МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
для виконання лабораторних робіт
з навчальної дисципліни
«КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ ТА ПРОГРАМУВАННЯ»

для здобувачів вищої освіти освітнього ступеня «бакалавр»
спеціальності 141 «Електроенергетика, електротехніка та електромеханіка»
освітня програма: «Комп'ютеризоване управління енергетичними системами»,
факультет комп'ютерно-інтегрованих технологій, мехатроніки та робототехніки
кафедра робототехніки, електроенергетики та автоматизації
імені проф. Б.Б.Самотокіна

Схвалено на засіданні кафедри
робототехніки, електроенергетики та
автоматизації ім. проф. Б.Б.Самотокіна
27 серпня 2024 р.
протокол № 6

Розробник: к.т.н., доцент кафедри автоматизації та комп'ютерно-інтегрованих технологій ім. проф. Б.Б.Самотокіна Доброжанський О.О.

Житомир
2024 – 2025 н.р.

Лабораторна робота 9.

Створення віконного застосунку з GUI засобами .NET Framework, C++.

Мета роботи: Здобути навики використання технології Windows Forms при створенні програм обробки даних у середовищі розробки Microsoft Visual C++.

1. Ознайомитись із завданням на лабораторну роботу:

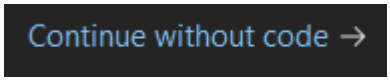
Відповідно до завдання на сьогоднішнє заняття, ми повинні створити таке програмне застосування, яке виконувало б такі функції:

- завантажувало код якої-небудь HTML сторінки
- забезпечувало можливість редагування коду HTML сторінки
- забезпечувало можливість переглядати зміни у відображенні HTML сторінки одночасно зі зміною самого коду

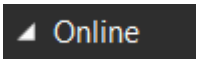
2. **Завантаження шаблону та створення нового проекту:**

!! Спосіб установки шаблону WinForms дійсна для Visual Studio 2022

-- відкрити **Visual Studio**

-- 

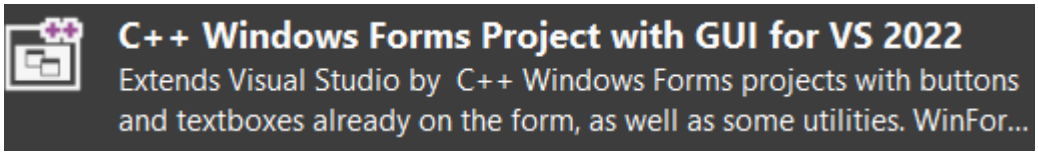
-- в головному меню -- **Exetentions** (розширення) -- обираємо **Manage Extensions** -- з'являється вікно розширень

-- у вікні **Manage Extensions** зліва обрати 

-- у вікні **Manage Extensions** -- поле пошуку (справа зверху)


-- прописуємо: **CLR**

-- обираємо зі списку, що з'являється



-- обираємо **Download**

-- далі просто закрити **Visual Studio** -- автоматично розпочнеться установка розширення -- дочекатись установки – погодитись на модифікацію та підтвердити встановлення

-- відкрити **Visual Studio**

-- почати створювати новий проєкт

-- в рядку пошуку серед типів проєктів ввести **CLR**

-- обрати тип проєкту

(також порядок встановлення розширення можливо знайти тут: https://youtu.be/IYLYX5Ei48I)

-- після створення проєкту спочатку запустити його ▶ (це іноді необхідно для налаштування зв’язків)

-- в **Solution Explorer** шукаємо файл **Form1.h**, клікаємо подвійно

-- відкривається вкладка з дизайном вікна --- видаляємо всі зайві елементи --- залишаємо віконну форму пустою

-- знову у **Solution Explorer** виділяємо **Form1.h** --- правою кнопкою мишки викликаємо випадаюче меню --- обираємо

--- видаляємо все між

```
#pragma endregion


};// end of class Form1
} // end of namespace CppCLRWinFormsProject
```

3. Виклик редактора редагування форми (конструктора форми):

Під час попередніх лабораторних робіт ми вже користувались **оглядачем рішення - Solution Explorer** . У ньому у вигляді структури-дерева відображено всі елементи проекту: основні файли коду, ресурсів тощо. У **Visual Studio** цей елемент типово зверху справа.

Знайдемо у ньому позицію **Form1.h** або щось подібне. Подвійний клік - і потрапляємо у **конструктор форми**.

4. Виклик вікна властивостей візуальних елементів форми

ПКМ --- і у **контекстному меню** обрати пункт **властивості**.

Панелька властивостей часто є виїжджаючою, але її можна закріпити інструментом за бажанням.
Прогляньте властивості форми та , наприклад, змініть **колір форми** на власний смак за допомогою редагування властивості **Back Color** на **панелі властивостей**. Тепер збільшимо розмір форми для зручності додавання обов’язкових елементів.

Ми також можемо обрати **курсор** для самої форми. Скористаємося однойменною властивістю **Cursore**.

Колір текстових елементів на формі можемо задати за допомогою властивості
Fore Color.

Тепер змінимо дещо стиль форми. Наприклад ми хочемо пласку форму у стилі, близькому до Windows 10. Задамо властивість **Form Border Style** як **None**.

Стрічка заголовку вікна зникла, тому перетягти вікно буде тимчасово не можливо. Задамо властивості форми так, щоб вона відображалася **по центру екрана**. Властивість **Start Position** задамо як **Center Screen**.

Та не захоплюйтесь, бо багато чого попереду.
Форма зовсім пуста, тому, мабуть, додамо до неї якісь елементи.

5. Виклик вікна елементів форми для додавання

Visual Studio має оригінальну виїжджаючу **панельку елементів** для додавання, подібну до такої:

У Visual Studio ця панелька зліва, у Visual Studio 2008 вона справа. Вона також може бути закріплена інструментом .

Додамо найважливіший елемент 

. Він знаходиться типово у підпункті **компоненти** на **панелі елементів**.

6. Огляд елементів наповнення форми

Почнемо з того, що видалимо елемент **error provider** виділивши його та натиснувши клавішу **del**. Помилок нам вистачить і без нього.

Відповідно до мети роботи ми повинні обов’язково розмістити такі компоненти:

Давайте поки що просто переглянемо їх спочатку, не додаючи на форму (після перегляду буде опис, як додати ці елементи):

1) Стрічка меню:

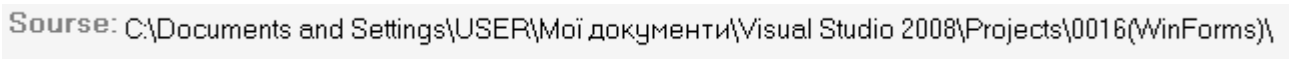
2) Підпис назви самої програми:

3) Кнопка виходу з програми 
4) Два вікна з підписами їх призначення:
   - редактор HTML джерела коду WEB сторінки
   - вікно WEB браузера для відображення змін у її коді у режимі online

5) Кнопка прийняття змін джерела коду сторінки з вікна редактора (по факту збереження коду) та виклику оновлення вікна браузера.

6) Елемент-мітка для встановлення виконання оновлення у автоматичному режимі, кожні кільканадцять секунд + елемент встановлення періоду оновлення:

7) Текстовий елемент з заголовком для відображення шляху файлу-джерела коду, який відкрито у вікні редактора коду:



(елемент є виключно інформативний)

Тепер додаватимемо елементи так, якби ми це робили у звичайному графічному редакторі.

Нагадуємо, що всі **елементи** дістаються **з панелі елементів**, яка типово згорнута на боковій стороні Visual Studio.

