МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«04» червня 2025 р.

«МОБИЛЬНИЙ ВІДЕОРЕДАКТОР НА ПЛАТФОРМІ UNITY 3D»

Кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки»
Анастасьєва Данила Олеговича
Науковий керівник:
Мнацаканян Марія Сергіївна,
кандидат технічних наук, доцент кафедри системного аналізу та інформаційних технологій
Рецензент:
Охріменко Тетяна Олександрівна,
кандидат технічних наук, старший дослідник, заступник декана з наукової роботи факультету комп'ютерних наук та технологій Державного університету «Київського авіаційного університету»

Кваліфікаційна робота захищена
з оцінкою задовільно 67 (D)
Секретар ЕК

«11» червня 2025 р.

Київ– 2025

ЗМІСТ

ВСТУП

У сучасному світі соціальні медіа, такі як TikTok, YouTube, X (раніше Twitter) та Instagram, відіграють надзвичайно важливу роль у спілкуванні, самовираженні та споживанні інформації. Відео, зокрема, є основним форматом вмісту, який привертає увагу користувачів та стимулює взаємодію. Від коротких кліпів до довгих форматів, відео створюють захоплюючий досвід, що дозволяє користувачам швидко сприймати інформацію та взаємодіяти з контентом.

Однак, незважаючи на широке поширення відео як основного медіа, багато мобільних додатків все ще не надають достатньо функцій для повноцінного редагування відео. На відміну від фотографій, які можна легко обробити за допомогою різноманітних фільтрів, редакторів, ефектів та корекцій кольору, відео вимагають більш складних та потужних інструментів для редагування. Необхідність у таких інструментах зумовлена різноманітністю задач: від простого обрізання та з’єднання відео до додавання тексту, музики, ефектів, водяних знаків, анімації та складних візуальних змін. Обмеження наявних мобільних редакторів відео часто змушує користувачів використовувати настільні рішення, що є незручним та займає багато часу, або обмежуватися базовими функціями, що може призвести до втрати потенціалу для творчості та самовираження. Відсутність зручних та функціональних мобільних редакторів відео стримує розвиток креативного контенту та може впливати на користувацький досвід в цілому.

РОЗДІЛ 1. ОПИС ПРОБЛЕМИ ТА ЦІЛЬ РОБОТИ

1.1. Введення

Метою даної роботи є розробка і проектування зручного у використанні візуального відеоредактора, оптимізованого для мобільних пристроїв.

Серед популярних програм для редагування відео, таких як Shotcut, DaVinci Resolve та VSDS, більшість із них призначені для використання на ПК. Однак у нашому швидкоплинному житті час редагування відео часто обмежений. Тому простий і зручний мобільний додаток, що дозволяє накладати ефекти, обрізати відео і виконувати інші базові операції, може значно спростити життя користувачів.

Яскравими прикладами популярності відео в жанрі розваг є два відеохостинги – YouTube та TikTok. YouTube відомий своєю широкою аудиторією та різноманітністю контенту, включаючи відео різних жанрів, від влогів до музичних кліпів. З іншого боку, TikTok став популярним серед молоді завдяки коротким відеороликам, танцям та творчому контенту, який швидко здобуває популярність. Обидва сервіси відкривають нові можливості для творчості та спілкування в онлайн-середовищі.

Рисунок 1.1 – Логотип YouTube

YouTube був заснований в 2005 році і до 2024 року став найпопулярнішим відеохостингом в світі з більш ніж 30 мільярдами користувачів.

Стаючи невід'ємною частиною повсякденного життя мільйонів людей, YouTube продовжує розвиватися і пропонувати унікальні можливості для створення, перегляду та обміну відеоконтентом.А з можливістю редагувати відео під час поїздки на автобусі або відпочинку на роботі спростить користувачам свою працю.

Рисунок 1.2 – Логотип TikTok

Запущена в 2017 році, міжнародна версія TikTok стала провідною відеоплатформою для коротких відео в Китаї і набуває все більшої популярності в інших країнах, ставши одним з найшвидше зростаючих та скачуваних додатків. У вересні 2021 року щомісячна аудиторія TikTok перевищила 1 мільярд осіб.

У зв'язку з цим я вибрав тему «мобільний відеоредактор на платформі Unity» для своєї кваліфікаційної роботи. Unity-це потужна платформа для розробки, яка дозволяє створювати інтерактивні програми та ігри, а також має великий потенціал для розробки мультимедійних інструментів. Використання

Unity для створення мобільного відеоредактора відкриває нові можливості для інтеграції різних функцій.

1.2. Чому Unity та що программа повинна робити?

Перш за все, я вже давно знайомий з цією платформою, яка переважно використовується для розробки ігор. Так, можливо, це не найкраща платформа для створення відеоредактора під мобільні пристрої. Є й кращі варіанти, як, наприклад, Android Studio або Xamarin. Проте, я дотримуюсь думки, що платформи та мови програмування – це лише інструменти в руках розробника. Тому, вважаю, що головне – вміння та бачення творця, а не конкретний вибір інструменту.

Відеоредактор повинен мати інтуїтивно зрозумілий та зручний функціонал, що дозволяє швидко вносити зміни та редагувати відеоматеріали. Зокрема, програма повинна забезпечувати такі можливості:

- **Зміна розширення.** Розширення файлу – це важлива операція, яка може бути необхідна для зручного використання.
- **Зміна формата відеофайлу.** Зміна формату корисна коли программа або платформа требует файл другого формату.
- **Об’єднання двох та більше відеофайлів.** Функція об’єднання декількох відеофайлів в один є ключовою, оскільки дозволяє користувачам зручно збирати та компонувати різні фрагменти відео в єдине цілісне відео.
- **Вирізання та обрізання фрагментів відео.** Вирізання та обрізка окремих фрагментів відео є універсальною функцією, яка дозволяє як видаляти небажані частини, так і отримувати цінні епізоди для використання в інших відеоматеріалах.

- **Додавання та вилучення аудіофайлу з відеофайлу.** Можливість додавання або вилучення аудіофайлів з відео є критично важливою, оскільки відеоматеріал є комплексним об’єктом, який поєднує в собі візуальну та аудіальну складові.
- **Застосування різноманітних фільтрів.** Застосування різноманітних фільтрів дозволяє покращити візуальну складову відео та надати йому унікального стилю.

РОЗДІЛ 2. ОГЛЯД ТЕХНОЛОГІЇ РОЗРОБКИ ВІДЕОРЕДАКТОРА

2.1. Технології

Програма буде розроблена на платформі Unity, а саме на версії 2022.1.23f1.

1.3 –Логотип Unity

Вибір Unity обумовлений її простотою використання та потужними інструментами для створення інтерфейсу користувача (UI). Unity значно полегшує розробку UI-елементів, що дозволяє зосередитися на функціональності та дизайні програми. Крім того, інтегровані плагіни NDK, SDK та JDK, які входять до складу Unity, забезпечують можливість безперешкодного перенесення проєкту на платформу Android, що робить процес розробки більш ефективним та зручним.

В Unity, основною мовою програмування для розробки є C#, хоча, звичайно, можливості розширення функціональності включають використання різних плагінів та бібліотек, написаних іншими мовами, такими як C++ та Java. C# широко використовується для сценаріїв, що визначають поведінку ігрових об’єктів, логіку гри, управління інтерфейсом користувача та багато іншого.

C# здобув популярність, як мова, що забезпечує більш зручний і ефективний досвід розробки в порівнянні з його попередниками, такими як C і C++.

Для написання та редагування коду я буду використовувати SharpDevelop 5.1. Мій вибір обумовлений кількома ключовими факторами, які, на мою думку, сприяють більш ефективному та комфортному розвитку.По-перше, SharpDevelop, на відміну від більш універсальної Microsoft Visual Studio, орієнтований на мову C#. Ця спрямованість забезпечує оптимальну підтримку функцій і можливостей.

По-друге, SharpDevelop відрізняється більш простим інтерфейсом і конфігурацією в порівнянні з Visual Studio. Це дозволяє швидше освоїтися з інструментом і не витрачати час на навігацію по складних меню і налаштувань, що в кінцевому підсумку прискорює процес розробки.Незважаючи на те, що Visual Studio є потужною і багатофункціональною IDE, для конкретно мого завдання, де пріоритет віддається швидкому запуску і комфортній роботі з C#, SharpDevelop представляється більш підходящим інструментом.

На мою думку, ключовою технологією проекту є плагін Ffmpeg.Саме на цей плагін дозволяє взаємодіяти і редагувати відеофайлами і цей плагін є фундаментом моєї програми.Використання Ffmpeg є критично важливим для реалізації ключових функцій, таких як додавання ефектів, накладання аудіо, зміна роздільної здатності та формату відео.

Тому я вважаю Ffmpeg не просто важливим компонентом, а фундаментом, на якому будується вся функціональність моєї програми.

FFmpeg – це потужний набір вільних бібліотек із відкритим вихідним кодом, що надає широкі можливості для роботи з мультимедійним контентом.

В основі Ffmpeg лежать такі компоненти:

libavcodec: Бібліотека, що забезпечує кодування та декодування аудіо- та відеоданих, підтримуючи велику кількість кодеків.

Libavformat: Бібліотека, що відповідає за мультиплексування (упакування) та демультиплексування (розпакування) аудіо- та відеопотоків у медіаконтейнери, такі як MP4, AVI, MKV та інші.

Назва FFmpeg, ймовірно, походить від абревіатури MPEG (експертна група з рухомого зображення) та FF, що можна інтерпретувати як “Fast Forward” (швидка перемотка вперед), що вказує на швидкість та ефективність обробки медіаданих цим інструментом. Використання Ffmpeg значно розширює можливості проекту, забезпечуючи підтримку широкого спектру форматів та кодеків.

2.2. План розробки

Для подальшої розробки програми бажано мати чіткий план, який допоможе структурувати роботу та забезпечити послідовну реалізацію функціональності. Враховуючи, що програма складатиметься з трьох основних частин – менеджера файлів, відеоплеєра та панелі з функціями Ffmpeg.

1. Оскільки в Unity немає вбудованого менеджера файлів, його доведеться реалізувати самостійно. Основна задача менеджера файлів у вашому проекті – забезпечити вибір файлів та навігацію по каталогах.

2. Відеоплеєр, як ключовий компонент програми, повинен забезпечувати зручне відтворення відеофайлів. Реалізація простих, але важливих функцій, таких як перемотування, запуск, зупинка, відображення часу та назву файлу.

3. Найважливішою складовою програми, без якої її функціонування було б неможливим, є плагін FFmpeg. Саме завдяки FFmpeg з’являється можливість здійснювати широкий спектр операцій з відеофайлами.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ПРОГРАМИ

3.1. Менеджер

Як я писав раніше в Unity відсутній вбудований файловий менеджер, тому його потрібно реалізовувати самостійно. Перш за все для зручної та структурованої організації файлів у проекті Unity створіть наступні папки: scripts, prefabs, plugins .Тепер почнемо реалізовувати менеджер, кликом правою кнопкою миші у вікні Hierarchy, перейдіть до UI та додайте елемент Canvas. Налаштуйте Canvas, щоб він відображався під вашою камерою (Render Mode: Screen Space - Camera, Render Camera: ваша основна камера). Далі, додайте Scroll View (UI -> Scroll View) як дочірній елемент Canvas. У цьому випадку, Scroll View буде представляти собою вертикальний список, який міститиме вміст файлового менеджера.

Створимо наступні елементи UI:

- **Button_Back:** Кнопка для переходу назад у попередню директорію.
- **Button_Exit:** Кнопка для виходу з менеджера.
- **Prefabs:**
    - **Directories:** Префаб, що представляє собою кнопку для відображення директорії.
    - **Files:** Префаб, що представляє собою кнопку для відображення файлу.

Рисунок – Фінальний вид менеджера

Щоб налаштувати Scroll View для вертикального списку, видаліть горизонтальну смугу прокрутки (scrollbar horizontal), оскільки вона не потрібна. Потім перейдіть до налаштувань об’єкта Content і в компоненті Grid Layout Group встановіть для параметра Constraint значення Fixed Column Count, а в Constraint Count вкажіть 1. Це розташує префаби у вертикальний стовпчик. Для створення інтервалу між префабами встановіть значення Y у секції Spacing на 50 также в об’єкті Scroll View в компоненті Image змінить прозорість до 255 для подальшого зручності.

В папці scripts створюємо скрипт Manager.cs відкриваєм його та пишемо такий код.

Файл Manager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Android;
using System.IO;
using System.Linq;
using UnityEngine.Video;
```

```
// Ці бібліотеки забезпечують необхідну функціональність для роботи скрипту
public class Manager : MonoBehaviour
{
public GameObject
filePrefab,fileListPanel,fileContentPanel,SelectButton,buttonsPanel,directoryPrefab;
//filePrefab;   Префаб для відображення файлів у списку.

//fileListPanel;  Панель, що містить Scroll View зі списком файлів.

//fileContentPanel;  Панель Content в Scroll View

//SelectButton;  Кнопка вибору файлу.

//buttonsPanel;  Панель з кнопками керування.

