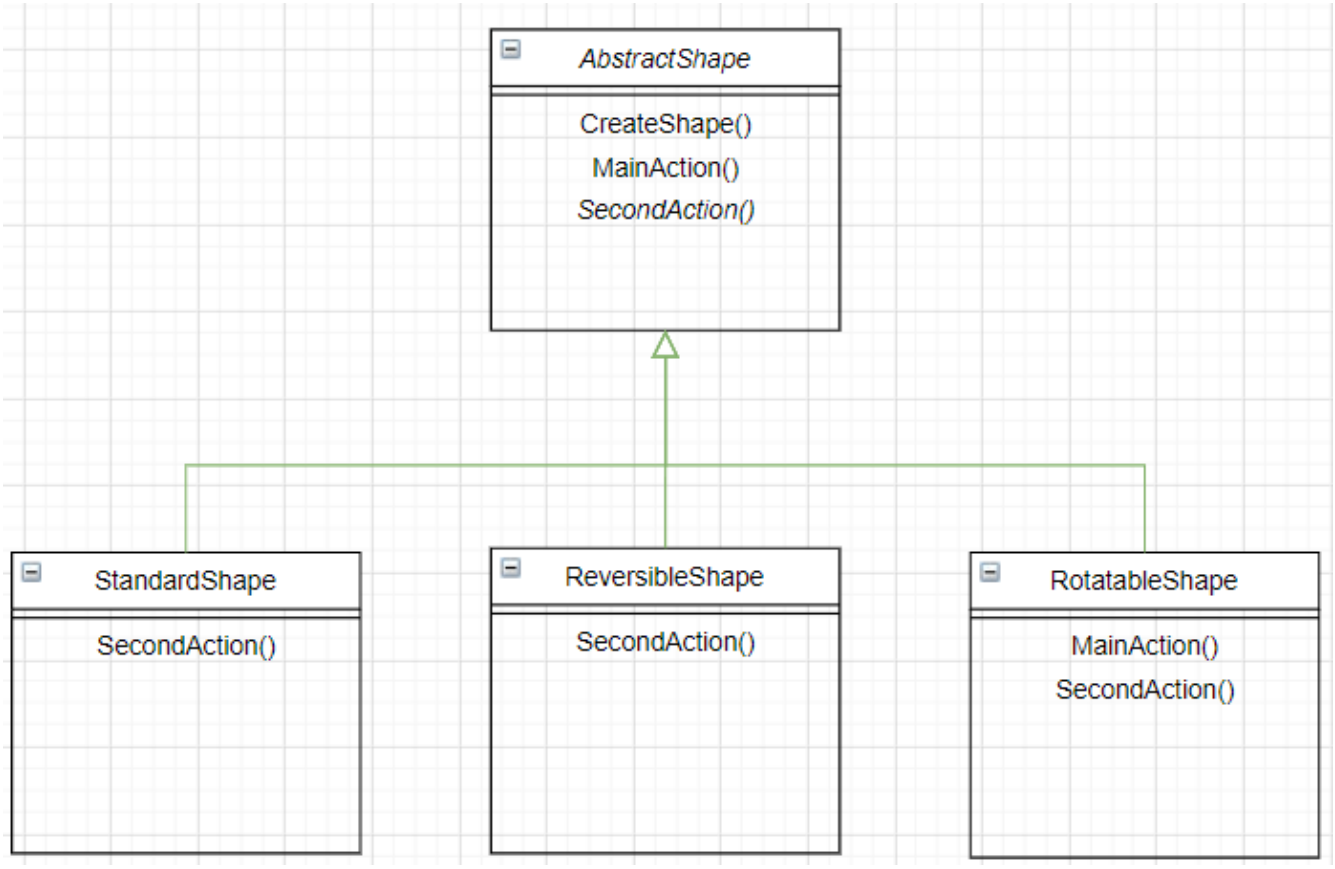

Рис 2.1 Діаграма класів різних видів колон

Як бачимо на Рис 2.1 у нас буде три вида колон. Батьківський клас AbstractShape буде містити в собі всі основні методи, що потребують колони такі як:

- Рух колони вправо
- Рух колони вліво
- Рух колони вниз
- Генерація колони
- Створення нової колони

Однин віртуальний метод MainAction() та один абстрактний метод SecondAction(). Клас стандартної колони буде містити лише пусту реалізацію метода основної дії. Клас реверсивної колони буде містити реалізацію метода побічної дії, який буде виконувати поворот на 180°. Клас Обертової колони має перевизначити основну дію, оскільки при повороті колони логіка прокрутки кольорів змінюється, та реалізувати метод побічної дії що буде виконувати поворот колони на 90°. Перевизначення та реалізація цих методів являє собою поліморфізм.