7. Додавання елемента меню до форми

Так цей елемент може виглядати на панелі елементів. Найцікавіше те, що елемент не розміститься явно на формі. Він з’явиться у нижній секції конструктора форми. Отже він невидимий, проте на формі з’явиться полоса:

Як тільки буде введено ім’я першого пункту меню, одразу ж з’явиться запрошення ввести наступний пункт або підпункт:

І т.д.

Сконструюємо меню з пунктів та підпунктів за структурою:

Файл
- **відкрити**
- **зберегти**
- **вийти**

Вид
- **редактор коду**
- **браузер**

Інфо
- **автор**

Якщо бажаємо, щоб у нашій програми можну було б користатись «гарячими» клавішами, то при введені імені перед літерою гарячої клавіші ставимо знак &. Наприклад, запис імені як **&файл** створить пункт меню **<u>ф</u>айл**. Тоді сполучення на клавіатурі Alt+ф викличе на виконання цей пункт меню.

Для пункту меню вид, зробимо мітки – елементи пам’яті здійснюваних користувачем майбутніх налагоджень:

Редактор коду
Браузер

Для цього задамо властивість **Checked** для кожного з двох пунктів меню як **true**.
Для визначення початкового стану цих міток відмітимо властивість **Check State** як **Checked**.

Можна запустити програму на виконання і перевірити функціональність викликів та правильність відображення.

Створимо перші програми для деяких найпростіших пунктів меню.

Подвійний клік на пункті меню **«вихід»** – і середовище переносить нас у файл опису форми, у функцію обробки **події натискання**.
Зверніть увагу, що назва функції відповідає назві елемента меню:

```
System::Void вихідToolStripMenuItem_Click(System::Object^
    sender,System::EventArgs^  e)
{
                    // сюди впишемо наші оператори
        }
```

З огляду на назви пунктів рідною мовою імена функцій містять кириличний текст.

Впишемо у якості оператора об’єктно-орієнтовану дію

```
Application::Exit();
```

Це оператор виходу з застосування.

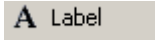
Забезпечимо програмку для пункту меню **«автор»** аналогічним чином. Передбачимо оператори обробки події натиснення як:

```
MessageBox::Show(L"Автор програми: Алєксєєв Олексій, студент групи ЕТ-10, факультету комп’ютерно-інтегрованих технологій, мехатроніки та робототехніки ДУЖП, 2027 рік");
```

Запустимо програму і перевіримо правильність.

Інші пункти працюють з елементами форми, тому перед їх програмуванням додамо всі елементи на форму.

8. Додавання елементів на форму

Всі **текстові підписи** можливо додати на форму за допомогою елемента 

Кнопка виходу з програми 

, що замінюватиме системну кнопку [X]
Вікно редактора коду HTML сторінки - це елемент 

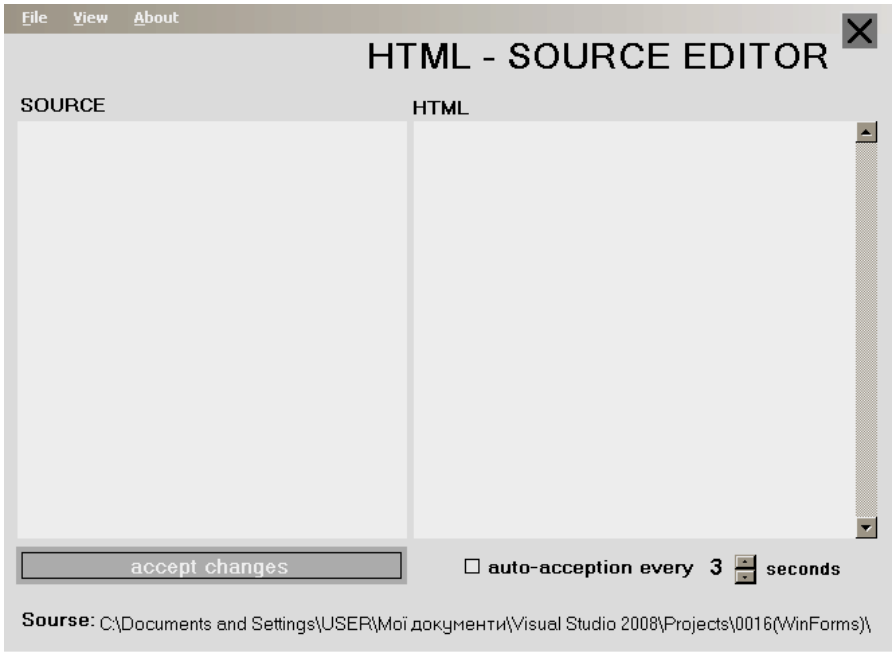
Вікно браузера 

Кнопка підтвердження редагування 

Віконце мітки для включення режиму автоматичного оновлення браузера 

Віконце зміни інтервалу оновлення 

Рядок відображення шляху до файлу коду HTML сторінки 



Після додавання елементів можна отримати щось подібне.

Віконцю 

елемента **numericUpDown** задамо початкове значення **3** у параметрі **Minimum** (відповідає мінімальному інтервалу оновлення браузера – 3 секунди). Параметри елемента можна викликати, якщо разово клікнути на елементі, тобто виділити його --- кліком правої кнопки мишки на елементі викликати меню команд --- обрати параметр властивості (**properties**).

9. Редагування властивостей доданих елементів

Шрифти елементів та кольори написів коректуються за допомогою властивостей **Font** та **Fore Color**.

Тло елементів це властивість **Back Color**.

Намалювавши у Paint картинку X, ви можете додати **файл-картинку до тла** кнопки «вихід» за допомогою властивості **Image**. Якщо це не вдається, то можна просто виставити текст «X» у полі **text** властивостей кнопки.

Плаский стиль задається властивістю або **Flat Style** як **Flat** або **Border Style** як **None**.

10.Додавання до форми службових невидимих елементів

Як і елемент головного меню, так само інші службові елементи заносяться **на нижню секцію конструктора форми**.

Це **елемент таймера** 

та **елемент файлового діалогу** 

Таймер вимірюватиме інтервали між оновленнями браузера. А файловий діалог допоможе нам обирати файл зі стандартного вікна **OpenFile Windows**.

Здійснимо єдине налагодження лише для таймера. Задамо властивість **Interval** як **3000**. Це початкове значення періоду оновлення браузера (3секунди = 3000 мілісекунд).

11.Програмування основних елементів форми

Подвійним кліком по елементах форми викликатимемо по черзі програмні функції обробки реакцій на події з цими елементами. Редагуватимемо ці функції: додаванням власного коду.

!! Після кожного додавання коду бажано перевіряти працездатність створюваної програми, запускаючи її на виконання.

!! Перед запуском радимо про всяк випадок примусово викликати збереження змін 

Опис інструкцій для кнопки [X] закриття вікна застосування (обробка події: Click)

```
// виклик функції Exit() для об’єкту Application
Application::Exit();
```

Опис інструкцій для пункту меню «відкрити» (обробка події: Click)

```
                    // виклик вікна вибору файлу
openFileDialog1->ShowDialog();
    // завантаження файлу у вікно редактора коду HTML
    // за допомогою функції LoadFile вікна редактора
    // параметри цієї функції:
    // ім'я файлу - береться з парметру FileName файлового діалогу
    // параметр PlainText - означає, що файл буде прочитано
    // як звичайний текстовий файл
            richTextBox1->LoadFile(openFileDialog1->FileName,
            RichTextBoxStreamType::PlainText);
                // завантаження шляху до того ж файлу
                // у параметр рядок адреси браузеру
                // для відкриття файлу вже як web сторінки
                // функція gcnew створює новий об'єкт
    // функція System::Uri повертає об'єкт
    // для рядку адреси
            webBrowser1->Url = gcnew System::Uri(
openFileDialog1->FileName, System::UriKind::Absolute
);
    // запис у текстовий рядок-сповіщення
    // інформації про шлях до файлу web сторінки
    textBox1->Text = openFileDialog1->FileName;
```

Перевіримо роботу програми:

- **завантажимо файл HTML сторінки, що знаходиться у папці «додаткові матеріали до л.р.7»**
- **перевіримо його відображення у вікні Редактор коду**

!! Якщо рядки відкритого файлу у редакторі коду відображаються з перенесеннями, це дещо спотворює сприйняття коду в цілому, тому знайдемо властивість **WordWrap** та змінимо її значення. Перевіримо зміни відображення вмісту HTML файлу.