//directoryPrefab;  Префаб для відображення директорій у списку.


private DirectoryInfo dirInfo = new DirectoryInfo("/mnt/sdcard/Download/");

/*dirInfo представляє директорію /mnt/sdcard/Download/. Ця директорія зазвичай
використовується на Android-пристроях для збереження завантажених файлів.*/
private DirectoryInfo[] directories;

/*Цей масив буде використовуватися для зберігання інформації про всі піддире-
кторії в поточній директорії */

private List<FileSystemInfo> allItems = new List<FileSystemInfo>();
//Буде зберігати як файли, так і директорії в одному списку

private FileInfo[] files;
//Масив об’єктів FileInfo

private GameObject[] instancedObject;
//Цей масив буде використовуватися для зберігання посилань на  об’єктів UI

private string typeFile;
//Зберігає тип файлу

private string[] extensions;
//Збірігає масив список розширень файлів

public static Manager instance;
/*Оголошує статичну змінну instance типу Manager. static означає, що ця змінна
належить не конкретному об’єкту класу Manager, а самому класу*/
```

```
private void Awake()
{
instance = this;
}
public void Load()
{
ClearFiles();
files = extensions.SelectMany(ext =>
dirInfo.GetFiles(ext,SearchOption.TopDirectoryOnly)).ToArray();
directories = dirInfo.GetDirectories();
/*Отримує список файлів з вказаними розширеннями (extensions) та директорій
з поточної директорії (dirInfo).*/

allItems.AddRange(files);
allItems.AddRange(directories);
// Об’єднує файли та директорії в один список (allItems).


instancedObject = new GameObject[allItems.Count];
/*Створює масив instancedObject для зберігання посилань на об’єкти, що пред-
ставляють файли та директорії в UI.*/


for(int i = 0;i < allItems.Count;i++)
{
if (allItems[i] is FileInfo)
{
FileScripts file =
Instantiate(filePrefab,fileContentPanel.transform).GetComponent<FileScripts>();
file.FileNameText.text = allItems[i].Name;
file.index = i;
instancedObject[i] = file.gameObject;
}
else if (allItems[i] is DirectoryInfo)
{
DirectoriesScripts directory =
Instantiate(directoryPrefab,fileContentPanel.transform).GetComponent<DirectoriesS
cripts>();
directory.FileNameText.text = allItems[i].Name;
directory.index = i;
instancedObject[i] = directory.gameObject;
}
```

```
/*Для кожного елемента в allItems створює відповідний префаб (файлу або директорії), встановлює його ім’я та індекс, і зберігає посилання на об’єкт в instancedObject.*/
}
}

public void SortiredFileLoad(string x)
{
 typeFile = x;
if(x == "firstVideoFile")
extensions = new string[] { "*mp4", "*mov","*mkv","*avi"};
else if(x == "image")
extensions = new string[] { "*jpg", "*png"};
else if(x == "secondVideoFile")
extensions = new string[] { "*mp4", "*mov","*mkv","*avi"};
else if(x == "audio")
extensions = new string[] { "*mp3"};
else
extensions = new string[] {x};
Load();
}
/*  Функція SortiredFileLoad(string x) сортує файли за заданим типом та відображає їх у списку.*/

public void SelectDirectories(int index)
{
dirInfo = (DirectoryInfo)allItems[index];
Load();
}
/*Функція SelectDirectories(int index) виконує переход з поточної діректорії в піддиректорію*/

public void BackDirectories()
{
if(dirInfo.FullName != "/mnt/sdcard"){
dirInfo =  dirInfo.Parent;
Load();
}}
/* Функція BackDirectories() дозволяє перейти на один рівень вгору у файловій системі. Якщо поточна директорія (dirInfo.FullName) не є кореневою директорією (“/mnt/sdcard”)*/
```

```
public void ClearFiles()
{
if(instancedObject != null)
{
foreach (GameObject obj in instancedObject) {
if(obj != null){
Destroy(obj);
}}
}
allItems.Clear();
instancedObject = null;
}}
/*Функція ClearFiles() очищає список відображених файлів та директорій, зни-
щуючи відповідні об'єкти в UI.*/
```

У вікні **Hierarchy** створіть новий пустий об’єкт ProjectManager і додайте до нього скрипт Manager.cs. Потім в інспекторі цього об’єкта перетягніть об’єкти Scroll View та Content, щоб присвоїти їм відповідні посилання.

Створіть три кнопки: Button_Exit, Button_Back та Button_Select останнього перетягуємо в Manager.cs.

Зробіть Button_Exit та Button_Back дочірніми для Scroll View.

Для кнопки **Button_Exit**: в компоненті Button, в секції On Click, додайте два виклики функції.

1. В першому виклику перетягніть об’єкт Scroll View і виберіть функцію GameObject.SetActive(false).
2. В другому виклику перетягніть об’єкт Button_Select і виберіть функцію GameObject.SetActive(true).

Для кнопки **Button_Back**: в компоненті Button, в секції On Click,додаєм виклик функції ,перетягуємо ProjectManager та вибераємо в скрипті Manager.cs функцію BackDirectories().

Для кнопки **Button_Select**: в компоненті Button, в секції On Click, додайте три виклики функції.

1. В першому виклику перетягніть об’єкт ProjectManager та вибераємо в скрипті Manager.cs функцію SortiredFileLoad(firstVideoFile).

2. В другому виклику перетягніть об’єкт Button_Select і виберіть функцію GameObject.SetActive(false).

3. В третьому виклику перетягніть об’єкт Scroll View і виберіть функцію GameObject.SetActive(true).

У скрипті Manager.Load() створюватимуться об’єкти Directories та Files. Об’єкти Directories потрібні для переходу в іншу директорію, а Files - для вибору файлу. Розпочнемо зі створення префабів.

Створіть в UI об’єкт Image, перейменуйте його на “File”, оберіть круглий спрайт та зафарбуйте його синім для візуальної відмінності від об’єктів Directory. Додайте до нього компонент Button. Щоб створити об’єкт Directory, виконайте ті самі дії, перейменувавши Image в “Directory” та зафарбувавши його червоним.

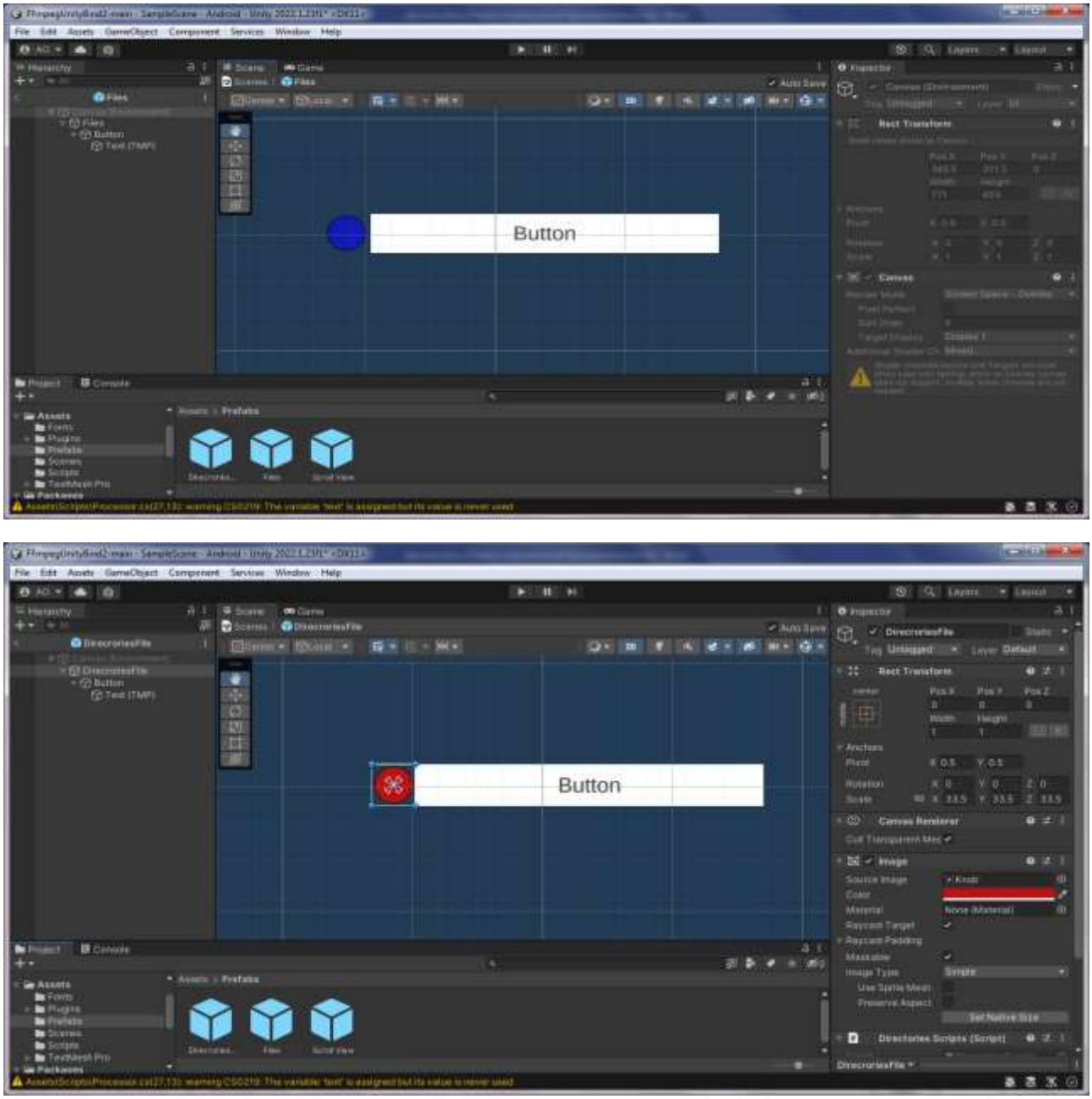

Рисунок 1.4 File та Directory

В папці Scripts створюємо два скрипти DirectoriesScripts та FilesScripts та пишимо такий код:

FileScripts.cs

```
using UnityEngine;
using UnityEngine.UI;
using TMPro;
public class FileScripts : MonoBehaviour
{
public TMP_Text FileNameText;
```

```
[HideInInspector]
public int index;

public void OnClick()
{
Manager.instance.SelectFile(index);
}
}
```

Цей код викликає функцію SelectFile(index) для вибору файла.

DirectoriesScripts.cs

```
using UnityEngine.UI;
using UnityEngine;
using TMPro;

public class DirectoriesScripts : MonoBehaviour
{
public TMP_Text FileNameText;
[HideInInspector]
public int index;

public void OnClick()
{
Manager.instance.SelectDirectories(index);
}}
```

Цей код викликає функцію SelectDirectories(index) для перехода на один рівень вгору у файловій системі.

Додаємо код в **Manager.cs**

```
public void SelectFile(int index)
{
if(typeFile == "firstVideoFile"){

}
else {
}
fileListPanel.SetActive(false); buttonsPanel.SetActive(true);
}
```

Зараз цей код виконує мінімальну функціональність і фактично є лише заглушкою. Щоб реалізувати його повноцінну роботу, необхідно інтегрувати плагін FFmpeg та розробити інтерфейс для перевірки працездатності.

В папку Plugins переносим плагин FFmpeg а в папці Scripts создаємо два файла FFmpegWrapper.cs та FFprobeWrapper.cs.

FFmpegWrapper.cs,

```
using UnityEngine;
using System.IO;
public class FFmpegWrapper
{
public static int Execute(string command)
{
#if UNITY_ANDROID
using (AndroidJavaClass configClass = new
AndroidJavaClass("com.arthenica.mobileffmpeg.Config"))
{
AndroidJavaObject paramVal = new
AndroidJavaClass("com.arthenica.mobileffmpeg.Signal").GetStatic<AndroidJavaObj
ect>("SIGXCPU");
configClass.CallStatic("ignoreSignal", new object[] { paramVal });

using (AndroidJavaClass ffmpeg = new
AndroidJavaClass("com.arthenica.mobileffmpeg.FFmpeg"))
{
int code = ffmpeg.CallStatic<int>("execute", new object[] { command });
return code;
}
}
#elif UNITY_IOS
return MobileFFmpegIOS.Execute(command);
#else
return 0;
#endif
}
```

Цей код забезпечує підключення до функцій FFmpeg, і саме через нього відбуватиметься взаємодія з ним.

FFprobeWrapper.cs

```
using System;
using UnityEngine;
using System.IO;
public class FFprobeWrapper
{
public static int Execute(string command)
{
#if UNITY_ANDROID
using (AndroidJavaClass configClass = new
AndroidJavaClass("com.arthenica.mobileffmpeg.Config"))
{
AndroidJavaObject paramVal = new
AndroidJavaClass("com.arthenica.mobileffmpeg.Signal").GetStatic<AndroidJavaObj
ect>("SIGXCPU");
configClass.CallStatic("ignoreSignal", new object[] { paramVal });

using (AndroidJavaClass ffprobe = new
AndroidJavaClass("com.arthenica.mobileffmpeg.FFprobe")) // Use FFprobe class
{
int code = ffprobe.CallStatic<int>("execute", new object[] { command });
return code;
}
}
#elif UNITY_IOS
return MobileFFmpegIOS.Execute(command); // Assuming MobileFFmpegIOS
supports FFprobe
#else
return 0;
#endif
}
public static float GetVideoDuration(string videoFilePath)
{
string command = $"-v error -show_entries format=duration -of
default=noprint_wrappers=1:nokey=1 \"{videoFilePath}\"";
int resultCode = Execute(command);

if (resultCode == 0)
{
string output = GetLastCommandOutput();
//Debug.Log(output);
float duration =
```

```
float.Parse(output.Trim(),System.Globalization.CultureInfo.InvariantCulture);
return duration;
}
else
{
Debug.LogError("FFmpegWrapper: Error executing FFprobe command.");
return -1;
}}
public static string GetVideoSize(string videoFilePath,string directory)
    {
        string filePath = Path.Combine(directory, "myFileProbe.txt");
        string command = $"-v error -select_streams v:0 -show_entries
stream=width,height -of csv=s=x:p=0  \"{videoFilePath}\"";
        int resultCode = Execute(command);
        string output = GetLastCommandOutput();
        output = output.Replace('x', ':');
        UnityEngine.Debug.Log(filePath);