2.2 Блок-схема алгоритму оновлення

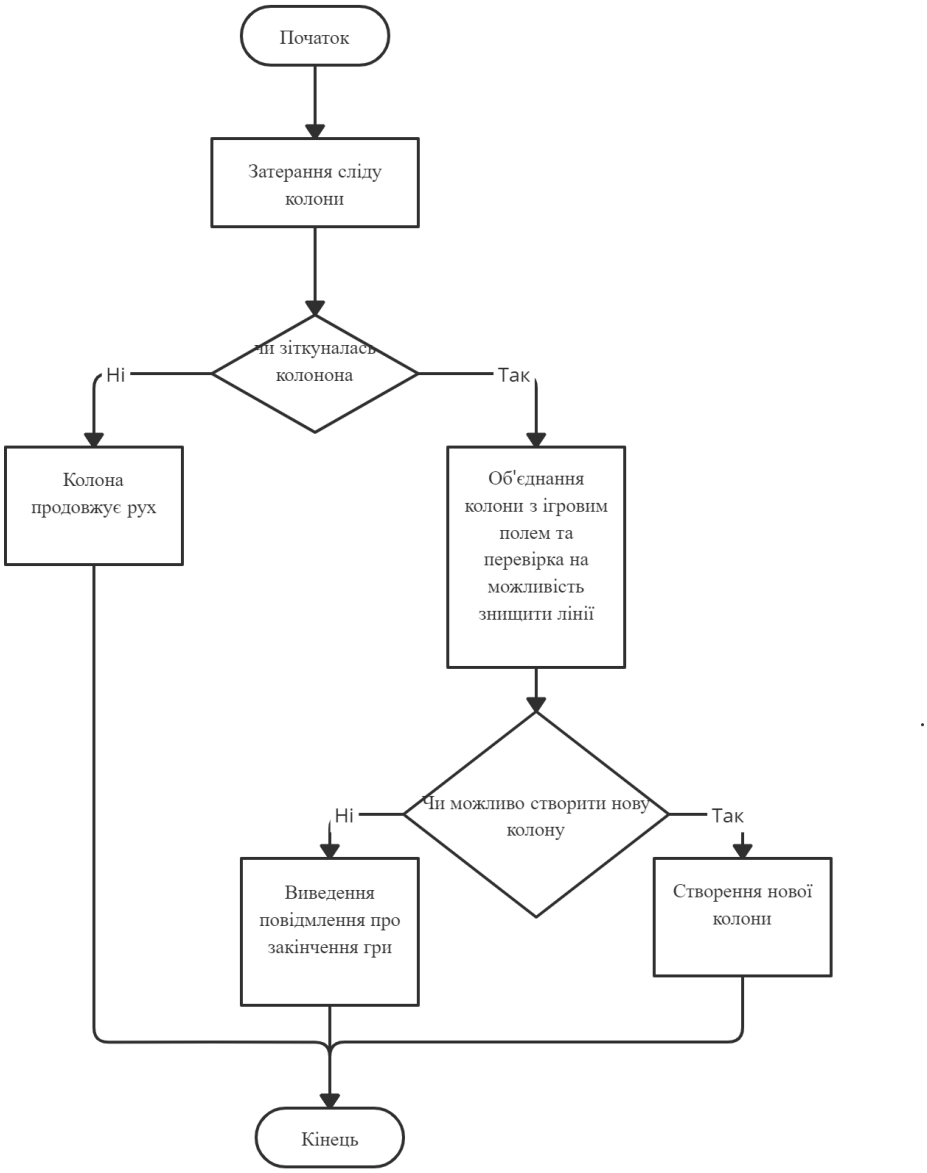

Рис 2.2 Блок-схема оновлення

Зіткнення колони буде перевіряти інший метод. Так як матриця колони та ігрової мапи ніяк не зв’язані до моменту зіткнення ми об’єднуємо колону та глобальне поле. Можливість створення нової фігурки перевіряє чи є пустим місце, що виділене під створення нових колон, якщо ні то запускається процес очистки ігрового поля, встановлення значення балів – по замовчуванню та виведення