Опис інструкцій для пункту меню «зберегти» (обробка події: Click)

```
                    // виклик функції файлового діалогу збереження файлу
            richTextBox1->SaveFile(openFileDialog1->FileName,
            RichTextBoxStreamType::PlainText);
                // виклик функції оновлення вмісту вікна браузера
            webBrowser1->Refresh(WebBrowserRefreshOption::Completely);
```

Для того, щоб не користуватись меню, а швидко виконати збереження файлу і оновлення змін у вікні браузеру, у нас на формі передбачена **кнопка**

«Assept changes», пропишіть **самостійно** команди для **обробки кліку** на ній. Передбачається, що вона викликатиме **збереження файлу** з кодом HTML сторінки, яка відкрита у вікні Редактора коду, а також викликатиме **оновлення** вмісту **вікна Браузер**.

Опис інструкцій для спрацювання таймера (обробка події: Tick)

```
        // аналогічно пункту меню «зберегти»
richTextBox1->SaveFile(openFileDialog1->FileName,
        RichTextBoxStreamType::PlainText);
        webBrowser1->Refresh(WebBrowserRefreshOption::Completely);

        // додатково виклик функції повернення фокусу
        // на вікно редактора коду сторінки
        richTextBox1->Focus();
```

Опис інструкцій для події проставлення користувачем відмітки [V] у елементі checkBox (обробка події: CheckedChanged) – дозвіл автоматичного режиму оновлення сторінки у браузері

```
                // запуск або зупинка таймеру
        // за результатами перевірки
                // проставлення користувачем відмітки [V]
                // інформація про це у парметрі CheckState
                // якщо проставлена відмітка CheckState буде
                // рівний ідентифікатору CheckState::Checked
        if (checkBox1->CheckState == CheckState::Checked)
                {
                                // якщо було відмічено
                                // запускаємо таймер
                                // записуємо у параметр Enabled
                                // таймеру значення true
                        timer1->Enabled = true;
                }
        else
                {
                                // інакше: галочки нема
                                // зупиняємо таймер
                        timer1->Enabled = false;
                }
```

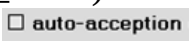
Опис інструкцій для події зміни користувачем числа у елементі numericUpDown (обробка події: ValueChanged) – збільшення/зменшення інтервалу оновлення сторінки у браузері

```
        // припиняємо роботу таймера
        timer1->Enabled = false;
```

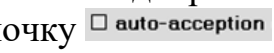
```
        // записуємо у параметр Interval таймера число
        // зчитане з елементу numericUpDown
        // * на 1000 для переведення у мілісекунди
timer1->Interval = ((int) numericUpDown1->Value)*1000;
        // запуск таймера знову з новим значенням інтервалу
timer1->Enabled = true;
```

Самостійне завдання:

Провести перевірку роботи програми:

a. Запустити форму
b. Перевірити пункт меню інформації про автора
c. Перевірити пункт меню вихід
d. За допомогою меню форми завантажити (відкрити) у вікно текстового редактора HTML коду (вікно із заголовком SOURCE) файл HTML сторінки (рекомендовано почати з HTML сторінки 2.htm, яка у архіві у папці лабораторної роботи -- !! розархівовувати рекомендовано разом з папкою файлів вебсторінки 2_files)
e. Зняти галочку 

f. У вікні текстового редактора HTML коду (вікно із заголовком SOURCE) знайти текст, що співпадає з текстом у вікні Web-браузера (вікно із заголовком HTML)
g. У вікні текстового редактора HTML коду спробувати змінити якийсь заголовок, наприклад вміст текстового тегу <h1>.
h. По натисканню на кнопку 

виконані зміни коду повинні відобразитись у вікні Web-браузера.
i. Виставити галочку 

j. Виставити інтервал оновлення коду у вікні Web-браузера:
наприклад так 

k. У вікні текстового редактора HTML коду знову змінити текст якогось тегу, наприклад, того ж тегу <h1>.
l. Чекати 3 секунди і перевірити, чи автоматично (кожні 3 секунди) вносяться зміни у вигляд HTML сторінки у вікні Web-браузера.
m. Перевірити кнопку закриття форми [X] у правому верхньому кутку форми.

12. Висновок

Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 10.

Робота з файлами зображень засобами GDI+, C++.

Мета роботи: Здобути навики використання технології GDI+ при створенні програм відображення інформації у середовищі розробки Microsoft Visual C++.

1. Ознайомитись із завданням:

Відповідно до завдання на сьогоднішнє заняття, ми повинні створити таке програмне застосування, яке виконувало б такі функції роботи з зображенням, яке містить прозоре тло.

2. Створити програмний застосунок згідно завдання:

Створення нового проекту:

Файл --- Створити --- Проект --- Visual C++ --- Windows Forms

Отже створено шаблон майже пустого проекту.
Можемо одразу ж запустити його на виконання.

Виклик редактора редагування форми (конструктора форми):

Під час попередніх лабораторних робіт ми вже користувались **оглядачем рішення**. У ньому у вигляді структури-дерева відображено всі елементи проекту: основні файли коду, ресурсів тощо. У Visual Studio цей елемент типово зверху справа.

Знайдемо у ньому позицію **Form1.h** або щось подібне. Подвійний клік - і потрапляємо у **конструктор форми**.

Виклик вікна властивостей візуальних елементів форми

ПКМ --- і у **контекстному меню** обрати пункт **властивості**.

Проглянемо властивості форми та змінимо **колір форми** на власний смак. За допомогою редагування властивості **Back Color** на **панелі властивостей**

задамо колір, що не зустрічатиметься у нас у рисунку. Рекомендується задати колір за назвою, наприклад, **Dark Magenta**.

Ми бажаємо, щоб форма була прозора під час роботи і видним було лише зображення завантаженого рисунку. Тому знайдемо властивість форми **Transparency Key** і задамо той самий колір, який обрали для тла форми.

Запустимо проект і впевнимося, що **форма прозора**.

!! Форма може вийти прозорою тільки візуально, але елементи інтерфейсу, які під формою не доступні при наведенні на них курсора (наприклад не можливо відкрити іконку файлу, який під формою **).**

!! Для створення повно-прозорої форми можна спробувати задати колір форми саме як ControlText з вкладочки System віконця вибору кольору, аналогічно задати Transparency Key для форми.

Якщо після цього **елементи закриття, згортання та максимізації форми стають недоступні**, то закрити запущену форму можна, натиснувши на  на панелі інструментів.

Тепер задамо властивість, яка автоматично розтягуватиме форму на весь екран під час виконання програми. Це властивість **Window State**. Виставимо цю властивість у значення **Maximized**.

Тепер змінимо дещо стиль форми. Наприклад, ми хочемо форму, без стиля. Навіщо нам стиль, якщо наша форма буде невидимою. Задамо властивість **Form Border Style** як **None**.

Стрічка заголовку вікна зникла, тому перетягти вікно буде не можливо. Знову ж таки, навіщо перетягати форму, яку і так не видно.

Форма зовсім пуста, але тепер можемо додати мінімальні елементи.
Додамо елемент відображення графічної інформації та елементи управління - кнопки.