        if (resultCode == 0)
        {
           File.WriteAllText(filePath, output);
          UnityEngine.Debug.Log(output);
          return output;
        }
        else
        {
           UnityEngine.Debug.LogError("FFmpegWrapper: Error executing FFprobe
command.");
           File.WriteAllText(filePath, output);
           UnityEngine.Debug.LogError(output);
           return "0:0";
        }} }
```

Цей код реалізує отримання різних характеристик відео та зображень. Зокрема, функція GetVideoDuration(string videoFilePath) призначена для визначення тривалості відео, що стане в нагоді на наступних етапах розробки.

3.2. Відеоплеєр

Тепер необхідно реалізувати відеоплеєр для роботи відеоредактора в Unity, оскільки вбудованого плеєра у Unity немає. Цей плеєр буде побудований за аналогією з файловим менеджером.

1. В ієрархії (Hierarchy) створіть об’єкт Image. Перейменуйте його на **VideoBackground**. Він буде слугувати фоном екрану плеєра.

2. Створіть об’єкт Raw Image і зробіть його дочірнім об’єктом **VideoBackground**. Перейменуйте його на **VideoWindow**. Цей об’єкт буде фактичним екраном, на якому відображатиметься відео.

3. Створіть три об’єкти Button і зробіть їх дочірніми об’єктами **VideoBackground**:

- **ButtonPause**: Кнопка для призупинення відтворення.
- **ButtonPlay**: Кнопка для початку відтворення.
- **ButtonBack**: Кнопка для повернення до вибору файла.

4. Створіть об’єкт Slider дочірнім **VideoBackground**. Перейменуйте його на **SliderTimeVideo**.Зробіть об’єкти.

- TimePanel: Image для фона індекатора часу.
- Text (TMP) дочірній TimePanel: Він буде індекатором часу.

5. Створіть об’єкт Image і зробіть його дочірнім об’єктом **VideoBackground**. Перейменуйте його на PanelTextNameVideo та додайте Text (TMP). Ця панель слугуватиме для відображення назви відео.

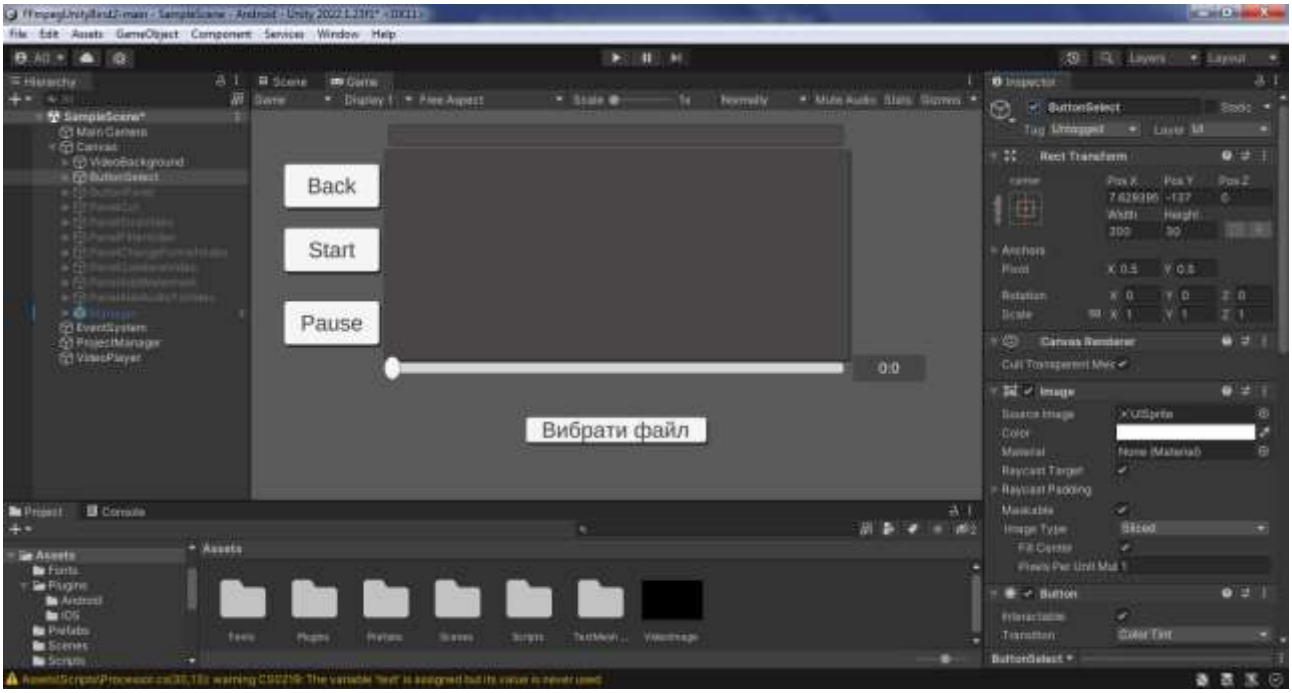

Рисунок 1.5 Фінальний вид Відеоплеєра

Розташуйте VideoBackground та VideoWindow по центру вікна програми, кнопки - ліворуч від VideoBackground, SliderTimeVideo - знизу від VideoBackground, TimePanel - зліва від SliderTimeVideo, а PanelTextNameVideo - зверху від VideoBackground. У Assets створіть Render Texture, перейменуйте на VideoImage, активуйте Dynamic Scaling та встановіть Size 720x480.

У компоненті Raw Image об’єкта VideoWindow перетягніть VideoImage в поле Texture. Створіть пустий об’єкт VideoPlayer та додайте до нього компонент VideoPlayer. У компоненті VideoPlayer в поле Target Texture перетягніть VideoImage. В папці Scripts створіть два файли: Processor.cs та VideoController.cs

VideoController.cs

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.Video;
using UnityEngine.EventSystems;
using TMPro;

public class VideoController : MonoBehaviour
{
public VideoPlayer videoPlayer; //Компонент об’єкта VideoPlayer
public TMP_Text timeText ,textNameVideo;//TimePanel та PanelTextNameVideo
```

```
public Slider timeLine;// Slider SliderTimeVideo
private float currentTime;
public Processor proces;// Скрипт Processor.cs
private int minutes,seconds;

void Update()
{
if(videoPlayer.url != "" && videoPlayer.isPrepared){
currentTime = (float)videoPlayer.time;
timeLine.value = currentTime;

minutes = Mathf.FloorToInt(currentTime / 60);
seconds = Mathf.FloorToInt(currentTime % 60);
timeText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
}
}

// В Update постійно вираховується час для PanelTime
public void OnSliderValueChanged()
{
if(videoPlayer.url != "" && videoPlayer.isPrepared){
videoPlayer.time = timeLine.value;
}
}
// OnSliderValueChanged() перемотує відео по Slider
public void SelectVideo(string adressVideo,float totalTime)
{
videoPlayer.url = adressVideo;
timeLine.maxValue = totalTime;
textNameVideo.text = adressVideo; }

//SelectVideo включає на екрані відео
}
```

Processor.cs

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

```
public class Processor : MonoBehaviour
{
private string firstInputPath;
private string directory;
private string extension;
private string outputPath;
public string secondInputPath;
public float GetLentgVideo(string path)
{
float rc = FFprobeWrapper.GetVideoDuration(path);
return rc;
}

// Функція GetLentgVideo() повертає тривалість відео.

public void SetSizeVideo(int x)
    {
//Код заглушка
    }

public void SortiredFile(string path,string typeFile)
    {
        if(typeFile == "firstVideoFile"){
        firstInputPath = path;
        directory = Path.GetDirectoryName(firstInputPath);
        extension = Path.GetExtension(firstInputPath);
        outputPath =
Path.Combine(directory,Path.GetFileNameWithoutExtension(firstInputPath));
        SetSizeVideo(0);
        }
        else{
            secondInputPath = path;
            if(typeFile == "image")
            SetSizeVideo(1);}}

/* Функція SortiredFile(), що викликається зі скрипта Manager.cs, передає інформацію про файл до скрипта Processor.cs. Ця інформація включає path (повний шлях до файлу) та typeFile (тип файлу, що вказує на його роль: відредаговане відео або додатковий матеріал).*/
```

Фактично, numberFile є ідентифікатором, який дозволяє Processor.cs розрізнити, чи отриманий файл є основним відредагованим відео, чи допоміжним ресурсом, таким як фонова музика, зображення або додаткові відеофрагменти.

Налаштування кнопок керування відтворенням відео:

- **ButtonPause:** У властивості OnClick додайте виклик функції VideoPlayer.Pause(), перетягнувши компонент VideoPlayer до поля об’єкта та вибравши відповідну функцію.
- **ButtonPlay:** У властивості OnClick додайте виклик функції VideoPlayer.Play(), перетягнувши компонент VideoPlayer до поля об’єкта та вибравши відповідну функцію.
- **ButtonBack:** У властивості OnClick додайте один виклик, перетягніть об’єкт Button_Select до поля об’єкта та виберіть функцію SetActive(true).

Налаштуйте слайдер SliderTimeVideo: додайте компонент Event Trigger з двома викликами на подію Pointer Down (перший викликає OnSliderValueChanged() з VideoController.cs, другий викликає Pause() з VideoPlayer) та одним викликом на подію Pointer Exit (викликає Play() з VideoPlayer).

Вносимо зміни в скрипт Manager.cs в функцію SelectFile().

```
public void SelectFile(int index)
   {
      if(typeFile == "firstVideoFile"){
      proces.SortiredFile(files[index].FullName,typeFile);
      videoController.SelectVideo(files[index].FullName,proces.GetLentgVideo(files[
index].FullName));
      }
      else {
         proces.SortiredFile(files[index].FullName,typeFile);
      }
      fileListPanel.SetActive(false); buttonsPanel.SetActive(true);
   }
```

Тепер, коли програма успішно знаходить, відтворює та взаємодіє з відеофайлами на телефоні, настав час розробити панель керування та реалізувати необхідні функції за допомогою плагіна FFmpeg.

3.3. Панель функції

Для кращої організації коду функції FFmpeg будуть знаходитись у класі Processor.cs, щоб уникнути перевантаження класу FFmpegWrapper. Створено пустий об’єкт PanelButton.

У функції OnClick кнопки ButtonBack додано виклик PanelButton.SetActive(false). Також, у класі manager.cs змінній buttonsPanel присвоєно об’єкт PanelButton.

Більшість із дев’яти кнопок використовуються для відображення панелей з додатковими параметрами та елементами керування.

Рисунок 3.1 – Фінальний вид панелі PanelButton.

Панелі

З метою забезпечення зручності та модульності, багато кнопок в інтерфейсі будуть відповідати за відображення/активацію різних панелей. Тому, для

підтримки чіткої структури проекту та полегшення майбутньої розробки, рекомендується описати їх функціональність та призначення на цьому етапі.

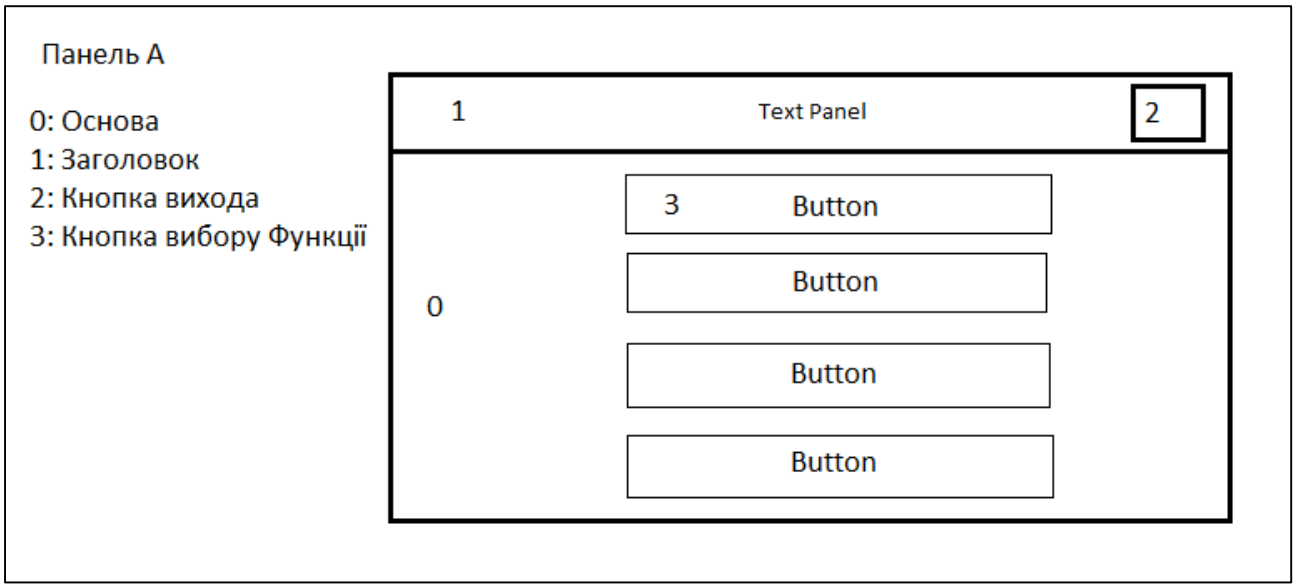

Рисунок 3.2 – Схема Панелі А

Інтерфейс складається з наступних елементів: базове зображення (“Основа”), дочірнє зображення для заголовка (“Заголовок”), кнопки виходу (“Кнопка виходу”) та кнопок вибору функції (“Кнопки вибору функції”). “Основа” є базовим контейнером. “Заголовок” візуально представляє заголовок панелі інтерфейсу.

При натисканні на “Кнопку виходу”, “Основа” закривається (ховається), а панель функцій стає знову активною (відображається). “Кнопки вибору функції” при натисканні викликають відповідну функцію, а також закривають “Основу”.

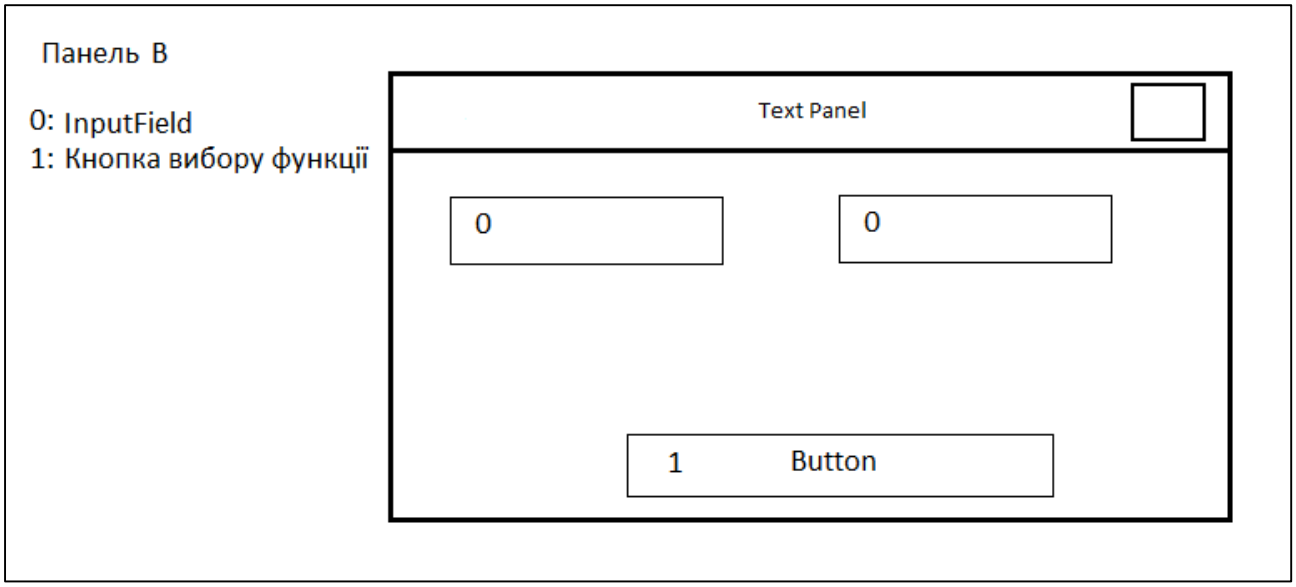

Рисунок 3.3 – Схема Панелі B

Панель B надає інтерфейс для введення числових значень, що складається з двох полів InputField, та кнопки для запуску відповідної функції обробки введених даних.