повідомлення про закінчення гри з кінцевим результатом. Після цього кроку гра буде придатна до нової гри тобто буде циклічною. Ця функція буде викликатися кожні n – секунд за допомогою таймера. Таймер – це один із інструментів, що має подію Timer.Tick, ця подія викликається через кожний проміжок часу. Проміжок часу через який викликається подія налаштовується параметром Timer.Interval. З кожним знищеним квадратом інтервал буде трішки зменшуватись, це буде поступово ускладнювати гру, що зробить неможливим нескінченну гру.
Джерело [5]

2.3 Блок-схема повороту колони

Поворот колони реалізується транспонуванням її матриці. Транспонування дає змогу повернути колону на 90°. Створюється тимчасова матриця колони для того щоб мати змогу повернути колону біля границь ігрового поля. Після повороту колони біля границі колона змішується вліво чи вправо в залежності від того біля якої границі знаходилася.

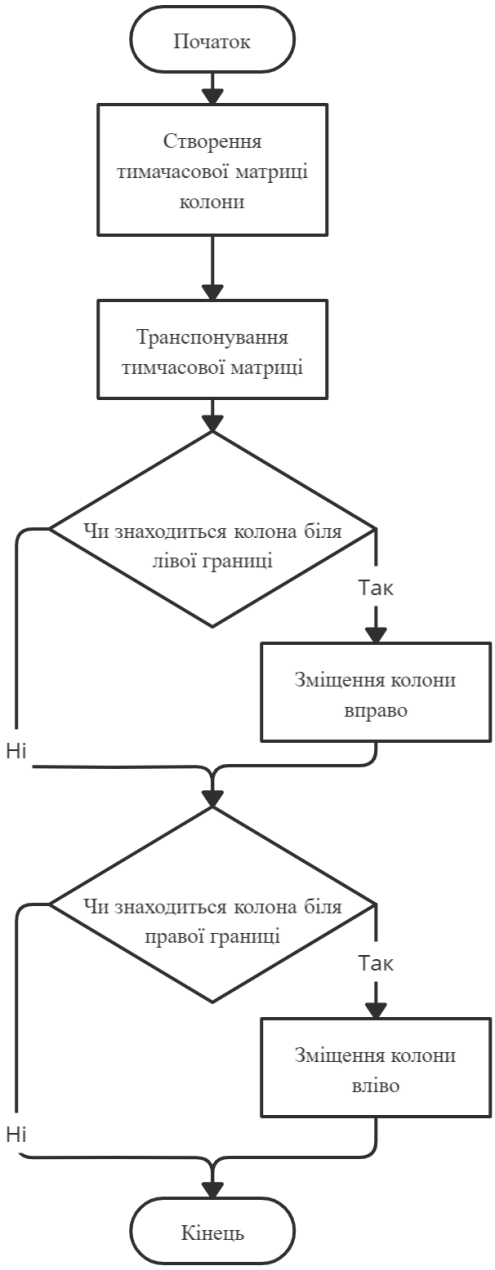

Рис 2.3 Блок-схема повороту фігурки

2.4 Висновки до другого розділу

Для повороту фігурки потрібно буде перевіряти ігрову мапу на наявність ненульових елементів щоб не було накладання колон однієї на іншу. При зміщенні вправо чи вліво також потрібно виконувати подібну перевірку. Потрібна реалізація

методу що буде зіставляти матрицю колони та матрицю ігрового поля. Знищення квадратів потребує перевірки в трьох напрямках. Це буде реалізовано вкладеними циклами для проходження по масиву ігрової мапи. Під час проходження будемо запам’ятовувати координати квадратиків, що відповідають умові (більше 3 в одну лінію). Після проходження буде викликатися метод, що замінює запам'ятовані клітинки на пусті та зараховує бали за їх знищення.