У якості елемента виведення картинки додамо на форму елемент **pictureBox**. Для того, щоб цей елемент розтягнувся по всій формі, зайдемо у його властивість Dock і задамо їй значення **Fill**. Цей елемент одразу ж займе всю площу форми.

У верхній правий кут форми додамо три кнопки. Перша кнопка «згорнути», тому змінимо її властивість **Text** на **_**. Друга кнопка «закрити» – її текст можна задати як **X**. І третя кнопка «пуск/стоп» нашої програми. Задамо їй текст **GO**.

Подвійним кліком на кнопці _ відкриємо для редагування функцію обробки події кліку **buttonx_Click**. Акуратно додамо туди рядок зміни стану вікна:

Form1::WindowState::set(System::Windows::Forms::FormWindowState::Minimized);

Подвійним кліком на кнопці X відкриємо для редагування обробки події **buttonx_Click**. Додамо в цю функцію рядок виходу з програми:

Application::Exit();

Запустимо програму і перевіримо правильність.

Для того, щоб кнопки були завжди у правому верхньому куті форми, задамо властивість кнопок **Anchor** як **Top, Right**.

Витягнемо на форму також елемент **таймера** 

. Він нам буде потрібний для керування зображенням.

Приступимо до задавання локальних та глобальних змінних у основній програмі.

Знайдемо такі рядки коду:

```
....
public:

        // шукане місце

        Form1(void)
        {
                ....
```

Та додамо між **public:** та **Form1(void)** оголошення змінних:

```
static float x=100, y=100, dx=10.0, dy=10.0, sx=1.0, sy=1.0;
```

Призначення специфікатора **static** вивчимо дещо пізніше на лекціях. Змінні x та y відповідатимуть за координату, де відображатиметься картинка. Змінні **dx** та **dy** будуть слугувати елементами зміни координат x та y. І, нарешті, **sx** та **sy**. Ці змінні показуватимуть знак зміни координати **x** або **y**. Якщо **sx** та **sy** додатні 1-ці, то координати **x y** збільшуватимуться. І, навпаки, від'ємні 1-ці сприятимуть зменшенню координат **x y**.

Працюватиме формула:

```
нове значення координати = старе значення координати +
        + значення зміни координати * на знак зміни координати;
```

Як бачимо тільки-но створеним змінним надано одразу значення ініціалізації.

Додамо на рядок нижче змінні-елементи:

```
static Image^ MyImage;
        static Graphics^ graph;
```

Перша змінна-елемент зберігатиме посилання на зображення. Друга змінна зберігатиме посилання на елемент, за допомогою якого і будемо виводити графічне зображення на форму. Типами тут є **Image^** та **Graphics^**. Елементами змінними будуть **MyImage** та **graph**. Нехай не лякає така форма запису зі знаком **^.** Це ознака спеціального шаблонного середовища розробки, яке підтримує **WindowsForms**.

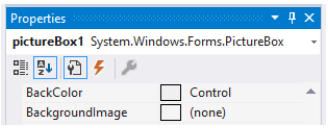
Додамо нижче також таку змінну-елемент:

```
Bitmap^ myBitmap;
```

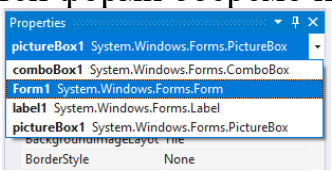
Ця змінна **myBitmap** відіграватиме роль графічного буфера. Спочатку виводитимемо рисунок у ньому, а потім, виводитимемо буфер у графічний елемент **PictureBox** на формі.

Тепер основні елементи готові і можемо запрограмувати основний алгоритм.

Зайдемо знову на форму **Form1.h** та викличемо її властивості. Клікнемо на форму, але бачимо, що відображено властивості **PictureBox1** (бо цей елемент закрив собою всю поверхню форми):



Для виклику саме властивостей форми оберемо її зі списку елементів:



Знайдемо на панельці властивостей зверху іконку з блискавкою – це перелік подій з елементом, чиї властивості переглядаються.

Знайдемо для форми властивість load. Заповнимо поряд комірку назвою функції, що запуститься, коли форма завантажуватиметься. Задамо

назву **Form1_load**. У вікні редактора коду задамо бажані рядки ініціалізації. Отже у функцію **Form1_load** додамо рядки:

```
SetStyle(ControlStyles::UserPaint, true);
 SetStyle(ControlStyles::AllPaintingInWmPaint, true);
 SetStyle(ControlStyles::DoubleBuffer, true);
```

Вони включають режим буферизації при виведенні зображень на форму.

Після них можна забезпечити рядок підготовки зображення з файлу у раніше підготовлену змінну-об’єкт.

```
MyImage = Image::FromFile("574.png");
```

Рисунок запропоновано у папці із завданням.

!! – скачаний файл картинки покласти у папку проекту, точніше у ту саму папку, де файл form1.h.

Можна не переміщувати файл, але тоді необхідно вставити у команду замість **"574.png"** повний шлях до файлу, наприклад,

```
"C:\\Users\\Lenovo\\Downloads\\574.png"
```

– подвійний \\ для того, щоб система не подумала, що це спецсимвол, наприклад, як \n

Можна також скачати свій рисунок. Ефект буде більший, якщо рисунок буде з прозорим тлом.

На цьому етапі ініціалізацію нашої форми для сприйняття графічних елементів можливо вважати виконаною.

Всім керуватиме таймер. Кнопкою **GO** ми запускатимемо його або призупинятимемо. Тому клікнувши на кнопку **GO** потрапимо до редагування функції обробки кліку по кнопці **button3_Click**:

```
if (Form1::timer1->Enabled == false)
        {
                Form1::timer1->Enabled = true;
                Form1::button3 ->Text = "STOP";
        }
 else if (Form1::timer1->Enabled == true)
        {
                Form1::timer1->Enabled = false;
                Form1::button3 ->Text = "GO";
        }
```

Отже кнопка буде у нас перемикачем: **GO/ STOP.**

Тепер перейдемо до зображення форми і подвійно клікнемо по таймеру.
Попередньо таймеру виставимо інтервал лічби 50 мілісекунд.
У середовищі розробки коду скоректуємо функцію реакції на закінчення лічби таймеру: **timer1_Tick** .

Рядки будуть такими:

```
x = x +(sx*dx) ;
y = y + (sy*dy);
```

Це рядки формування положення зображення на формі.
Для краси, ще додамо контроль за невиходом зображення за межі форми:

```
if ((x >= Form1::Width-100)||(x<=0))  sx=sx*(-1);
                    if ((y >= Form1::Height-100)||(y<=0)) sy=sy*(-1);
```

Коли сформовані і перевірені координати :

- задамо новий чистий графічний буфер для виведення картинки:

```
myBitmap = gcnew Bitmap(Form1::Width, Form1::Height);
```

- підготуємо об’єктний елемент, спеціально передбачений для роботи з графікою. Вище ми вже задали його з типом **Graphics^** . А зараз ще і ініціалізуєм у прив’язці до графічного буфера:

```
graph = Graphics::FromImage(myBitmap);
```

- рисуємо картинку у заданих координатах прямо у графічний елемент графічного буфера:

```
graph->DrawImage(MyImage, x, y, 100.0, 100.0);
```

(Цифри **100.0** означають тут ширину і висоту зображення, що відображатиметься.)

- виводимо графічний буфер як картинку у елемент відображення графіки **pictureBox**:

```
Form1::pictureBox1->Image = myBitmap;
```

Запускаємо і перевіряємо.

Змінюючи властивість таймера (значення у мілісекундах), та значення приросту координат картинки dx та dy при ініціалізації, добиваємося бажаної швидкості та плавності переміщення картинки по екрану.

Самостійно:

Перевірка оперативної пам'яті, виділеної під процес виконання модифікованої програми:

- Не зупиняючи програму, переглянути вікно у Visual Studio, яке відповідає за виведення в режимі реального часу графіку використання пам'яті програмою.
Відобразити цей графік у звіті.