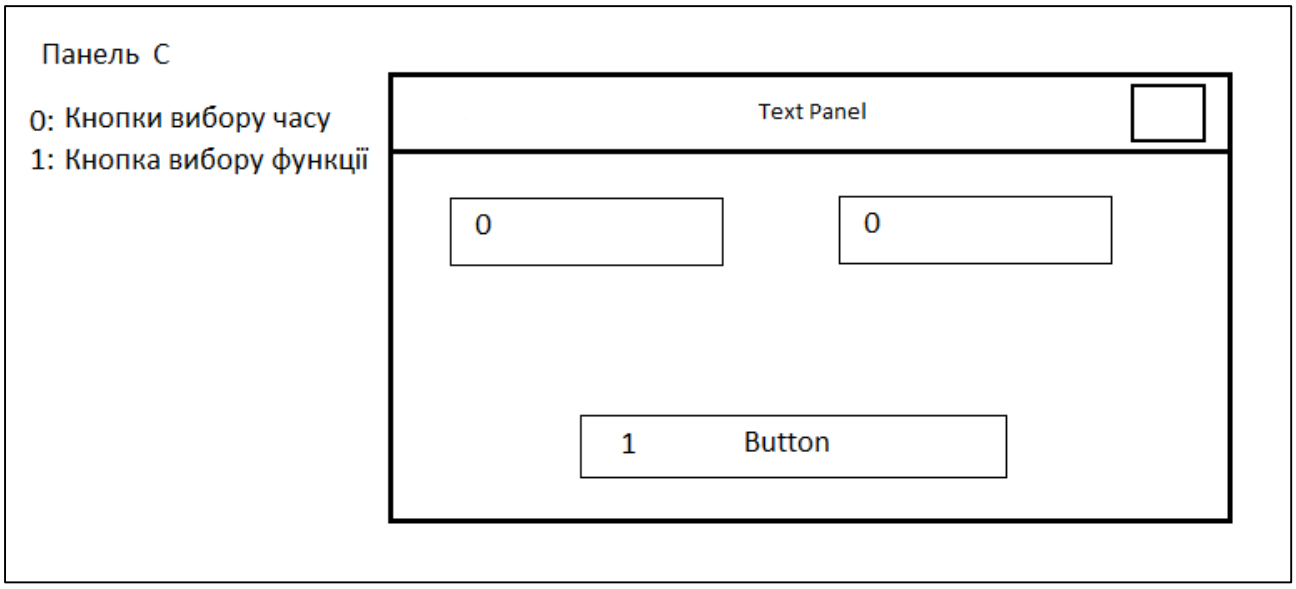

Рисунок 3.4 – Схема Панелі C

Панель C функціонально подібна до панелі B, але замість полів InputField, вона використовує Buttons для зчитування позицій у відео. Користувач може встановлювати мітки на слайдері, а потім натиснути кнопку, щоб передати ці мітки у функцію для вирізання відповідного фрагменту відео.

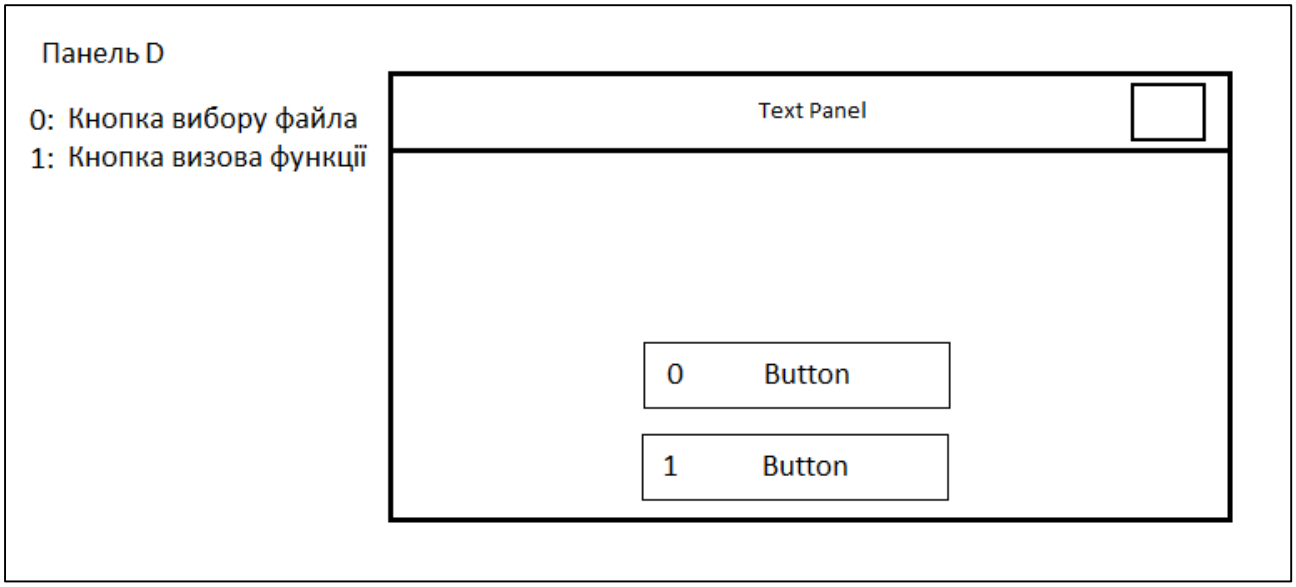

Рисунок 3.5 – Схема Панелі D

Панель D містить дві кнопки: кнопка 0 для вибору файлу (що також активує кнопку 1), а кнопка 1 для запуску функції.

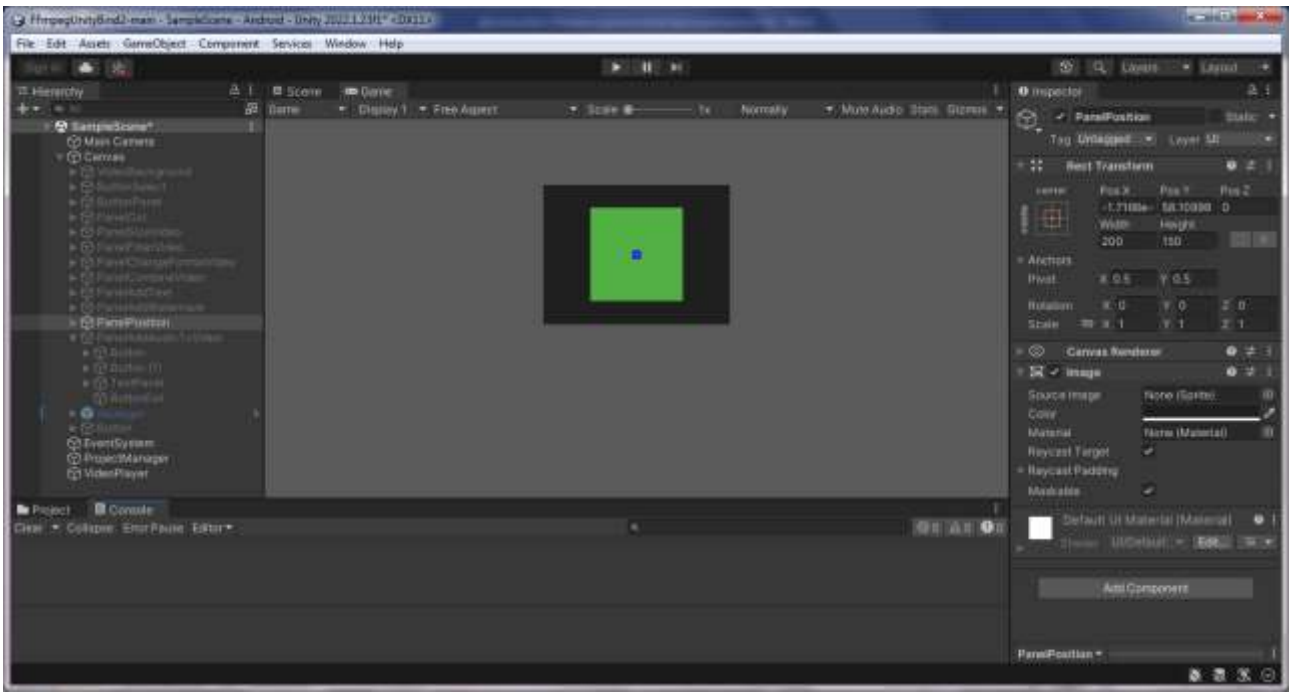

Рисунок 3.6 – Підпанель

Для реалізації функціоналу додавання зображення та тексту до відео, необхідно створити підпанель. На підпанелі розмістити два елементи Image:

imagefilefirst: Більший елемент Image (100x100), забарвлений у зелений колір. Він візуально представляє основне відео та слугує для вибору місця накладення зображення/тексту на відео.

imagefilesecond: Менший елемент Image (10x10), забарвлений у синій колір. Він відіграє роль “точки прив’язки” для тексту та є зменшеною копією зображення, яке буде накладено на відео. Позицію цього елемента визначає місце розташування тексту/зображення на відео.

Функції FFmpeg

Тепер можна розпочинати додавання функцій до класу Processor.cs.

Processor.cs

```
public void FormatChange(string x)
{
string fileName = Path.GetFileNameWithoutExtension(firstInputPath);
string outputPath = Path.Combine(directory,fileName + x);
string iPath = Path.Combine(directory,fileName + extension);
int rc = FFmpegWrapper.Execute(string.Format("-i {0} {1}",iPath,outputPath));
Debug.Log("Return Code is " + rc);
}
```

Функція FormatChange(string x): Додана до класу Processor.cs. Вона відповідає за зміну розширення відео (наприклад, з .mp4 на .mov).

Кнопка ButtonChangeFormatVideo: Створена як дочірній об’єкт для PanelButton.

Панель Image PanelChangeFormatVideo: Створена панель типа A для розміщення елементів керування зміною формату. Містить:

П’ять кнопок для вибору нового формату (з розширеннями: .mp4, .mov, .mkv, .avi, .gif). Кожній кнопці призначено виклик функції FormatChange(string x) з відповідним розширенням.

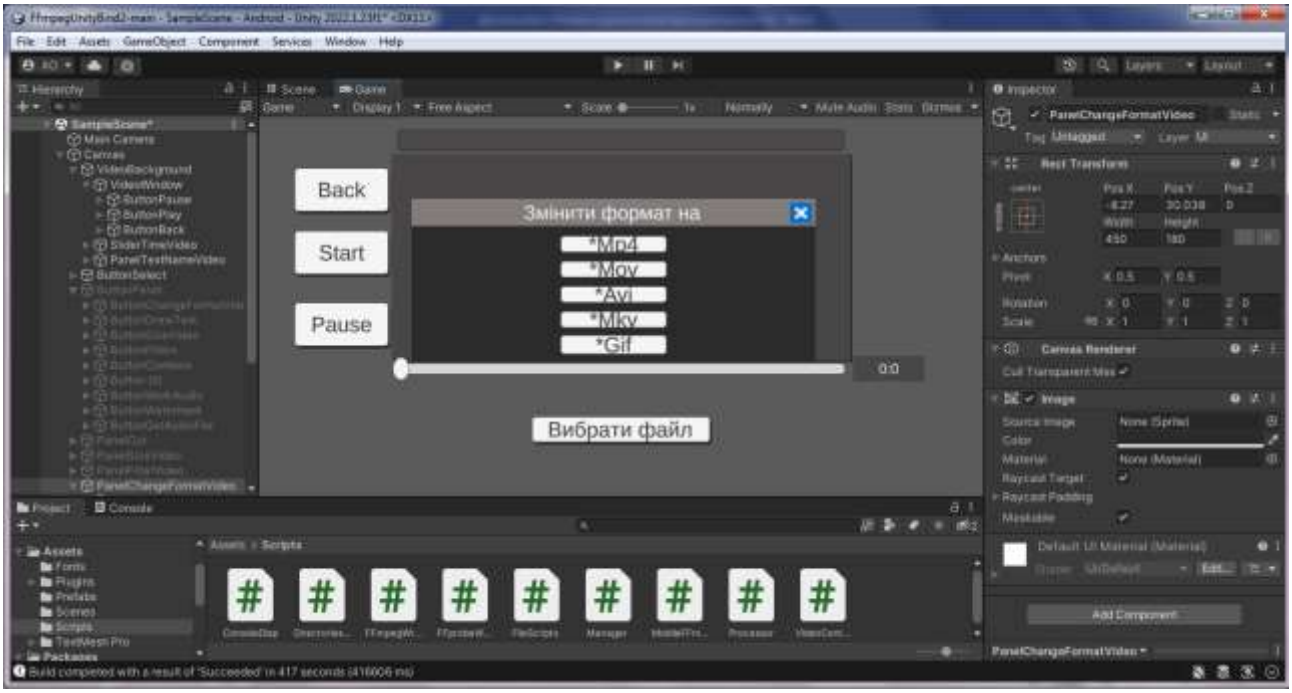

Рисунок 3.7 - Панель зміни формату

Функція SelectExtension(string x) змінює довжину та ширину відео. Метод SelectSizeVideo() зчитує дані з двох полів введення та передає рядок у функцію SelectExtension(string x).Створить панель В.

VideoController.cs

```
public TMP_InputField inputSizeFirst,inputSizeSecond;

public void SelectSizeVideo()
{
string x = inputSizeFirst.text +":"+inputSizeSecond.text;
proces.SelectExtension(x);
}
```

Processor.cs

```
public void SelectExtension(string x)
{
outputPath = outputPath+"resize"+extension;
int rc = FFmpegWrapper.Execute(string.Format("-i {0} -vf scale={2}
{1}",firstInputPath,outputPath,x));
Debug.Log("Return Code is " + rc);
}
```

Функція SelectSizeVideo() зчитує значення з двох текстових полів (InputField), формує з них рядок розміру відео, наприклад, “1260:560”, та передає цей рядок у функцію SelectExtension(string x).

Filters(string filterName) реалізовано функцію застосування фільтрів до відео.

Processor.cs

```
public void Filters(string filterName)
{
outputPath = outputPath+"filters"+extension;
int rc = FFmpegWrapper.Execute(string.Format("-i {0} -vf {2}
{1}",firstInputPath,outputPath,filterName));
Debug.Log("Return Code is " + rc);
}
```

Videocontroller.cs

```
public void FilterSaturation(Slider x)
{
proces.Filters(string.Format("'hue=s={0}*0.1'",x.value));
}
public void FilterShade(Slider x)
{
proces.Filters(string.Format("'hue=h={0}'",x.value));
}
```

Панель A містить елементи керування для застосування відеофільтрів. У верхній частині панелі розташовані три кнопки, кожна з яких викликає функцію Filters(string filterName) з одним із наступних аргументів: 'gaussianblur=sigma=5' (застосування розмиття Гауса), 'negate' (інвертування кольорів), або 'hflip' (горизонтальне відображення). Збоку розміщені два слайдери, один з яких контролює насиченість (за допомогою функції FilterSaturation(Slider x)), а інший - відтінок (за допомогою функції FilterShade(Slider x)).