Розділ 3 Реалізація основних методів та тестування програми

3.1 Реалізація метода знищення ліній

Горизонтальне знищення ліній реалізовано наступним чином:

```
for (int i = 0; i < y; i++)
{
    count = 0;
    int prev = 0;
    for (int j = 0; j < x; j++)
    {
        if (map[i, j] != 0 && map[i, j] == prev)
        {
            count++;
        }
        else
        {
            if (count >= 3)
            {
                for (int k = j - count; k < j; k++)
                {
                    toRemove[i, k] = true;
                }
            }
            count = map[i, j] == 0 ? 0 : 1;
        }
        prev = map[i, j];
    }
    if (count >= 3)
    {
        for (int k = x - count; k < x; k++)
        {
            toRemove[i, k] = true;
        }
    }

}
```

Рис 3.1 фрагмент коду горизонтального знищення

Count – змінна, що зберігає кількість клітинок одного кольору що ідуть підряд. Вкладений цикл дозволяє перевірити кожен елемент масиву. prev – змінна, що зберігає попереднє значення, по замовчуванню – 0. За допомогою змінної prev ми можемо порівнювати поточне значення з минулим, що дає змогу рухатися по лінії одного кольору. Зовнішній цикл дозволяє рухатися по вертикалі вложений цикл реалізує прохід по горизонтальним елементам, тобто іде по наступному порядку:

0.0 потім 0.1 потім 0.2 а вже потім 1.0 (це для випадку якщо масив був би розмірності 3 на 3). Остання перевірка на те що лічильник нарахував більше або рівно три квадратика одного кольору, після неї ми проходимо в зворотному напрямку на кількість квадратів одного кольору та помічаємо їх для подальшого знищення. Головною умовою на порівняння рівності кольорів служить порівняння поточного значення з пустою клітинкою(тобто 0) та попередньою клітинкою, якщо такої клітинки не було вважається що вона була пустою

```
count = map[i, j] == 0 ? 0 : 1;
```

Рис 3.2 фрагмент коду з тернарним оператором

На Рис 3.2 фрагмент коду дозволяє почати рахунок лінії з першого ненульового елемента.

Вертикальне знищення відбувається за тим самим принципом що і горизонтальне але в другому напрямку.

```
for (int j = 0; j < x; j++)
{
    count = 0;
    int prev = 0;
    for (int i = 0; i < y; i++)
    {
```

Рис 3.3 демонстрація проходження масиву по вертекалі

Діагональне знищення :

```
for (int i = 0; i <= y; i++)
{
    int d = i;
    count = 0;
    int prev = 0;
    for (int j = 0; j < x && d <= y; j++, d++)
    {
        if (map[d, j] != 0 && map[d, j] == prev)
        {
            count++;
        }
        else
        {
            if (count >= 3)
            {
                for (int k = 1; k <= count; k++)
                {
                    toRemove[d - k, j - k] = true;
                }
            }
            count = map[d, j] == 0 ? 0 : 1;
        }
        prev = map[d, j];
    }
    if (count >= 3)
    {
        for (int k = 1; k <= count; k++)
        {
            toRemove[d - k, x - k] = true;
        }
    }
}
```

Рис 3.4 фрагмент коду діагонального знищення

Рис 3.4 – знищення ліній одного кольору, що йдуть по діагоналі зліва направо
Відмінність від інших напрямків тільки в специфічному проходженні по масиву, що вимагає додавання в цикл додаткового умов. Вкладений цикл іде по діагоналі починаючи з висоти яка задається зовнішнім циклом. В цілому логіка помітки квадратиків, які потрібно знищити, не змінюється. Все йде по такому ж принципу: запам’ятовуємо кількість однакових кольорів підряд потім йдемо в зворотному напрямку на кількість однакових кольорів підряд та відмічаємо їх для подальшого знищення.
Також оброблюється граничний випадок, коли цикл відмічає але так як останній елемент відповідав минулому то за цю ітерацію тільки збільшився лічильник, для цього робимо перевірку поза циклом, що проходить по горизонталі.

```
for (int i = 0; i <= y; i++)
{
    int d = i;
    count = 0;
    int prev = 0;
    for (int j = x - 1; j >= 0 && d <= y; j--, d++)
    {
```

```
if (count >= 3)
{
    for (int k = 1; k <= count; k++)
    {
        toRemove[d - k, k - 1] = true;
    }
}
```

Рис 3.5 Демонстрація зміни напрямку в коді

Для діагонального проходження справа наліво змінюється лише напрямок проходження та спеціальна умова для знищення граничних ліній. Гранична умова помічає діагональ, що починається з правого нижнього кута ігрового поля Реалізація Знищення:

```
for (int j = 0; j < x; j++)
{
    for (int i = 0; i < y; i++)
    {
        if (toRemove[i, j])
        {
            for (int k = i; k >= 1; k--)
            {
                map[k, j] = map[k - 1, j];
            }
            score++;
            flag = true;
        }
    }
}
```