- З Панелі завдань Windows запустити Диспетчер завдань. Відкрити там вкладку Процесів. Знайти там свій процес «назва_проекту.exe» і занотувати у звіт дані про використання оперативної пам'яті.

3. Відповідно до завдання до попередньої лабораторної роботи, ми створили таке програмне застосування, яке виконувало виведення зображення з певною періодичністю.

По кожному такту таймера виділяється новий (чистий) фрагмент графічної пам'яті **Bitmap** за допомогою команди **gcnew**.

Негатив – виділення все нових і нових фрагментів графічної пам'яті у циклі. Використані фрагменти при цьому прямо не звільняються.

Перевіряємо та коректуємо за потреби код проекту

Ось так мають виглядати оголошені змінні:

```
static float x=100,y=100,dx=10.0,dy=10.0,sx=1.0,sy=1.0;
static Image^ MyImage;
        static Graphics^ graph;
        Bitmap^ myBitmap; (повинно бути)
```

На формі встановлено елемент **pictureBox**. Він розміщений на задньому плані (що стає повністю прозорий при запуску програми). Йому задана властивість **Dock** як **Fill**.

У функції (процедурі) **Form1_load** обробки первинного завантаження форми тільки 4 рядки:

```
SetStyle(ControlStyles::UserPaint, true);
        SetStyle(ControlStyles::AllPaintingInWmPaint, true);
        SetStyle(ControlStyles::DoubleBuffer, true);
```

```
MyImage = Image::FromFile("574.png");
```

У функції (процедурі) **timer1_Tick** обробки спрацювання таймеру всього 8 рядків:

```
x = x +(sx*dx) ;
y = y + (sy*dy);
                    if ((x >= Form1::Width-100)||(x<=0))  sx=sx*(-1);
                    if ((y >= Form1::Height-100)||(y<=0)) sy=sy*(-1);

                    myBitmap = gcnew Bitmap(Form1::Width, Form1::Height);
```
(повинно бути)
```
graph = Graphics::FromImage(myBitmap);
```
(повинно бути)
```
graph->DrawImage(MyImage, x, y, 100.0, 100.0);
Form1::pictureBox1->Image = myBitmap;
```
(повинно бути)

Запуск готового проекту

Запустіть проект на відлагодження інструментом  Локальний відлагоджувач.
Натисніть на кнопку **Go** запуску генерації графічних об’єктів **Bitmap**.

Перевірка оперативної пам’яті, виділеної під процес виконання програми:

3.1. Не зупиняючи програму, переглянути вікно у Visual Studio 2017, яке відповідає за виведення в режимі реального часу графіку використання пам’яті програмою.
Відобразити цей графік у звіті.

3.2. З **Панелі завдань Windows** запустити **Диспетчер завдань**. Відкрити там вкладку **Процесів**. Знайти там свій процес **«назва_проекту.exe»** і занотувати у звіт дані про використання оперативної пам’яті.

Модифікація програми

Видалимо з функції **timer1_Tick** виділені рядки:

```
                    if ((x >= Form1::Width-100)||(x<=0))  sx=sx*(-1);
                    if ((y >= Form1::Height-100)||(y<=0)) sy=sy*(-1);

                    myBitmap = gcnew Bitmap(Form1::Width, Form1::Height);
graph = Graphics::FromImage(myBitmap);
graph->DrawImage(MyImage, x, y, 100.0, 100.0);
```

```
Form1::pictureBox1->Image = myBitmap;
```

перенесемо їх у функцію **Form1_load**, щоб вона виглядала так:

```
SetStyle(ControlStyles::UserPaint, true);
                SetStyle(ControlStyles::AllPaintingInWmPaint, true);
                SetStyle(ControlStyles::DoubleBuffer, true);
                MyImage = Image::FromFile("574.png");
                myBitmap = gcnew Bitmap(Form1::Width, Form1::Height);
graph = Graphics::FromImage(myBitmap);
```

Тепер виділення буферної графічної пам'яті **Bitmap** буде відбуватись лише одноразово – під час первинного завантаження форми.

Додамо до функції **timer1_Tick** виділений рядок, який замість виділення нової графічної пам'яті буде просто очищати ту, що виділена одноразово:

```
if ((x >= Form1::Width-100)||(x<=0))  sx=sx*(-1);
if ((y >= Form1::Height-100)||(y<=0)) sy=sy*(-1);
```

```
graph->Clear(Color::Transparent);
graph->DrawImage(MyImage, x, y, 100.0, 100.0);
Form1::pictureBox1->Image = myBitmap; У функції
```

Запускаємо програму і впевнюємося у її працездатності.

Самостійно:

Використавши властивість головної форми TopMost забезпечити перебування форми завжди над іншими вікнами у Windows. Показати як це можливо зробити програмно та при конструюванні форми.

Перевірка оперативної пам'яті, виділеної під процес виконання модифікованої програми:

- Не зупиняючи програму, переглянути вікно у Visual Studio, яке відповідає за виведення в режимі реального часу графіку використання пам'яті програмою.

Відобразити цей графік у звіті.

- З Панелі завдань Windows запустити Диспетчер завдань. Відкрити там вкладку Процесів. Знайти там свій процес «назва_проекту.exe» і занотувати у звіт дані про використання оперативної пам'яті.

- Порівнюємо результати перевірок використання оперативної пам'яті до та після модифікації (л.р. 1 та л.р. 2). Пояснити причини відмінностей.

Доопрацювати програму:

Додати на форму візуальний елемент збільшення або зменшення числа, який визначатиме розмір рухомого об'єкту. Розмір має змінюватися без зупинки виконання програми.

a. Підказка1: Можна використати елемент NumericUpDown, TrackBar, Поле введення
b. Підказка2: Іноді потрібно перетворити введені користувачем дані з рядка у число
   Для перетворення цілком спрацьовує така конструкція:

```
Змінна_типу_int = (int)  змінна_типу_рядок;
```

Додати на форму візуальний елемент збільшення або зменшення числа, який визначатиме швидкість руху об'єкту. Швидкість має змінюватися без зупинки виконання програми.

c. Підказка3: див. підказки 1 та 2

(бажано спробувати) На початку роботи програми (або у будь-який момент роботи) забезпечити кнопку, яка викликає вікно вибору графічного файлу для відображення

d. Підказка4: Використати

```
openFileDialog1->ShowDialog();
openFileDialog1->FileName;
```

(спробувати) Забезпечити два або більше рухомих об'єктів

e. Підказка5: творчий підхід, або повернутися до цього завдання після виконання наступних лабораторних робіт)

4. Висновок

Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 11.
Ознайомлення з середовищами розробки мовою Python.
Базові операції з даними засобами Python.
Застосування об'єктно-орієнтованого програмування засобами Python.

Мета роботи: Вивчити основи базових операцій з даними та застосування об'єктно-орієнтованого підходу в мові програмування Python.

1. Переглянути приклади фрагментів програмного коду мовою Python. Спробувати виконати подані фрагменти коду і показати результати виконання у звіті.