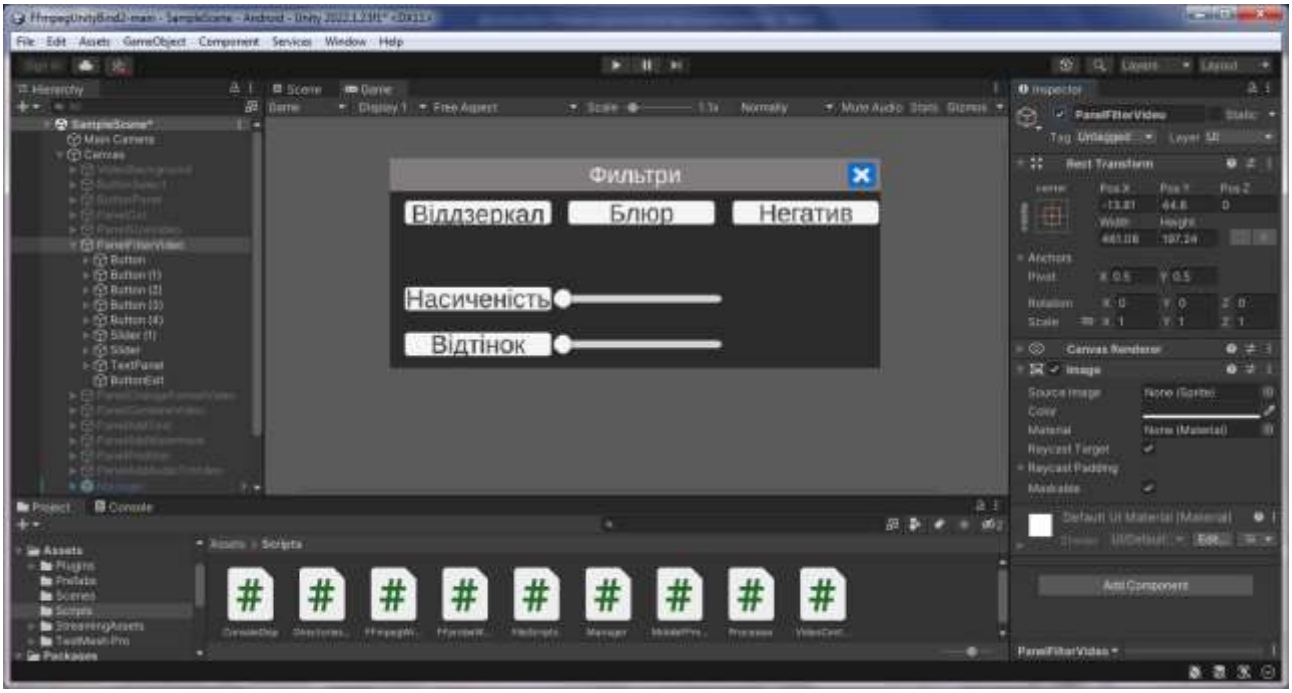

Рисунок 3.8 - Фінальний вид панелі

Для **функції GetAudio()**, яка видобуває аудіо з відео, створення окремої панелі не потрібне.

```
public void GetAudio()
{
 outputPath = outputPath+"audio.mp3";
 int rc = FFmpegWrapper.Execute(string.Format("-i {0} -vn  -acodec libmp3lame -ac 2 -ab 160k -ar 44100 {1}",firstInputPath,outputPath));
 Debug.Log("Return Code is " + rc);
 }
```

Функція CutVideo() вирізає фрагмент з відео но треба змінити Videocontroller.cs.

Processor.cs

```
public void CutVideo(float x, float y)
{
float timeLenght = y - x;
string timeStringLenght =
TimeSpan.FromSeconds(timeLenght).ToString(@"h\:mm\:ss\.fff");
string timeString = TimeSpan.FromSeconds(x).ToString(@"h\:mm\:ss\.fff");
outputPath = outputPath+"Cut"+extension;
int rc = FFmpegWrapper.Execute(string.Format("-i {0} -ss {1} -t {2}
```

```
{3}",firstInputPath,timeString,timeStringLenght,outputPath));
Debug.Log("Return Code is " + rc);}
```

Videocontroller.cs

```
public void SelectTimeCutFirst()
{
timeCutFirst = currentTime;
CutTextFirst.text = string.Format("{0:00}:{1:00}", minutes, seconds);
}

public void SelectTimeCutSecond()
{
timeCutSecond = currentTime;
CutTextSecond.text = string.Format("{0:00}:{1:00}", minutes, seconds);
}
public void SetTimeCut()
{
proces.CutVideo(timeCutFirst, timeCutSecond);
}
```

Створено панель C з трьома основними кнопками: перша для функції SelectTimeCutFirst(), друга для SelectTimeCutSecond(), а третя для SetTimeCut().

Функція AddAudioToVideo() заменяет оригинальное аудіо на вибраное.

```
public void AddAudioToVideo()
{
if(secondInputPath != ""){
outputPath = outputPath+"audioComplanet"+extension;
int rc = FFmpegWrapper.Execute(string.Format("-i {0} -i {1} -map 0:v -map 1:a -c:v
copy -c:a copy -shortest {2}", firstInputPath,secondInputPath,outputPath));
Debug.Log("Return Code is " + rc);
secondInputPath = "";
}}
```

для цієї функції потрібно створити панель D .

Функція AddWatermark() додає зображення на відео. Її можна використовувати для брендування відеоматеріалів, захисту авторських прав або просто для додавання інформації (наприклад, логотипу каналу) до відео.

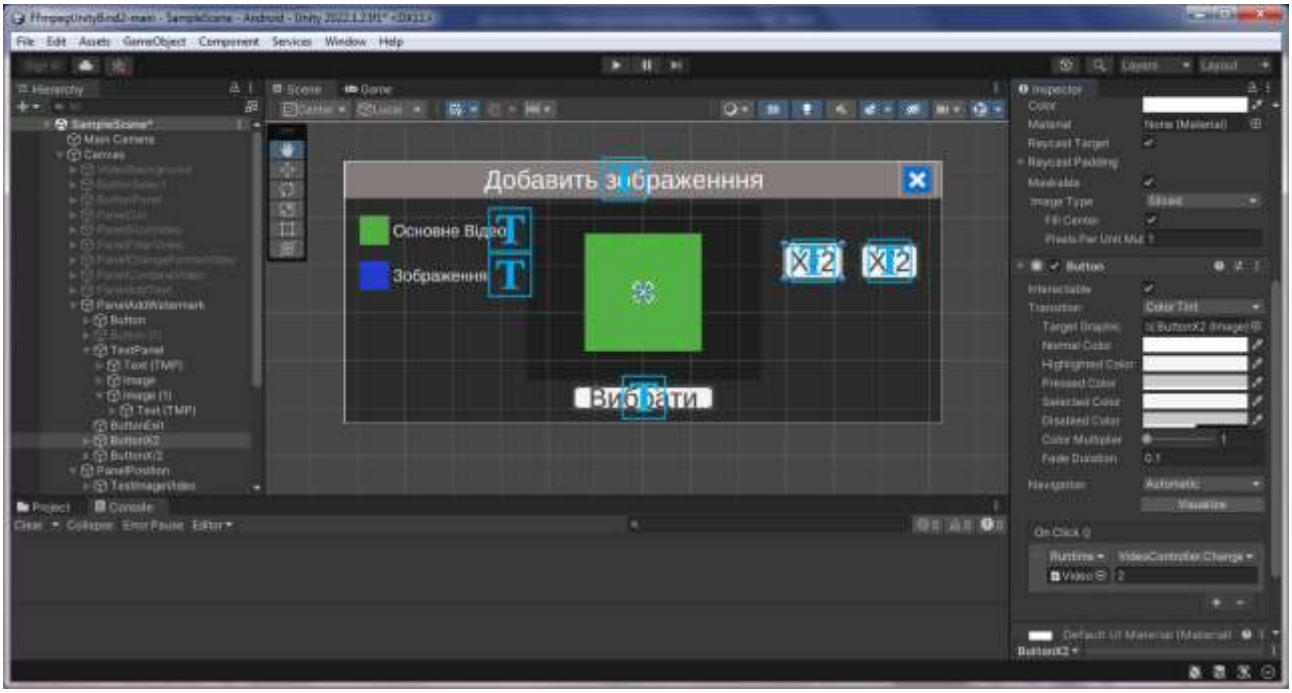

Рисунок 3.9 - Панель D з підпанелю та доповненням.

Processor.cs

```
public void AddWatemark(float x,float y,float size)
{
 if(secondInputPath != ""){
 outputPath = outputPath+"watermarka"+extension;
 x=x*8;
 y=y*-8;
 int rc = FFmpegWrapper.Execute(string.Format(
 System.Globalization.CultureInfo.InvariantCulture,
  "-i {0} -i {1} " +
    "-filter_complex \"[1:v]scale=iw*{5}:ih*{5}[wm];[0:v][wm]overlay=x=(main_w-
overlay_w)/2 + {3}:y=(main_h-overlay_h)/2 + {4}\" " +
    "-codec:a copy {2}",
    firstInputPath, secondInputPath, outputPath, x, y,size));
 Debug.Log("Return Code is " + rc);
  secondInputPath = "";
 }
 }

public void SetSizeVideo(int x)
{
if(x==0)
        videoController.ChangeExampel(FFprobeWrapper.GetVideoSize(firstInputPath,
directory),x);
```

```
        else
videoController.ChangeExampel(FFprobeWrapper.GetVideoSize(secondInputPath,di
rectory),x);}
```

Videocontroller.cs

```
public RectTransform rectTransformFirst,rectTransformSecond;

private float sizeSecondFile = 1;

public void ChangeExampel(string output,int x)
{
Match match = Regex.Match(output, @"(\d+):(\d+)");
int width = int.Parse(match.Groups[1].Value);
int height = int.Parse(match.Groups[2].Value);
if(x == 0)
rectTransformFirst.sizeDelta = new Vector2(width/8, height/8);
else
rectTransformSecond.sizeDelta = new Vector2(width/8, height/8);
}
public void ChangeSizeSecondFile(float x)
{
sizeSecondFile=sizeSecondFile*x;
rectTransformSecond.sizeDelta = new Vector2(rectTransformSecond.rect.width*x,
rectTransformSecond.rect.height*x);
}

public void TransmissionPositionAndDate(int x)
{
if(x==0){
proces.AddWatemark(rectTransformSecond.localPosition.x,rectTransformSecond.loc
alPosition.y,sizeSecondFile);
rectTransformFirst.sizeDelta = new Vector2(100, 100);
rectTransformSecond.sizeDelta = new Vector2(10, 10);
}
else
proces.CreateTextVideos(inputTextFirst.text,rectTransformSecond.localPosition.x,re
ctTransformSecond.localPosition.y,color,sizeFont);
}

public void ClickFindPostion()
{
Vector2 screenPoint = Input.mousePosition;
```

```
Vector3 worldPosition;
if (RectTransformUtility.ScreenPointToWorldPointInRectangle(
            rectTransformFirst,
            screenPoint,
            Camera.main,
            out worldPosition))
        {
      rectTransformSecond.position = worldPosition;
      }}
```

Створить панель D та підпанель. Всередині панелі D розмістимо підпанель (не роблячи підпанель дочернім) та дві кнопки для зміни розміру. Кнопка “Збільшити” викликає функцію ChangeSizeSecondFile(2f), збільшуючи розмір другого файлу, а кнопка “Зменшити” викликає ChangeSizeSecondFile(0.5f), зменшуючи його розмір.

Елемент imagefilefirst (більший зелений прямокутник, що представляє основне відео) матиме прикріплений EventTrigger, який відстежуватиме кліки. При натисканні мишею на imagefilefirst викликатиметься функція ClickFindPostion(), яка обчислюватиме координати кліку та на їх основі переміщуватиме елемент rectTransformSecond (менший синій прямокутник, що представляє точку прив’язки).

Основна кнопка, що раніше запускала функцію накладення, тепер викликає функцію VideoController.TransmissionPosition(0). Ця функція в свою чергу, враховує поточні координати rectTransformSecond для визначення місця накладення зображення на відео.

Функція SetSizeVideo(int x) отримує розміри файлу за допомогою FFprobeWrapper.GetVideoSize(firstInputPath, directory) та передає їх у функцію ChangeExampel(string output, int x). Параметр x вказує, до якого елемента відноситься розмір відео: основне відео або накладене зображення.

Функція ChangeExampel(string output, int x) змінює розміри елементів інтерфейсу imagefilefirst та imagefilesecond, а саме: встановлює нові розміри, пропорційні розмірам відео, але зменшені у вісім разів для більш зручного відображення та маніпулювання на екрані.

Функція TransmissionPositionAndDate(int x) виконує роль диспетчера, який направляє дані, що стосуються позиціонування та накладання елементів, у відповідну функцію для накладання зображення або тексту. Значення параметра x визначає, в яку саме функцію будуть передані дані: для накладення зображення або для накладення тексту.

Функція CombineTwoVideo() призначена для з'єднання двох відеофайлів, що вимагає реалізації складнішої логіки порівняно з іншими функціями. Це зумовлено особливостями роботи concat, яка використовує проміжний файл для об'єднання. Додатково, перед фактичним з'єднанням, файли будуть перекодуванні для забезпечення сумісності.Створить панель D

```
public void CombineTwoVideo()
{
if(secondInputPath != ""){
outputPath = outputPath+"concat"+extension;
string filePath = Path.Combine(directory, "myFileConcat.txt");
string textFileConcat = string.Format(@"
file '{0}'
file
'{1}'",firstInputPath+"_converted"+extension,secondInputPath+"_converted"+extensi
on);
File.WriteAllText(filePath, textFileConcat);
FFmpegWrapper.Execute(string.Format("-i {0} -vf
scale="+FFprobeWrapper.GetVideoSize(firstInputPath,directory)+",fps=30 -c:v
mpeg4 -pix_fmt yuv420p -c:a aac  {1}_converted"+extension,firstInputPath,
firstInputPath));
FFmpegWrapper.Execute(string.Format("-i {0} -vf
scale="+FFprobeWrapper.GetVideoSize(firstInputPath,directory)+",fps=30 -c:v
mpeg4 -pix_fmt yuv420p -c:a aac  {1}_converted"+extension,secondInputPath,
secondInputPath));
int rc = FFmpegWrapper.Execute(string.Format("-f concat -safe 0 -fflags +genpts -i
{0} -c copy {1}",filePath,outputPath));
      Debug.Log("Return Code is " + rc);
      File.Delete(filePath);
      File.Delete(firstInputPath+"_converted"+extension);
      File.Delete(secondInputPath+"_converted"+extension);
      secondInputPath = "";}}
```

Функція CreateTextVideos(string text, float x, float y, string color, string fontSize) призначена для додавання текстових елементів до відео.

Processor.cs

```
   public string fontFileName = "BaskeroldshadowRegular.ttf";

   private string persistentFontPath;

   void Start()
   {
      string streamingAssetsFontPath =
Path.Combine(Application.streamingAssetsPath, "Fonts", fontFileName);
      persistentFontPath = Path.Combine(Application.persistentDataPath,
fontFileName);

      if (!File.Exists(persistentFontPath))
      {
         CopyFontFromStreamingAssets(streamingAssetsFontPath,
persistentFontPath);
      }
   }
   private void CopyFontFromStreamingAssets(string streamingAssetsPath, string
persistentPath)
   {
      UnityEngine.Networking.UnityWebRequest request =
UnityEngine.Networking.UnityWebRequest.Get(streamingAssetsPath);
      request.SendWebRequest();
      while (!request.isDone) { //Wait for the request to complete (Can add timeout) }
      if (request.result ==
UnityEngine.Networking.UnityWebRequest.Result.Success)
         {
            File.WriteAllBytes(persistentPath, request.downloadHandler.data);
         }}}
public void CreateTextVideos(string text,float x,float y,string color,string fontSize)
   {
      Start();
      outputPath = outputPath+"textvideo.mp4";
      x=x*8;
      y=y*-8;
      int rc = FFmpegWrapper.Execute(string.Format(
```

```
    System.Globalization.CultureInfo.InvariantCulture,
    "-i {0} " +
    "-vf \"drawtext=text="{2}":fontfile="{3}":fontcolor={6}:fontsize={7}:x=(w-
text_w)/2+{4}:y=(h-text_h)/2+{5}\" " +
    "-codec:a copy {1}",
    firstInputPath, outputPath, text, persistentFontPath, x, y,color,fontSize));
        Debug.Log("Return Code is " + rc);
    }
```

Код Start() та CopyFontFromStreamingAssets(string streamingAssetsPath, string persistentPath) гарантує, що файл шрифту буде скопійовано з папки StreamingAssets (де він зберігається разом з додатком) до папки persistentDataPath (доступної для запису), якщо він ще не існує в останній. Шрифти, що знаходяться безпосередньо в папці StreamingAssets, не можуть бути використані для динамічного завантаження, тому копіювання в persistentDataPath робить їх доступними для використання в рантаймі.