Рис 3.6 Фрагмент коду знищення помічних квадратів

toRemove – це двовимірний масив булевих значень, яких зберігає в собі інформацію про клітинки що потрібно видалити. Під час знищення клітинок нараховуються бали. flag – булева змінна, яка зберігає в собі інформацію про те чи було запущене знищення.

```
if (flag)
{
    SliceMap();
}
```

Рис 3.7 Рекурсія методу

На Рис 3.7 фрагмент коду дозволяє рекурсивно викликати метод та перевірити чи можливо після знищення попередніх ліній з’явилися нові.

3.2 реалізація методу оновлення

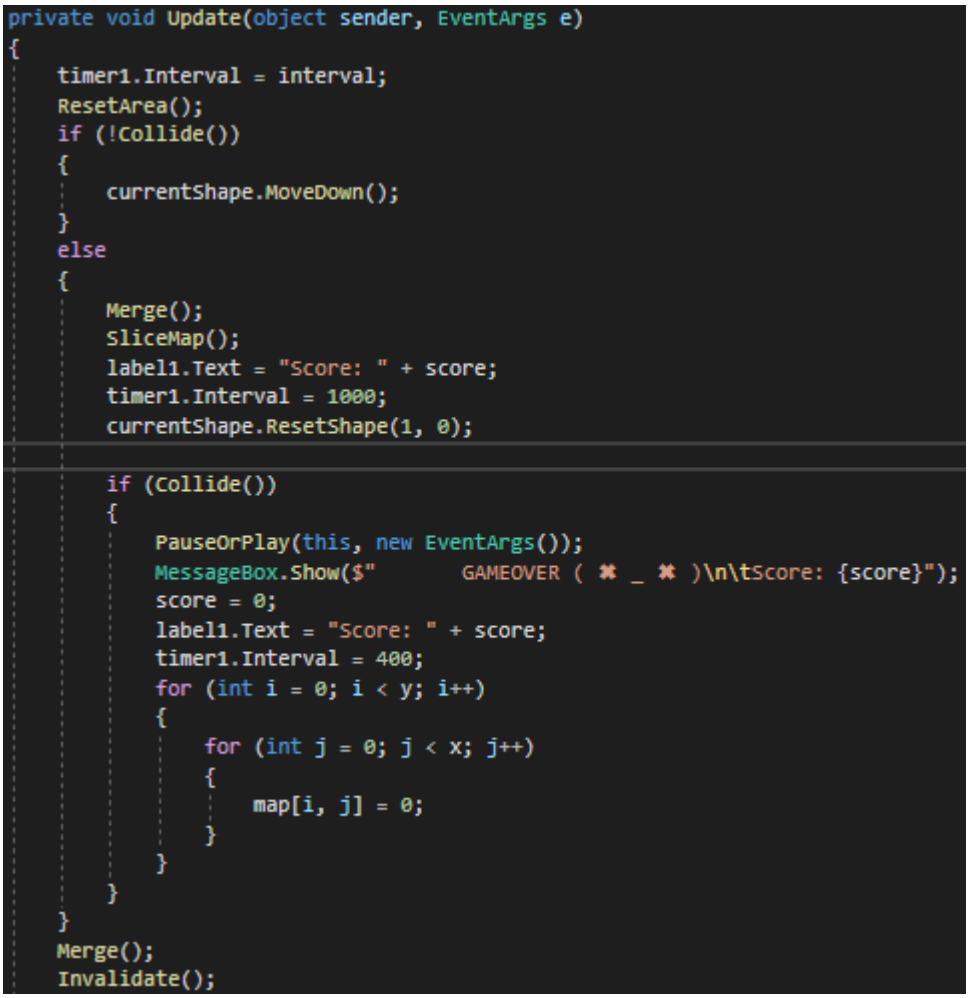

Рис 3.8 Реалізація методу оновлення

Метод що проілюстрований на Рис 3.8 викликається кожен тік таймера та служить певного роду анімацією руху колон. Якщо руху колони нічого не заважає то вона просто продовжує рух вниз. Коли руху колони щось заважає (дно ігрового поля чи інша фігура) викликається метод склейки колони з ігровим полем і метод перевірки та знищення ліній одного кольору – SliceMap. Якщо нова колона не може з’явитися то гра призупиняється, виводиться повідомлення про кінець гри,