Оператори перевірки умови:

Python підтримує звичайні логічні умови:

- рівність: A == B
- нерівність: A != Ь
- менше ніж: a < Ь
- менше ніж або дорівнює: a <= Ь
- більше , ніж: a > Ь
- більше або дорівнює: a >= Ь

Запуск дії print("b is greater than a") при виконанні умови

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Оператор if , який ускладнений іншим варіантом умови та дії

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
```

```
elif a == b:
  print("a and b are equal")
```

Вибір дії за умови невиконання жодної з умов

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Формування умови складеної з 2-х умов a > b and c > a

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Формування умови складеної з 2-х умов a > b and c > a

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Випадкові числа

Потрібна бібліотека:

```
import random
```

Формує дробове випадкове число між 0 та 1

```
b = random.random()
print(b)
```

Формує ціле випадкове число між 1 та 10

```
b = random.randint(1,10)  - цей варіант працює лише для python 3.x
print(b)
```

Виконує вибір будь якого елементу з переліку

```
a = ['*', 'A', 'B']
b = random.choice(a)
print(b)
```

Перемішує порядок елементів у переліку

```
a = ['*', 'A', 'B']
random.shuffle(a)
print(a)
```

Формує випадкове дробове число із заданого інтервалу

```
# -100.00...+100.00
b = 200*random.random() - 100
print(b)
```

Цикли

Перебір елементів списку

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Перебір по одній літері

```
for x in "banana":
  print(x)
```

Взяття x почергово з певного діапазону значень:

```
for x in range(6):
  print(x)
```

!!! Слід зазначити , що діапазон (6) НЕ значення від 0 до 6, але значення від 0 до 5.

```
range(2, 6) – діапазон від 2 до 6
range(2, 30, 3) – діапазон від 2 до 30 з кроком 3
```

!! дроби неможливі - тільки цілі числа

Робота зі змінними та типами

Ви можете призначити значення для декількох змінних в одному рядку:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

Строкові змінні може бути оголошено або за допомогою одинарних, або подвійних лапок:

```
x = "John"
# is the same as
x = 'John'
```

Для того, щоб об'єднати текст і рядкову змінну, Python використовує символ `+`:

```
x = "awesome"
print("Python is " + x)
```

Ви також можете використовувати символ `+`, щоб додати рядкову змінну в іншу змінну:

```
x = "Python is "
y = "awesome"
```

```
z =  x + y
print(z)
```

Ви можете отримати тип даних будь-якого об'єкта за допомогою функції type():

```
x = 5
print(type(x))
```

Можуть бути випадки, коли необхідно, щоб вказати тип на змінну:

- Int() – створює ціле число
- float() – створює число з плаваючою точкою (дробне)
- str() - створює рядок з різних типів даних

```
x = int(1)   # x буде 1
y = int(2.8) # y буде 2
z = int("3") # z буде 3
```

```
x = float(1)     # x буде 1.0
y = float(2.8)   # y буде 2.8
z = float("3")   # z буде 3.0
w = float("4.2") # w буде 4.2
```

```
x = str("s1") # x буде 's1'
y = str(2)    # y буде '2'
z = str(3.0)  # z буде '3.0'
```

Робота з багаторядковими змінними

Якщо використовувати три подвійні лапки

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

або три одинарні лапки:

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

в результаті розриви рядків додаються в тому ж місці, як це представлено в коді.

Можливо отримати окремо будь-який по порядку символ з рядка:

```
a = "Hello, World!"
print(a[1])
```

strip() – метод, що видаляє пробільні символи спочатку та вкінці рядка:

```
a = " Hello, World! "
print(a.strip()) # виведе "Hello, World!"
```

replace() – метод, що замінює частину рядка (символ) іншим рядком (символом):

```
a = "Hello, World!"
print(a.replace("H", "J"))
```

split() – метод, що розбиває рядок на частини, якщо він знаходить символи-сепаратори:

```
a = "Hello, World!"
print(a.split(",")) # поверне список ['Hello', ' World!'],
                    # елементи списку тут слова
                    # "," тут сепаратор
```

Рядки можна об’єднувати, використовуючи метод format()
Метод приймає передані йому аргументи, форматує їх, і розміщує їх в рядку, де зазначені заповнювачі {}:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age)) # поверне My name is John, and I am 36
```

Метод може приймати необмежену кількість аргументів:, і розміщувати їх значення у відповідні наповнювачі:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

Ви можете використовувати індексні номери {0}, щоб бути впевненим , що аргументи поміщаються в правильних заповнювачах:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Спец символи для вставлення в рядкові змінні:

Спец символ	**Результат**
\'	символ одиничних лапок
\\	зворотний слеш (Backslash)
\n або \ r	Перехід на новий рядок
\t	символ табуляції (пропуску)

2. Виконати завдання

Завдання 1.

Надрукуйте в консоль рядок без застосування жодної змінної та із застосуванням команд друку кожного рядка сповіщення окремо.

В рядку подайте інформацію у вигляді:

факультет,
група, прізвище, ім’я студента,
номер лабораторної роботи, дата виконання лабораторної роботи,
тематика лабораторної роботи.

Виконайте те ж завдання але із застосуванням єдиної команди друку, але із збереженням розташування інформації в консолі

Завдання 2.

Наведіть приклад однорядкового коментаря до будь-якої команди на ваш вибір.

Наведіть приклад багаторядкового коментаря до будь-якої команди на ваш вибір.

Завдання 3.

Виправте помилку в рядку

```
x = 5
y = "John"
print(x + y)
```

Завдання 4.

Виправте помилку в рядку

```
if 5 > 2:
print("Five is greater than two!")
```

Та запустіть його для перевірки.

Завдання 5.

Присвойте значення двом змінним застосовуючи один оператор =

Завдання 6.

Присвойте 6-тизначне числове значення одразу двом змінним з довільними іменами, але лише однією командою.

Завдання 7.

Маємо дві змінні, які мають однакові за звучанням імена, але з різними регістрами літер в іменах. Запропонуйте набір команд, виконання яких дозволить зробити висновок про те, чи є ці змінні однією і тією ж змінною, чи ці змінні є двома різними змінними.

Завдання 8.

Запропонуйте набір команд, які перевіряють на рівність дві рядкові змінні: значення першої задане за допомогою одинарних лапок, значення другої – за допомогою подвійних лапок. Результат перевірки на рівність має бути відображено в консолі.

Завдання 9.

Створіть глобальну змінну у середині функції. Функція виконує виведення змінної у консоль. Змініть її значення поза функцією. Викличте функцію на виконання і перевірте змінене значення змінної.

Завдання 10.

Створіть набір змінних зі значеннями всіх можливих типів.

Застосуйте інструкцію визначення типу та виведіть двома стовпчиками значення та визначений його тип у консоль.

Завдання 11.

Згенеруйте 20 випадкових чисел в межах від 20.5 до 31.8

Виведіть їх у консоль рядком з проміжками у 4 пробіли.

Завдання 12.

Застосовуючи цикл виведіть кожний парний символ з рядку:

« Python can be used on a server to create web applications. »

Прибравши попередньо пробільні символи з початку та з кінця рядку.

Завдання 13.

Застосовуючи цикл та команди, надані у методичному забезпеченні, виведіть кожне слово з тексту у стовпчик:

« Python can be used on a server to create web applications. Python can be used alongside software to create workflows. Python can connect to database systems. It can also read and modify files. Python can be used to handle big data and perform complex mathematics. Python can be used for rapid prototyping, or for production-ready software development. »

Прибрати попередньо пробільні символи з початку та з кінця рядку, символи ком, крапок.

Завдання 14.

Здійсніть необхідні виправлення в наданому коді:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

Та запустіть його для перевірки.

3. Переглянути приклади фрагментів програмного коду мовою Python. Спробувати виконати подані фрагменти коду і показати результати виконання у звіті.

Python є об'єктно-орієнтованою мовою програмування.
Майже все в Python є об'єктом, зі своїми властивостями і методами.
Клас подібний конструктору об'єкта, або «плану» чи «опису» для створення об'єктів.

Клас – це загальний опис.
Клас описує, який перелік ознак (характеристик) буде у об’єктів.

З одного класу можна створити будь-яку кількість об’єктів.
У всіх цих об’єктів буде однаковий перелік ознак.

Об’єкт – це структура, яка займає ділянку оперативної пам’яті, і яка зберігає конкретні значення ознак, перелік яких описані у певному класі.