Videocontroller.cs

```
 public TMP_InputField inputSizeFirst,inputSizeSecond,inputTextFirst;
  private string sizeFont ="24";
  private string color ="white";
public void ChangeSizeFont(TMP_InputField x)
{
 sizeFont = x.text;
}

public void ChangeColorFont(string x)
{
color = x;
}
```

Створемо панель D : видаляємо кнопку вибору файлу, оскільки вона більше не потрібна. Додаємо два текстових поля (InputField): одне для введення тексту, який потрібно накласти на відео, а інше - для визначення розміру шрифту. Також на панелі з’являються вісім невеликих кнопок різних кольорів: червоного, білого, чорного, зеленого, синього, сірого, фіолетового та жовтого. Кожна з цих

кнопок викликає функцію ChangeColorFont(string x), передаючи назву відповідного кольору. Основна кнопка на панелі, що викликала функцію, тепер викликає VideoController.TransmissionPosition(0). Кнопка виклику панелі при натисканні відображає як саму панель, так і її підпанель.

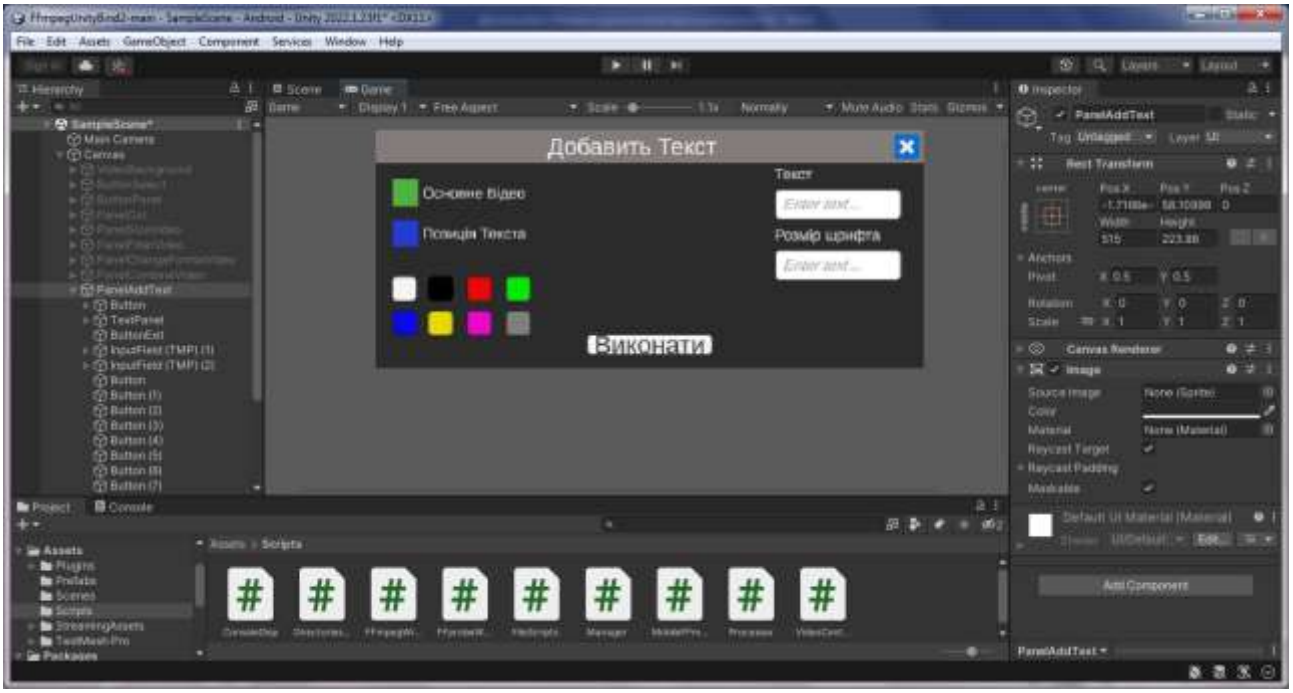

Рисунок 3.10 – Фінальний вид панелі

На завершення розробки необхідно додати кнопку для виходу з програми.

VideoController.cs

```
public void Escape ()
    {
        Application.Quit();
    }
```

Після додавання останніх рядків коду, та успішної збірки проекту під Android, наша програма офіційно завершена.

3.4 Тестування програми

Завершення розробки програми вимагає обов’язкового етапу тестування для забезпечення її надійної роботи. Ретельне тестування дозволить перевірити працездатність усіх функцій та виявити потенційні недоліки для подальшого редагування та вдосконалення.

Оскільки наша програма розроблена виключно для платформи Android, тестування проводитиметься вручну на реальних пристроях, щоб максимально точно відтворити умови використання кінцевими користувачами. Під час ручного тестування ми зосередимось на перевірці коректності роботи UI-елементів, плавності відтворення відео, точності накладання тексту та зображень, швидкості обробки фільтрів, а також стабільності програми в цілому.

Для проведення тестування я обрав відеоматеріал із зображенням сови, який має наступні характеристики: роздільна здатність 576x1024 пікселів, тривалість 1 хвилина 8 секунд та формат файлу MP4.

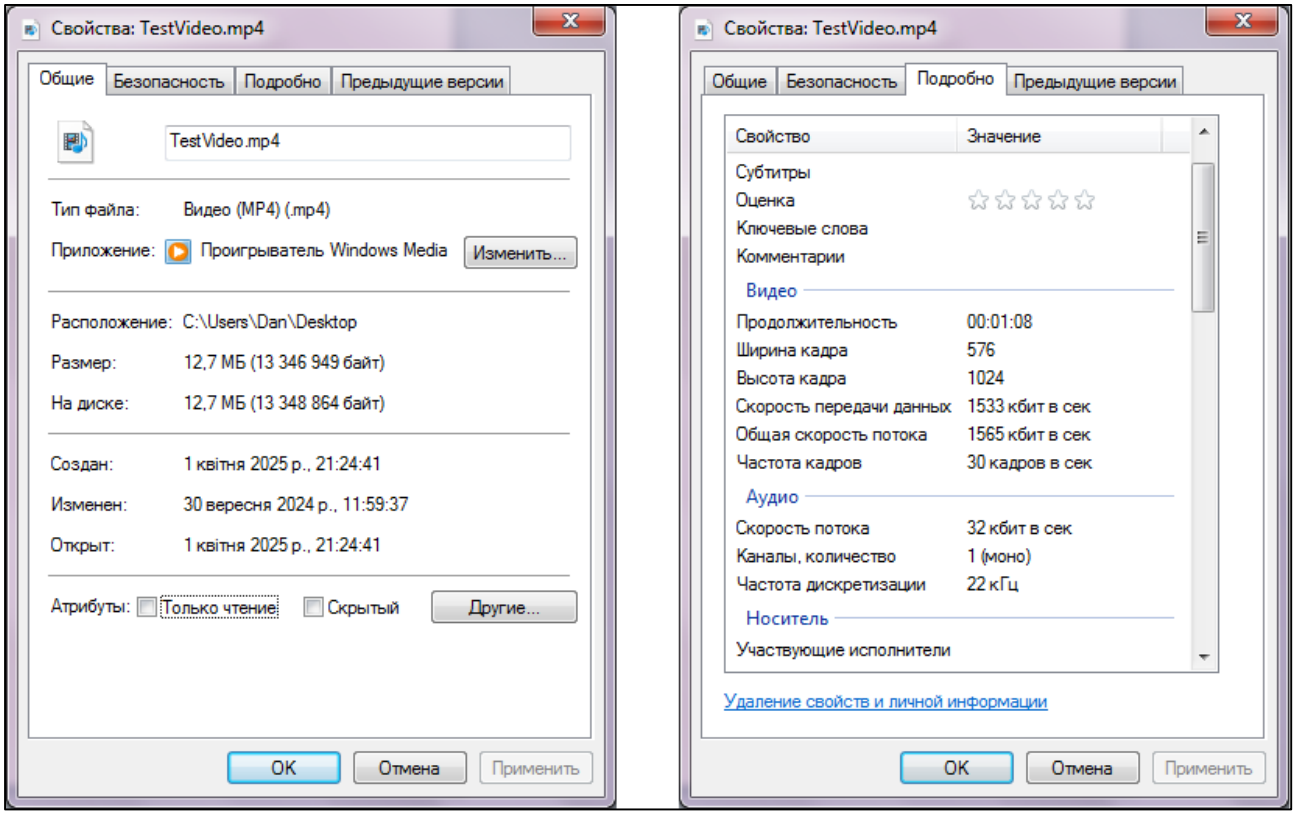

Рисунок 3.11 – Властивості основного файлу.

На першому етапі тестування перевіряємо функціональність зміни формату відео, конвертуючи файли з MP4 у MOV, AVI, MKV, GIF та навпаки, з MOV

в MP4.

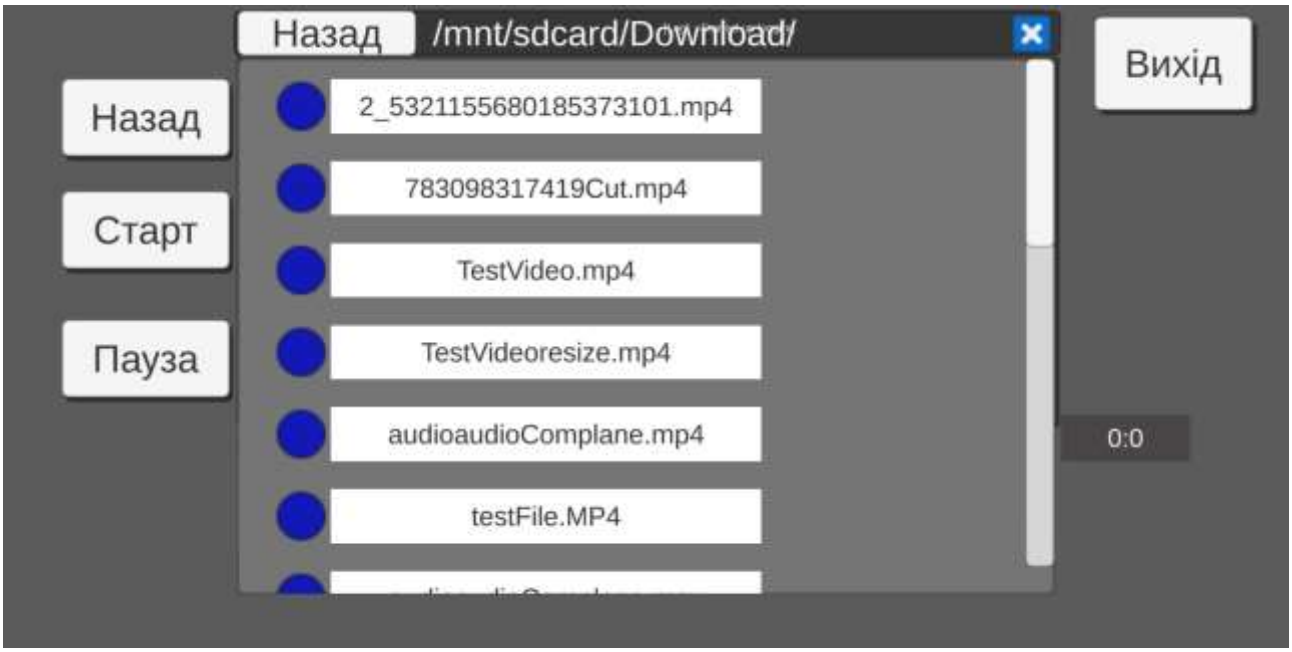

Рисунок 3.12 – Лист файлів

Процес тестування зміни формату відбувався наступним чином: спочатку натискаємо кнопку “Вибрати файл”, у вікні вибору файлу обираємо тестовий файл “TestVideo.mp4”. Після завантаження файлу з’являється панель з доступними функціями, де ми обираємо “Зміна формату”.

Відкривається панель з п’ятьма кнопками, що відповідають різним форматам.

По черзі натискаємо кнопки *mov, *avi, *mkv та *gif для конвертації в відповідні формати. На завершення, вибираємо “TestVideo.mov” для конвертації назад у формат MP4.

Після конвертації було виявлено, що окрім формату, змінився також і розмір файлів.

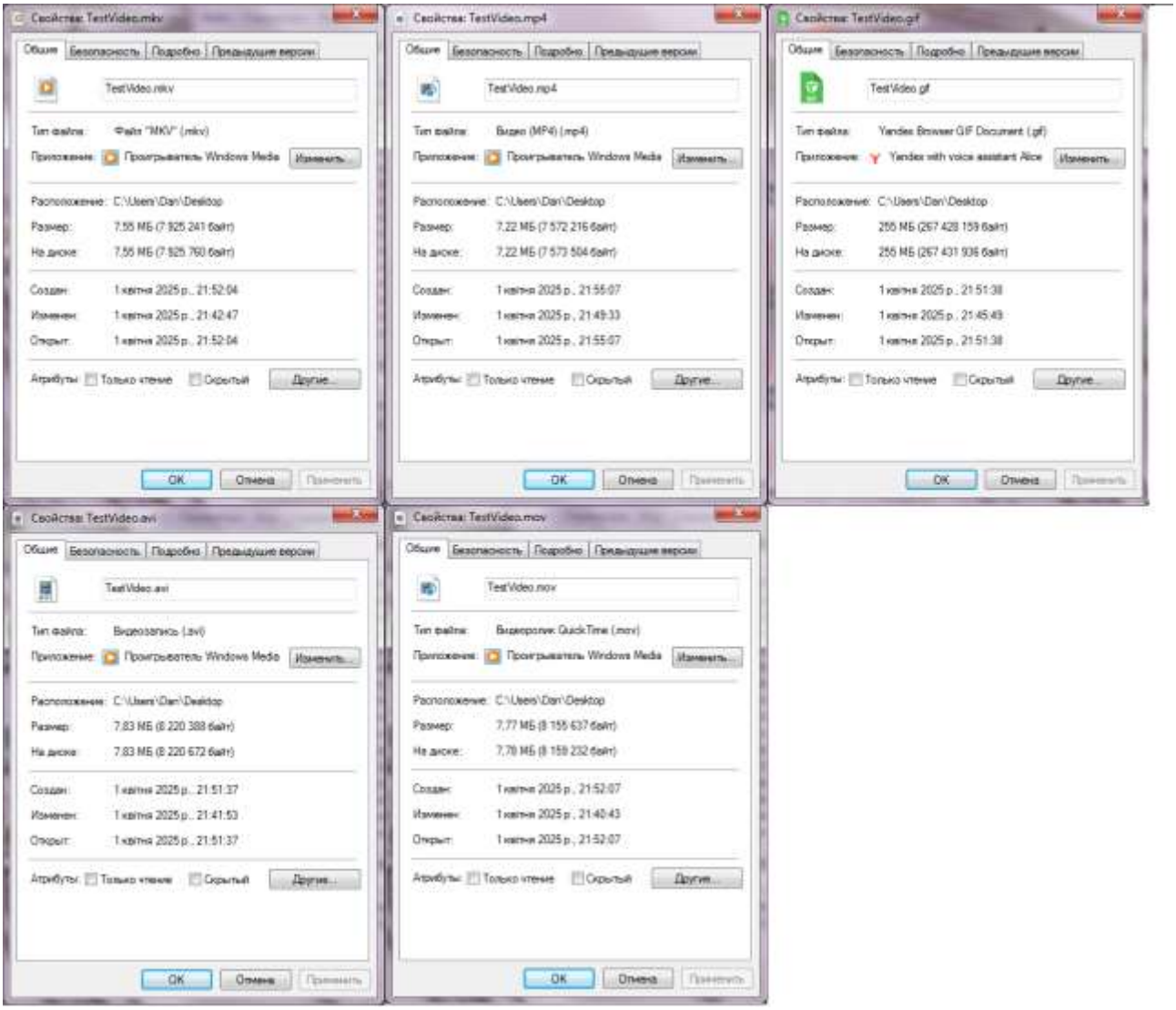

Рисунок 3.13 – Властивість змінених файлів.