встановлюється значення балів по замовчуванню та очищується ігрове поле. По замовчуванню вкінці метода викликається метод склейки, щоб синхронізувати колону з ігровою мапою

3.3 реалізація ООП у видах колон

Було прийнято рішення додати можливість зміни типу колон а не їх всіх наявність в одній грі, тобто зміна типу є свого роду обирання складності гри. Обрати тип колон можна за допомогою елемента ComboBox

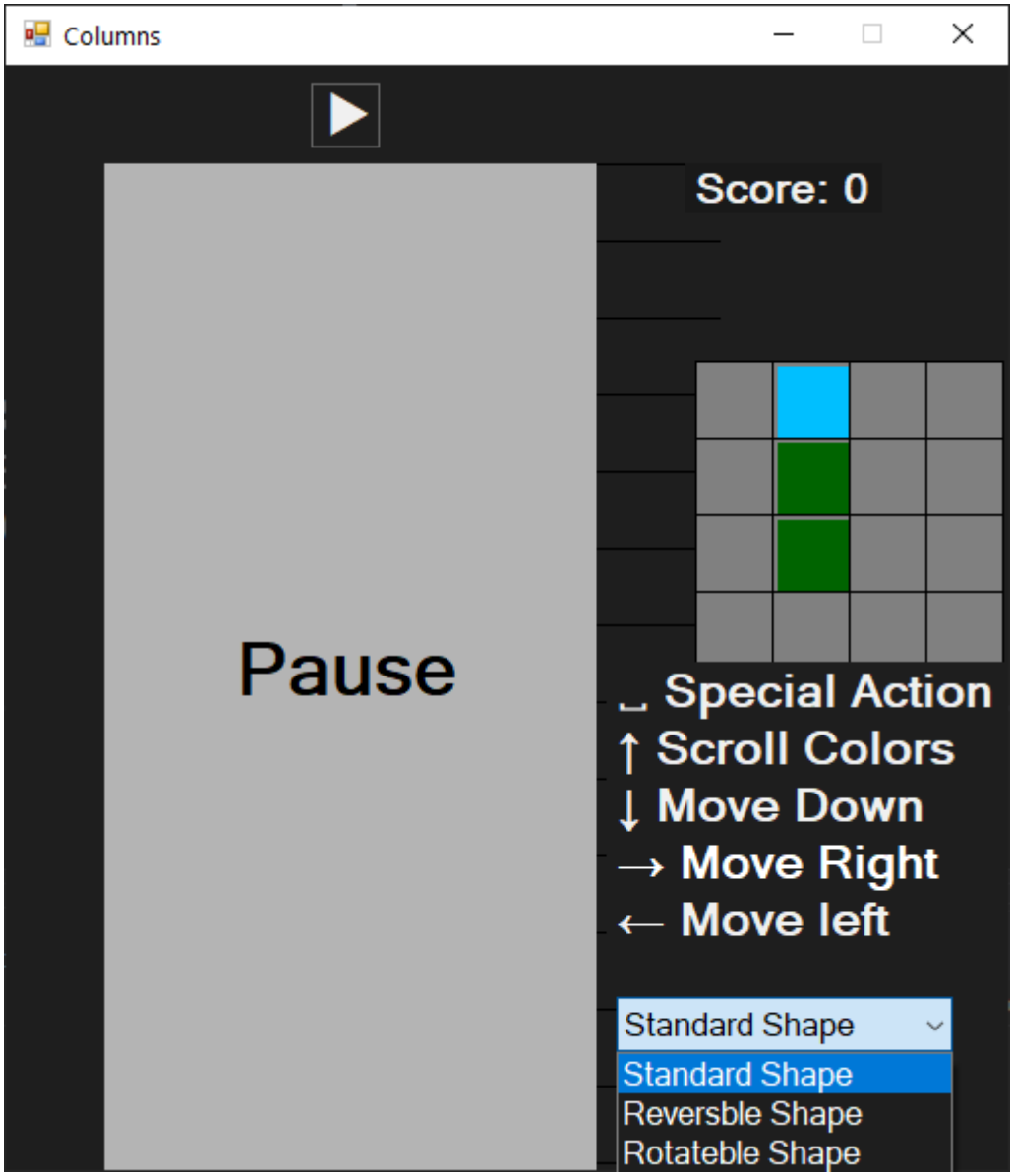

Рис 3.9 демонстрація можливості вибору типу

Абстрактний клас AbstractShape – батьківський клас в якому реалізовано основні та загальні для всіх типів фігурок методи та конструктор.

```
public AbstractShape(int _x, int _y)
{
    x = _x;
    y = _y;
    r = new Random(GetHashCode() + (int)DateTime.Now.Ticks);
    matrix = GenerateMatrix();
    sizeMatrix = (int)Math.Sqrt(matrix.Length);
    nextMatrix = GenerateMatrix();
    sizeNextMatrix = (int)Math.Sqrt(nextMatrix.Length);
}
```

```
public RotatebleShape(int _x, int _y) : base(_x, _y)
{

}
```

Рис 3.10 Демонстрація наслідування конструктора

Конструктор приймає на вхід дві координати для створення нової колони на ігровому полі. Створює нову та наступну колони.

```
public virtual void MainAction()
{
    int temp = matrix[2, 1];