Ознаки класу називають ще властивостями класу, або полями класу.
Для того, щоб створити клас, використовуйте ключове слово class:

Створіть клас з ім'ям MyClass, з властивість з ім'ям x:

```
class MyClass:
    x = 5
```

Тепер ми можемо використовувати клас з ім'ям MyClass для створення об'єктів:

```
p1 = MyClass()
print(p1.x
```

Наведені вище приклади - це класи і об'єкти в своїй простій формі, і не дуже корисні в реальних додатках.

Для того, щоб зрозуміти сенс класів ми повинні розуміти функцію вбудованого методу (функції) __init __ ().

Всі класи мають функцію з ім'ям __init __ (), яка виконується завжди, коли ініціюється клас.

Використовуйте __init __ () функцію для присвоєння значень властивостей об'єкта або інших операцій, які необхідно робити, коли об'єкт створюється.

Створіть клас з ім'ям Person, використовуйте __init __ () функцію для присвоєння значень ім'я і вік:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Примітка: __init__() функція викликається автоматично кожного разу, коли клас використовується для створення нового об'єкта.

Об'єкти можуть також містити методи. Методи в об'єктах є функціями, які належать до об'єкта.

Давайте створимо метод в класі Person:

Вставте функцію, яка друкує вітання, і виконайте його на об'єкті p1:

```
class Person:
  def __init__(self, name, age):
    self.name = name
     # властивість name додається автоматично
   # і їй присвоюється значення з аргументу name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Примітка: self параметр являє собою посилання на поточний екземпляр класу, і використовується для доступу до змінних, які належать до класу.

self параметр є посиланням на поточний екземпляр класу, і використовується для доступу до змінних, які належать до класу.

Застосуйте self-параметр, який не повинен бути названий self, ви можете назвати його будь-як (і в кожному методі також може мати будь-яку назву), але він повинен бути першим параметром будь-якої функції в класі: Використайте слова mysillyobject і abc замість self попереднього прикладу:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

4. Виконати завдання (відповісти на питання або виконати завдання)

Завдання 1. Що таке клас?

Завдання 2. Що таке об’єкт?

Завдання 3. Чи можливо створити клас з об’єкту ?

Завдання 4. Чи можливо створити об’єкт з класу ?

Завдання 5. Чи повинен мати клас ім’я class або Class ?

Завдання 6. Яке призначення функції __init__ () у класі ?

Завдання 7. Що таке властивість класу ?
- наведіть приклад власного класу з переліком властивостей

Завдання 8. Що таке метод класу ?

Завдання 9. Наведіть приклад власного класу

a. придумайте властивості для класу, що описуватимуть певні спільні характеристики якоїсь сукупності об’єктів
b. придумайте кілька методів для класу,
i. що дозволяють вивести на екран значення кожної ознаки
ii. що дозволяють вивести на екран вітання від об’єкту
iii. що дозволяють вивести на екран всі значення всіх властивостей об’єкту

Завдання 10. Створіть об’єкт для вашого класу.

Завдання 11. Задайте властивості для створеного об’єкта.

Завдання 12. Викличте метод для створеного об’єкта.

Завдання 13. Створіть ще кілька об’єктів для вашого класу.

Завдання 14. Задайте їм властивості, відмінні від властивостей попереднього об’єкту

Завдання 15. Застосуйте вибірково для цих об’єктів їх методи.

5. Висновок

Зробити висновок про те, наскільки мова Python відрізняється від мови C++. Навести приклади схожих та відмінних операцій.

Лабораторна робота 12.
Створення одно та багатовіконних застосунків та робота з 2D- Vector-графікою засобами Python.
Частина 1. Розробка GUI.

Мета роботи: Здобути навики використання бібліотек Python для роботи з віконними застосунками та векторною графікою.

1. Створення нового проекту

- Встановити PyQt5:

```
pip install pyqt5
```

- Створити файл minipaint.py.

2. Формувати текст програми

Створення базового вікна

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MiniPaint 1.0")
        self.setGeometry(100, 100, 800, 600)
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MiniPaint()
    window.show()
    sys.exit(app.exec_())
```

Пояснення:

`QMainWindow` — головне вікно програми.

`setWindowTitle` — задає назву вікна.

`setGeometry` — задає розмір і позицію вікна.

Додавання полотна для малювання

```
from PyQt5.QtWidgets import QLabel
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MiniPaint 1.0")
        self.setGeometry(100, 100, 800, 600)

        # Полотно
        self.canvas = QLabel(self)
        self.canvas.setPixmap(QPixmap(800, 600))
        self.canvas.pixmap().fill(Qt.white)
        self.setCentralWidget(self.canvas)
```

Пояснення:

`QLabel` + `QPixmap` — виступає як "pictureBox".

`fill(Qt.white)` — робить фон білим.

Додавання меню

```
from PyQt5.QtWidgets import QMenuBar, QAction, QColorDialog

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MiniPaint 1.0")
        self.setGeometry(100, 100, 800, 600)

        self.canvas = QLabel(self)
        self.canvas.setPixmap(QPixmap(800, 600))
        self.canvas.pixmap().fill(Qt.white)
        self.setCentralWidget(self.canvas)

        # Меню
        menubar = self.menuBar()
        file_menu = menubar.addMenu("File")
        exit_action = QAction("Exit", self)
        exit_action.triggered.connect(self.close)
        file_menu.addAction(exit_action)

        color_menu = menubar.addMenu("Color")
        color_action = QAction("Choose Color", self)
        color_action.triggered.connect(self.choose_color)
        color_menu.addAction(color_action)
```

```
        self.pen_color = Qt.black

    def choose_color(self):
        color = QColorDialog.getColor()
        if color.isValid():
            self.pen_color = color
```

Пояснення:

`QMenuBar` — створює меню.

`QAction` — додає пункти меню.

`QColorDialog` — стандартне вікно вибору кольору.

Додавання вибору інструментів

```
from PyQt5.QtWidgets import QComboBox, QVBoxLayout, QWidget

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MiniPaint 1.0")
        self.setGeometry(100, 100, 800, 600)

        self.canvas = QLabel()
        self.canvas.setPixmap(QPixmap(800, 600))
        self.canvas.pixmap().fill(Qt.white)

        self.tools = QComboBox()
        self.tools.addItems(["Line", "Curve", "Eraser"])
        self.tools.setCurrentText("Line")
```

```
        layout = QVBoxLayout()
        layout.addWidget(self.tools)
        layout.addWidget(self.canvas)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)
```

Пояснення:

`QComboBox` — список інструментів.

`addItems` — додає варіанти (Line, Curve, Eraser).

Реалізація малювання

```
from PyQt5.QtGui import QPainter, QPen
from PyQt5.QtCore import QPoint

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        # ... (попередній код)
        self.drawing = False
        self.last_point = QPoint()
        self.pen_width = 3

    def mousePressEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.drawing = True
            self.last_point = event.pos()
```

```
    def mouseMoveEvent(self, event):
        if (event.buttons() & Qt.LeftButton) and self.drawing:
            painter = QPainter(self.canvas.pixmap())
            tool = self.tools.currentText()

            if tool == "Line" or tool == "Curve":
                pen = QPen(self.pen_color, self.pen_width, Qt.SolidLine)
                painter.setPen(pen)
                painter.drawLine(self.last_point, event.pos())
                self.last_point = event.pos()

            elif tool == "Eraser":
                pen = QPen(Qt.white, 20, Qt.SolidLine)
                painter.setPen(pen)
                painter.drawLine(self.last_point, event.pos())
                self.last_point = event.pos()

            self.canvas.update()

    def mouseReleaseEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.drawing = False
```

Пояснення:

`mousePressEvent` — початок малювання.

`mouseMoveEvent` — малювання лінії або гумкою.