Результати конвертації файлу MP4 розміром 12.7 МБ в різні формати показали наступні зміни в розмірі: MKV - 7.55 МБ, GIF - 255 МБ, AVI - 7.83 МБ, MOV - 7.77 МБ. При конвертації MOV назад у MP4 отримали файл розміром 7.22 МБ.

Далі перевіряємо функціональність зміни роздільної здатності відео. Знову вибираємо файл, натискаємо кнопку “Змінити розмір”, після чого з’являється панель з двома полями для введення нових значень ширини та висоти. Вводимо значення 565 та 780 у відповідні поля, та натискаємо кнопку підтвердження. В результаті отримуємо файл з новою роздільною здатністю 565x780.

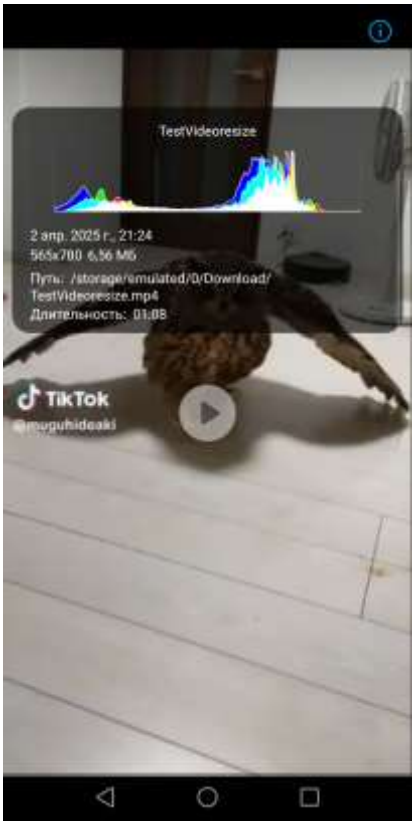

Рисунок 3.14 – Результат зміни розміру

Для перевірки функції вирізання фрагмента відео, обираємо відповідний пункт меню “Вирізати”. З’являється панель з елементами керування для визначення початку та кінця фрагмента. Пересуваємо слайдер до бажаного початку фрагмента та натискаємо першу кнопку, щоб встановити відповідну позначку. Далі пересуваємо слайдер до бажаного кінця фрагмента та натискаємо другу кнопку для фіксації кінцевої точки. Перевіряємо встановлені межі фрагмента та натискаємо третю кнопку, щоб запустити процес вирізання. Я вибрав 15 секундний відрізок.

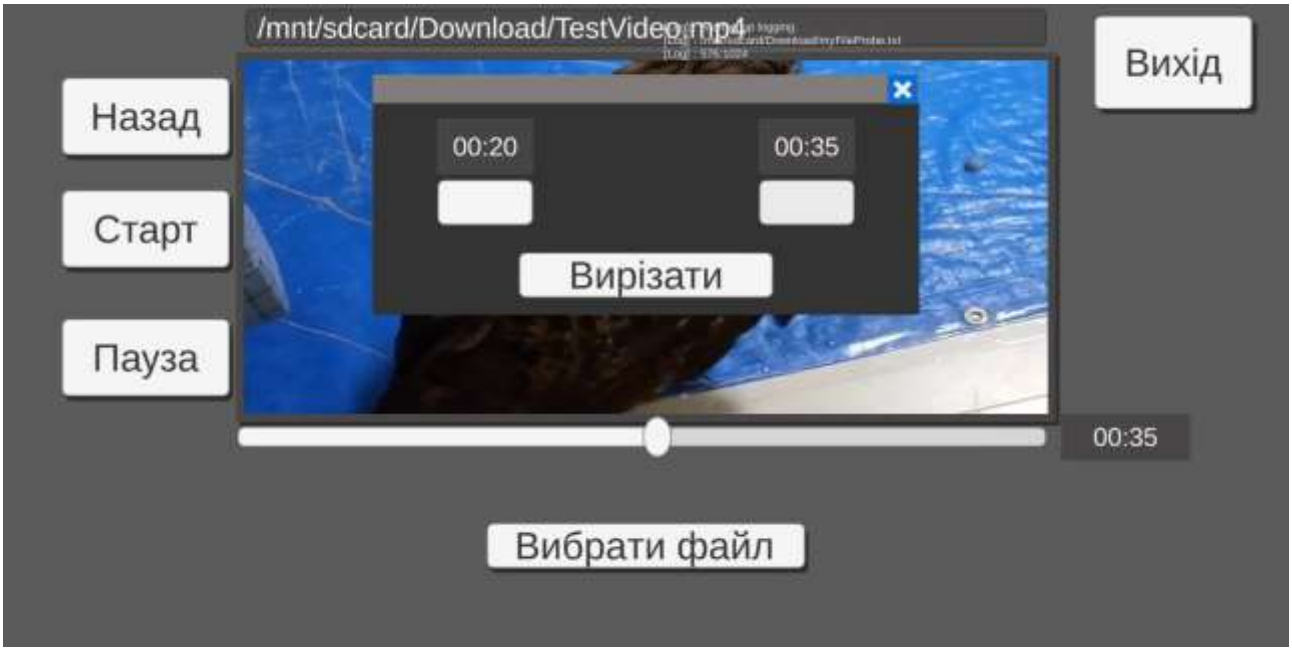

Рисунок 3.15 – Панель «Вирізання»

Переходимо до тестування функції об’єднання файлів. Після вибору першого файлу, натискаємо кнопку “Об’єднати”, щоб відкрити панель для вибору другого файлу. Обираємо другий файл зі списку та натискаємо кнопку “Виконати” для запуску процесу об’єднання. В результаті успішного об’єднання файлів, створено новий файл “TestVideoconcat” загальною тривалістю 1 хвилина 24 секунди, що відповідає сумі тривалості вихідного файлу (1 хвилина 8 секунд) та вирізаного фрагмента “TestVideoCut” (15 секунд).

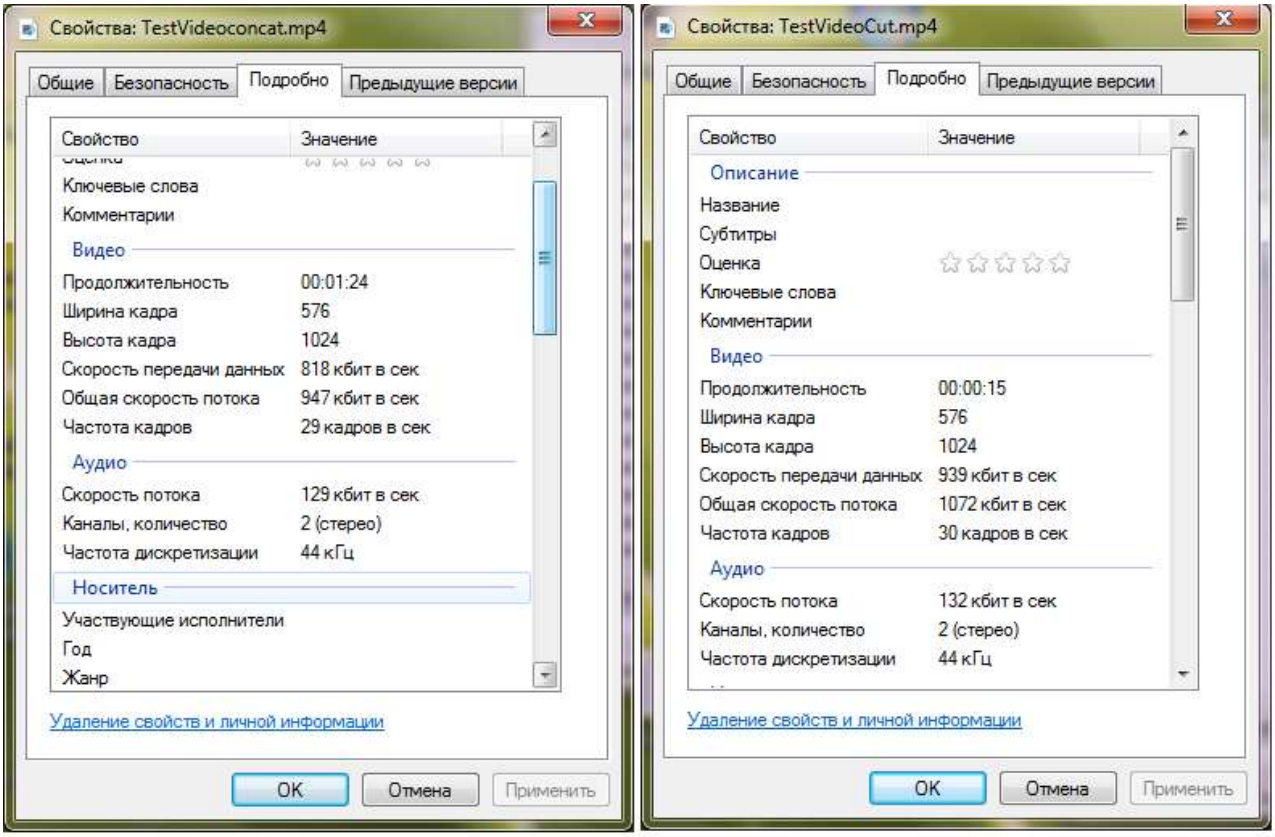

Рисунок 3.16 – Властивість вирізаного фрагмента та кінцевий результат

Наступним кроком тестуємо функціональність роботи зі звуком. Спочатку обираємо файл “TestVideo.mp4” і натискаємо кнопку “Витягнути аудіо”, в результаті чого створюється аудіофайл “TestVideoAudio.mp3”. Потім, перевіряємо функцію додавання аудіо. Обираємо той самий файл “TestVideo.mp4” та натискаємо “Замінити аудіо”. Вказуємо файл “audio.mp3” як новий аудіо супровід. В результаті отримуємо файл “TestVideoAudioComplect.mp4”, але тривалість відео змінилась і стала 1 хвилиною, як і тривалість файлу “audio.mp3”, а не початкові 1 хвилина 8 секунд. Це свідчить про те, що тривалість відео була змінена відповідно до тривалості доданого аудіо, а не збережена.

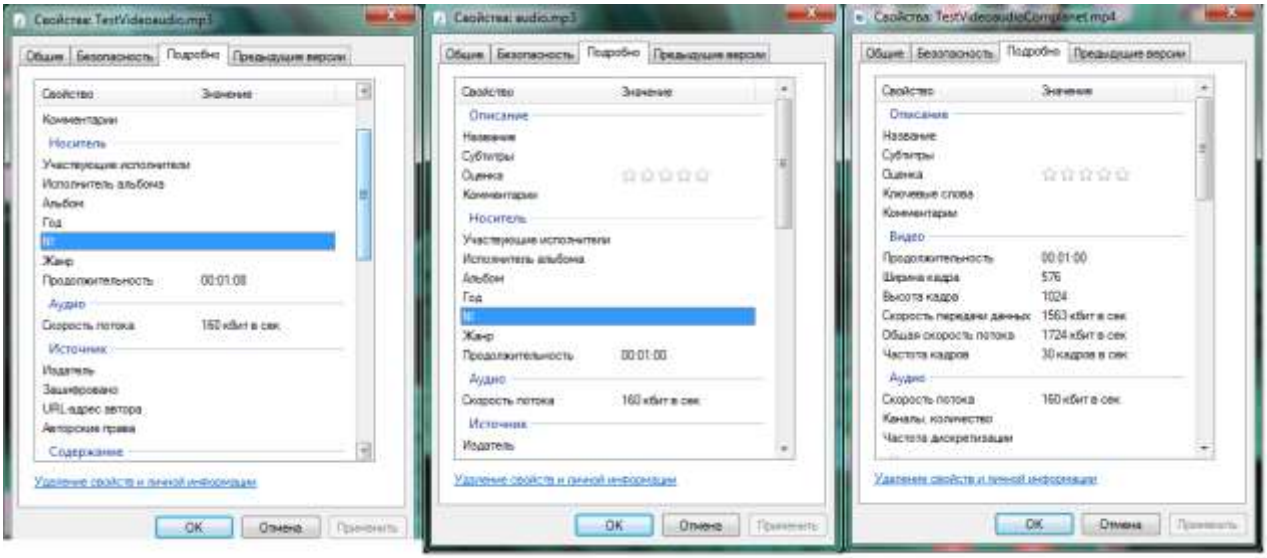

Рисунок 3.17 – Властивість файлів

Переходимо до перевірки функцій вставки елементів. Спершу, обравши відеофайл, застосовуємо функцію додавання водяного знаку. В якості водяного знаку використано чорний квадрат розміром 100x100 пікселів, який розміщено у верхній частині відео. Розмір водяного знаку збільшено вдвічі, а потім ще раз вдвічі, досягнувши 400x400 пікселів.