    matrix[2, 1] = matrix[1, 1];
    matrix[1, 1] = matrix[0, 1];
    matrix[0, 1] = temp;
}
```

```
public override void MainAction()
{
    if (matrix[0, 1] != 0)
    {
        base.MainAction();
    }
    else
    {
        int temp = matrix[1, 2];
        matrix[1, 2] = matrix[1, 1];
        matrix[1, 1] = matrix[1, 0];
        matrix[1, 0] = temp;
    }
}
```

Рис 3.11 Перевизначення віртуального методу MainAction

В батьківському класі реалізується віртуальний метод MainAction. Віртуальний метод може бути перевизначеним в нащадках цього класу якщо в цьому є потреба. В випадку наведеному на Рис 3.11 перевизначається метод прокрутки, оскільки логіка прокрутки змінюється в залежності від положення колони: горизонтального чи вертикального. Прокрутка реалізована звичайним зсувом елементів масиву.

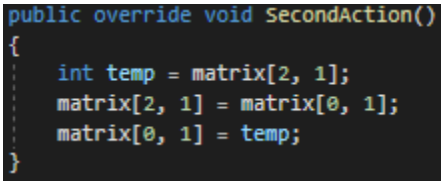

Рис 3.12 Приклад реалізації нащадком абстрактного методу SecondAction

В батьківському класі представлений абстрактний метод SecondAction, що призначений для виконання спеціального руху. На рис 3.12 реалізовано поворот колони на 180°, це просто зробити достатньо лише поміняти місцями верхній та нижній кольори.

Реалізація методів, проілюстрованих на Рис3.11 та Рис 3.12, являє собою поліморфізм.

3.4 Тестування програми

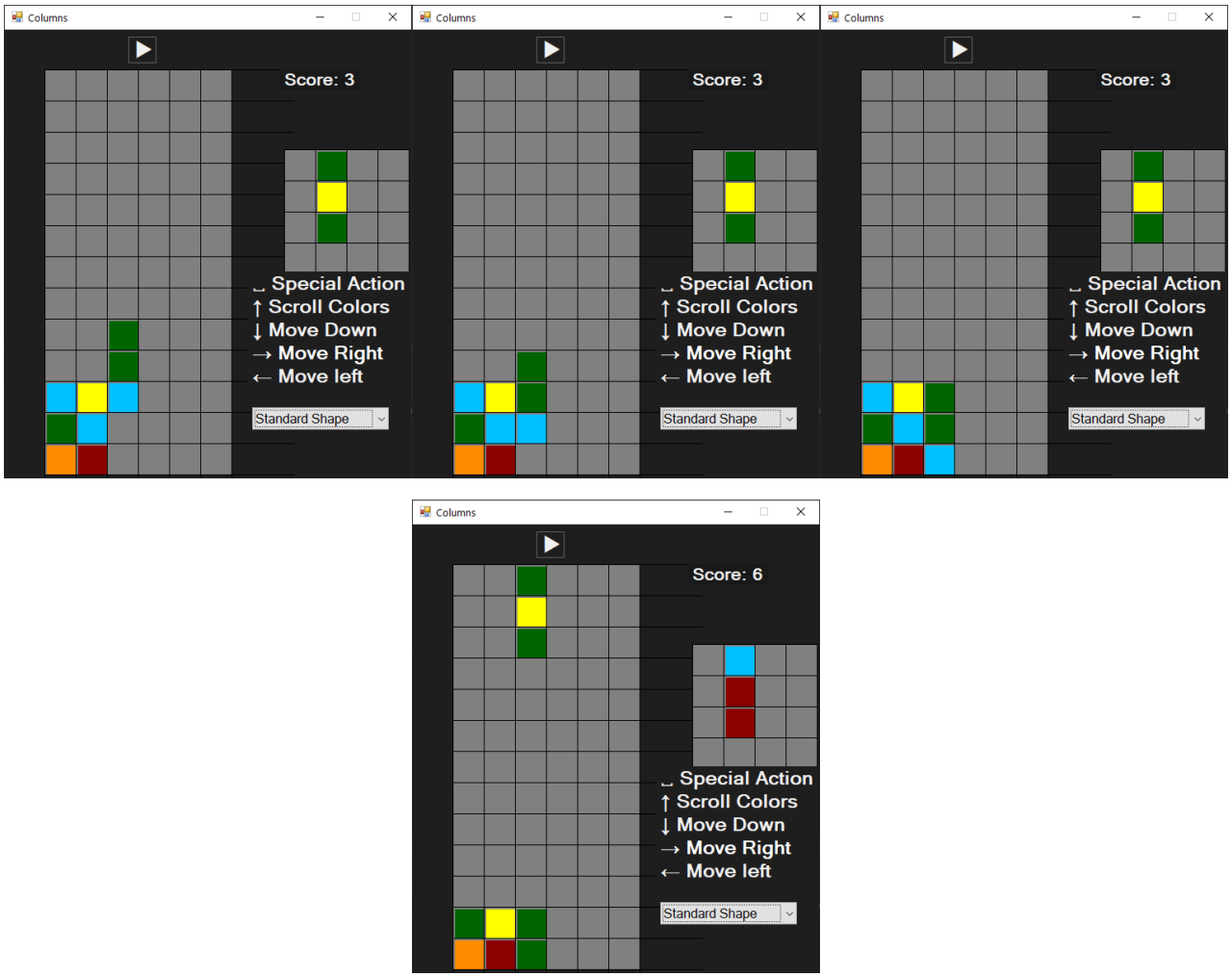

Рис 3.13 Демонстрація знищення лінії

На Рис 3.13 послідовно видно як падає колона та як знищується голуба лінія.

Висновоки

У результаті гра отримала весь запланований функціонал. Всю гру було написано на об'єктно-орієнтованій мові C#. Тестування гри показало нормальну роботу та не було виявлено ніяких непередбачуваних ситуацій. Було реалізовано задуману логіку наслідування типів колон та поліморфізм. Повністю реалізовано знищення ліній в усіх трьох напрямках. Win forms не пристосовані для створення за їх допомогою ігор але для таких простих як Columns їх функціоналу вистачило. Доречною ідеєю було б використовувати якийсь ігровий двигун.

Можливі розвитки програми:

- Створення локального списку рекордів
- Створення анімації знищення
- Додати нові види колон
- Перенесення гри з Windows Forms
- Покращення інтерфейсу

Список інформаційних джерел

1. Методичні вказівки до виконання та оформлення курсової роботи з дисципліни “Об‘єктно-орієнтоване програмування” для студентів, які навчаються за напрямами підготовки 050101 – „Комп’ютерні науки”, 050103 – „Програмна інженерія” та 040303 - „Системний аналіз” усіх форм навчання/ О.О. Супруненко, Ю.Є. Гребенович // Видавничий відділ Черкаського національного університету імені Богдана Хмельницького.
2. Columns (video game). Режим доступу: https://en.wikipedia.org/wiki/Columns_(video_game) перевірено: 02.06.2021.
3. My Top 10 Puzzle Games: #3 – Columns Режим доступу: https://www.youtube.com/watch?v=JPd4Yu2vEm8 перевірено 02.06.2021.
4. Godot. Режим доступу: https://ru.wikipedia.org/wiki/Godot перевірено 02.06.2021.
5. Timer Class Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer?view=net-5.0 перевірено 02.06.2021.