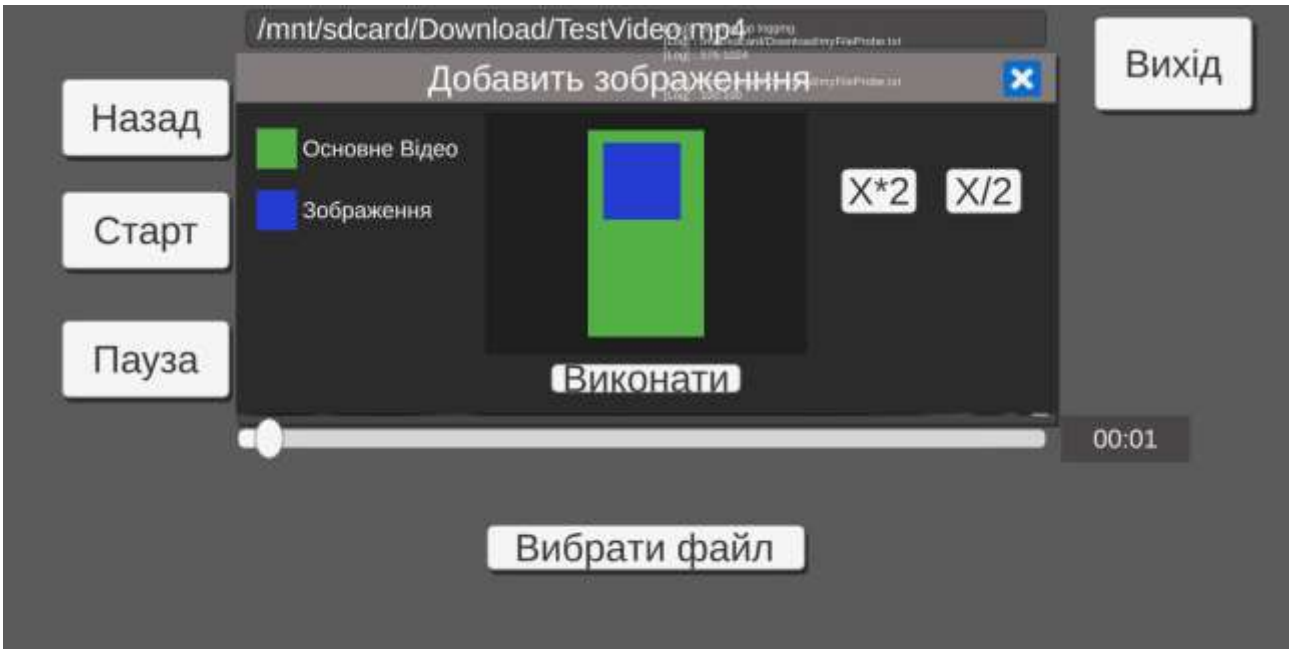

Рисунок 3.18 – Панель розміщення зображення.

Після підтвердження операції виконуємо наступну функцію – додавання тексту. Вибираємо синій колір тексту, вводимо текст “Hello world” розміром 26, та розміщуємо його приблизно в області квадрата водяного знаку.

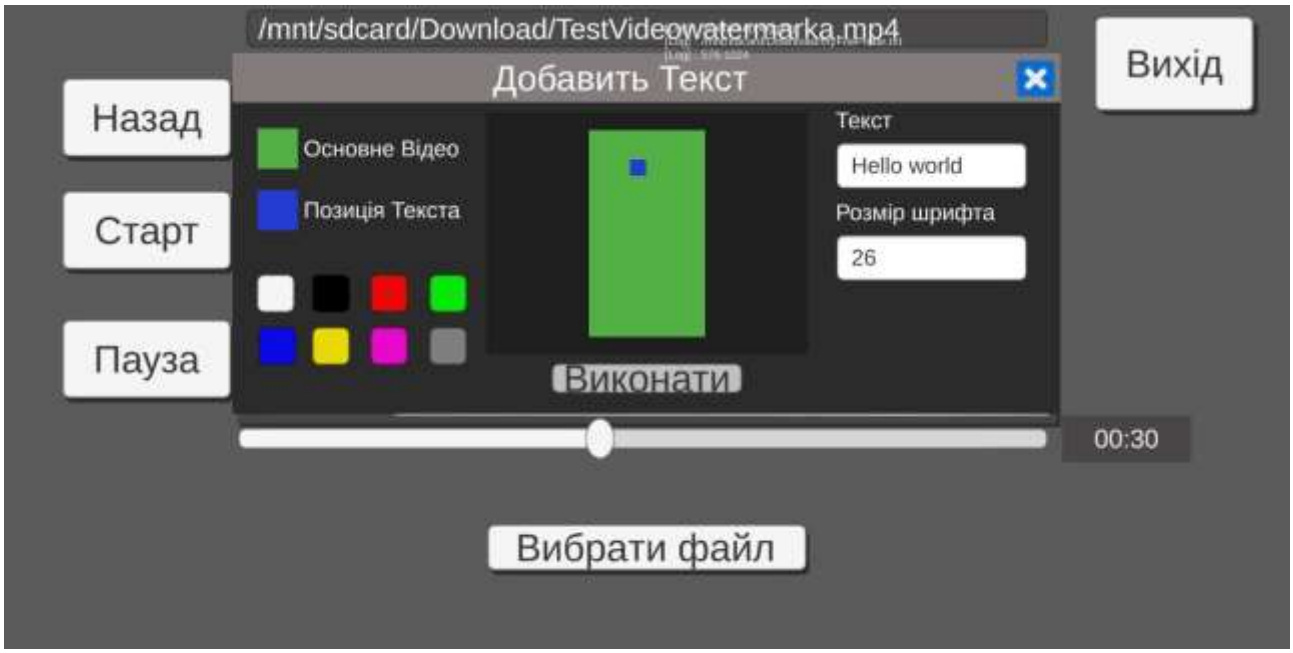

Рисунок 3.19 – Панель в дії

Результат застосування обох функцій зафіксовано на скриншоті.

Рисунок 3.20 – Результат застосовування функції.

На завершення вирішено протестувати застосування фільтрів. Спочатку обираємо відеофайл, а потім по черзі застосовуємо доступні фільтри. Першим фільтром, який перевіряємо, є “Віддзеркалити”, в результаті застосування якого отримуємо відео, віддзеркалене по горизонталі. Далі послідовно тестуємо фільтри “Негатив”, “Насиченість” та “Відтінок”.

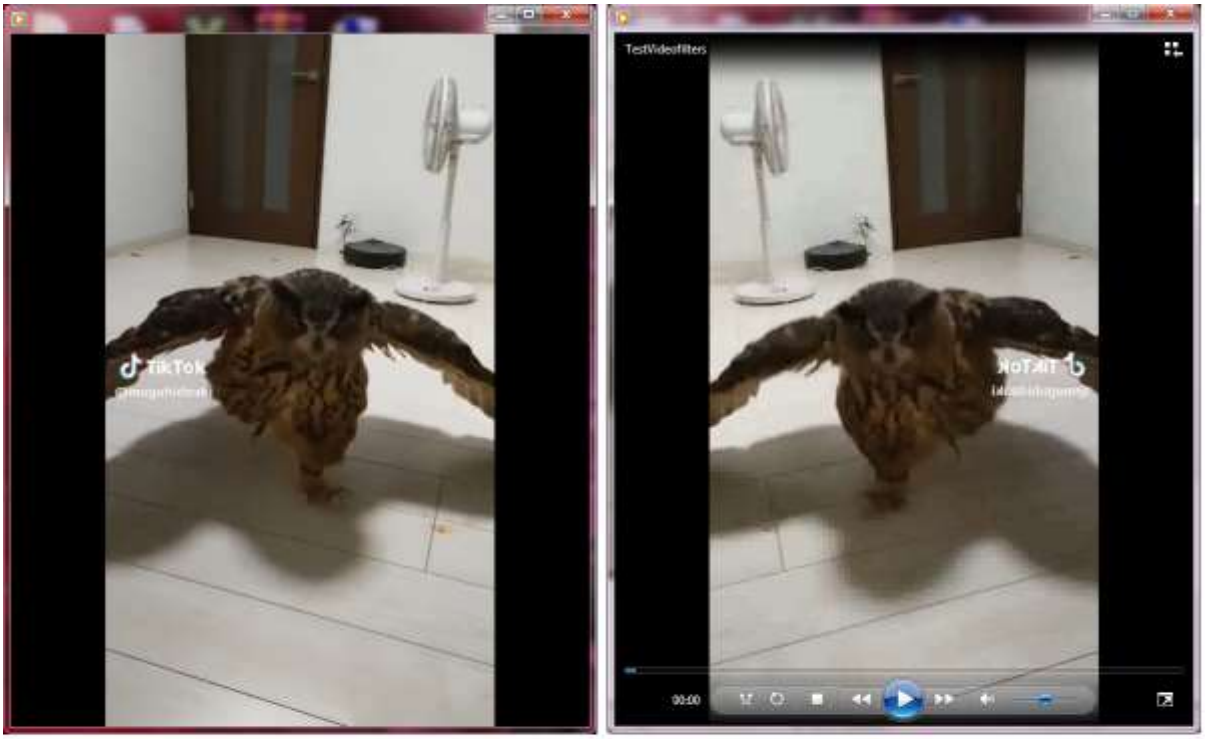

Рисунок 3.21 – Результат відзеркалення

Рисунок 3.21 – Результат негатив

При застосуванні фільтру насиченості, якщо встановити слайдер у мінімальне положення, відео стає чорно-білим.

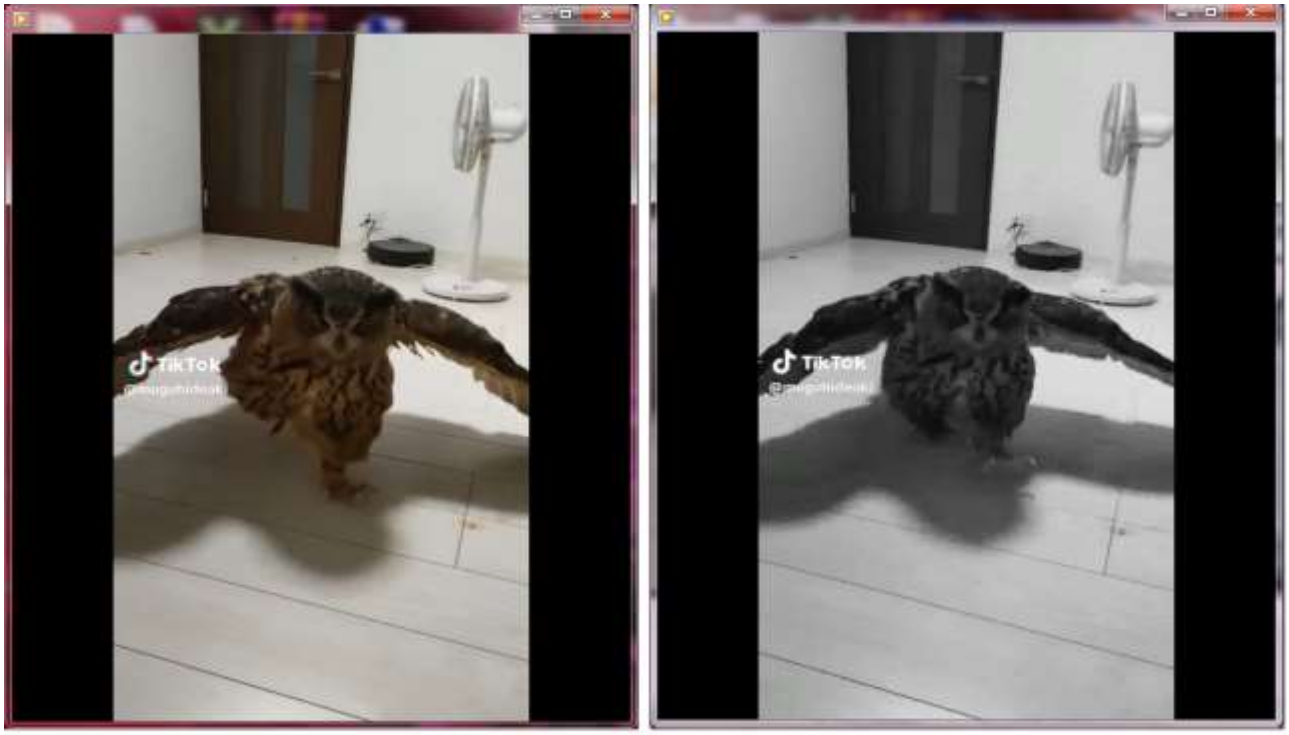

Рисунок 3.22 – Результат насиченості.

При застосуванні фільтру відтінку, встановлення слайдера у середнє положення призводить до накладання синього кольору на відео.

Рисунок 3.22 – Результат відтінку.

За допомогою данного тестування, ми виявили, що всі функції даної програми працюють справно.

ВИСНОВОК

У процесі розробки кваліфікаційної роботи було проведено ретельне дослідження різноманітних методів та технологій програмування. Вибір не найефективнішого підходу з точки зору продуктивності був свідомим рішенням, що дало можливість отримати цінний досвід у розробці програмного забезпечення, а також розширити знання та практичні навички у різних аспектах інформаційних технологій. Такий підхід сприяв глибшому розумінню тонкощів розробки та дозволив освоїти широкий спектр інструментів та технологій.

Результатом проведеної роботи є функціональний відеоредактор, розроблений для операційної системи Android. Він оснащений широким спектром функцій, що забезпечують зручне та інтуїтивно зрозуміле користувацьке середовище.

До функціональних особливостей відеоредактора належать: вбудований файловий менеджер для зручного керування відеофайлами, а також набір базових інструментів для редагування, що включають в себе обрізку, склеювання, додавання ефектів та корекцію кольору. Завдяки своїй простоті у використанні та широкому набору можливостей, даний відеоредактор може бути корисним як для професійних потреб, так і для особистого використання, полегшуючи процес обробки, редагування та перегляду графічного контенту.

У подальших планах розробки передбачено значне розширення функціоналу програми, що включатиме додавання нових ефектів, інструментів редагування та можливостей імпорту/експорту.

Крім того, планується суттєве покращення інтерфейсу користувача, з метою створення більш привабливого та інтуїтивно зрозумілого дизайну, що полегшить роботу з додатком.

Важливим етапом стане розробка версії для платформи iOS (iPhone), що значно розширить аудиторію користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FFmpeg Filters Documetion. https://ffmpeg.org/ffmpeg-filters.html
2. GitHub - FFmpeg/FFmpeg: Mirror of https://git.ffmpeg.org/ffmpeg.git. https://github.com/FFmpeg/FFmpeg
3. Use Mobile-FFmpeg with Unity - Source Example. https://sourceexample.com/en/671cf50fddfec13eb5e1/
4. FFmpeg — Вікіпедія. https://uk.wikipedia.org/wiki/FFmpeg
5. Unity Game Engine. https://unity.com/ua
6. Unity Documentation. https://docs.unity3d.com/Manual/index.html
7. YouTube — Вікіпедія. https://uk.wikipedia.org/wiki/YouTube
8. TikTok — Вікіпедія. https://uk.wikipedia.org/wiki/TikTok
9. Як користуватися ffmpeg / Шпаргалка зі знань. https://stavisdev.github.io/manuals/ffmpeg/
10. mobile-ffmpeg | FFmpeg for Android, iOS and tvOS. Not maintained anymore. Superseded by FFmpegKit. https://tanersener.github.io/mobile-ffmpeg/