`mouseReleaseEvent` — завершення малювання.

Самостійне завдання

Додати інструмент **CirclingLines** (малювання відцентрових ліній).

Реалізувати гумку через `QImage.setPixel()` для прозорого стирання.

Додати зміну товщини пера через `QSpinBox`.

Пояснення ключових моментів

QLabel + QPixmap → виступає як pictureBox у Windows Forms.

QPainter → аналог Graphics у GDI+.

QPen → аналог Pen для малювання ліній.

QColorDialog → стандартне діалогове вікно вибору кольору.

QComboBox → вибір інструменту (Line, Curve, Eraser).

Eraser реалізовано як малювання білим кольором (колір тла).

3. Висновок

Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 13.
Створення одно та багатовіконних застосунків та робота з 2D- Vector-графікою засобами Python.
Частина 2. Впровадження функцій роботи з 2D-Vector-графікою.

Мета роботи: Розширення функціоналу віконних застосункців Python для роботи з векторною графікою можливостями

- змінювати розмір області малювання (canvas),
- створювати інструменти малювання: SmoothCurve, Line, CirclingLines,
- працювати з масивами точок та буферами.

1. Сформувати текст програми:

Зміна розміру області малювання
У PyQt немає `pictureBox`, але ми можемо використати `QLabel` з `QPixmap`.
Для зміни розміру полотна:

```
self.canvas.setPixmap(QPixmap(new_width, new_height))
self.canvas.pixmap().fill(Qt.white)
```

Щоб реалізувати "розтягування" області малювання:
- можна додати маленький `QPushButton` у правий нижній кут,
- при русі мишки над кнопкою змінювати розмір `QPixmap`.

```
def resize_canvas(self, new_width, new_height):
    old_pixmap = self.canvas.pixmap()
    new_pixmap = QPixmap(new_width, new_height)
    new_pixmap.fill(Qt.white)
```

```
    painter = QPainter(new_pixmap)
    painter.drawPixmap(0, 0, old_pixmap)  # копіюємо старе зображення
    painter.end()

    self.canvas.setPixmap(new_pixmap)
```

Інструмент SmoothCurve

У PyQt є метод `drawPolyline` та `drawPath`. Для згладжених кривих можна використати `QPainterPath`.

```
self.points = []  # масив точок

def mousePressEvent(self, event):
    if self.tools.currentText() == "SmoothCurve":
        self.points = [event.pos()]

def mouseMoveEvent(self, event):
    if self.tools.currentText() == "SmoothCurve" and self.drawing:
        self.points.append(event.pos())
        painter = QPainter(self.canvas.pixmap())
        pen = QPen(self.pen_color, self.pen_width, Qt.SolidLine)
        painter.setPen(pen)

        path = QPainterPath(self.points[0])
        for p in self.points[1:]:
            path.lineTo(p)
        painter.drawPath(path)
        self.canvas.update()
```

```
def mouseReleaseEvent(self, event):
    if self.tools.currentText() == "SmoothCurve":
        self.points = []
```

Інструмент Line

```
def mousePressEvent(self, event):
    if self.tools.currentText() == "Line":
        self.last_point = event.pos()

def mouseMoveEvent(self, event):
    if self.tools.currentText() == "Line" and self.drawing:
        # малюємо тимчасову лінію
        temp_pixmap = self.canvas.pixmap().copy()
        painter = QPainter(temp_pixmap)
        pen = QPen(self.pen_color, self.pen_width, Qt.SolidLine)
        painter.setPen(pen)
        painter.drawLine(self.last_point, event.pos())
        self.canvas.setPixmap(temp_pixmap)

def mouseReleaseEvent(self, event):
    if self.tools.currentText() == "Line":
        painter = QPainter(self.canvas.pixmap())
        pen = QPen(self.pen_color, self.pen_width, Qt.SolidLine)
        painter.setPen(pen)
        painter.drawLine(self.last_point, event.pos())
        self.canvas.update()
```

Інструмент CirclingLines

Цей інструмент малює всі лінії у основному буфері без використання тимчасового клону.

```
def mousePressEvent(self, event):
    if self.tools.currentText() == "CirclingLines":
        self.last_point = event.pos()

def mouseMoveEvent(self, event):
    if self.tools.currentText() == "CirclingLines" and self.drawing:
        painter = QPainter(self.canvas.pixmap())
        pen = QPen(self.pen_color, self.pen_width, Qt.SolidLine)
        painter.setPen(pen)
        painter.drawLine(self.last_point, event.pos())
        self.canvas.update()
```

Самостійне завдання

Додати можливість зміни товщини пера (QSpinBox).

Реалізувати "гумку" через QImage.setPixel() для прозорого стирання.

Зробити збереження малюнка у файл (QPixmap.save("output.png")).

2. **Висновок**
   Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 14.
Створення одно та багатовіконних застосунків та робота з 2D Vector-графікою засобами Python.
Частина 3. Інтегрування додаткових вікон налаштувань.

Мета роботи: Навчитися створювати графічний редактор на Python + PyQt5 з можливістю:

- малювання ліній, прямокутників, еліпсів та залитих фігур;
- вибору кольорів для ліній та заливки;
- налаштування стилів і товщини ліній та стилів заливки через окремі діалогові вікна;
- збереження та відкриття зображень.

1. Формувати текст програмного застосунку

Створення базового вікна

Створюємо головне вікно з полотном для малювання на основі QLabel + QPixmap.

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel,
QVBoxLayout, QWidget
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt

class MiniPaint(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MiniPaint 2.0")
```

```
        self.setGeometry(100, 100, 800, 600)

        # Полотно
        self.canvas = QLabel()
        self.canvas.setPixmap(QPixmap(800, 600))
        self.canvas.pixmap().fill(Qt.white)

        layout = QVBoxLayout()
        layout.addWidget(self.canvas)

        container = QWidget()
        container.setLayout(layout)
        self.setCentralWidget(container)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MiniPaint()
    window.show()
    sys.exit(app.exec_())
```

Додавання меню

Додаємо меню File (Save, Open) та Color (For Lines, For Fill).

```
from PyQt5.QtWidgets import QMenuBar, QAction, QColorDialog,
QFileDialog

menubar = self.menuBar()
file_menu = menubar.addMenu("File")

save_action = QAction("Save", self)
```

```
save_action.triggered.connect(self.save_image)
open_action = QAction("Open", self)
open_action.triggered.connect(self.open_image)

file_menu.addAction(save_action)
file_menu.addAction(open_action)

color_menu = menubar.addMenu("Color")
line_color_action = QAction("For Lines", self)
line_color_action.triggered.connect(self.choose_line_color)
fill_color_action = QAction("For Fill", self)
fill_color_action.triggered.connect(self.choose_fill_color)

color_menu.addAction(line_color_action)
color_menu.addAction(fill_color_action)
```

Діалогові вікна для стилів

Створюємо окремі діалогові вікна для стилів пера та заливки.

PenDialog — товщина та стиль ліній:

```
class PenDialog(QDialog):
    def __init__(self, current_width=3, current_style=Qt.SolidLine):
        super().__init__()
        self.setWindowTitle("Pen Presets")
        layout = QFormLayout()
        self.width_spin = QSpinBox()
        self.width_spin.setValue(current_width)
        self.width_spin.setRange(1, 20)
        self.style_combo = QComboBox()
        self.style_combo.addItems(["Solid", "Dash", "Dot", "DashDot"])
```

```
        layout.addRow("Width:", self.width_spin)
        layout.addRow("Style:", self.style_combo)
        buttons = QDialogButtonBox(QDialogButtonBox.Ok |
QDialogButtonBox.Cancel)
        buttons.accepted.connect(self.accept)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)
        self.setLayout(layout)

    def get_values(self):
        style_text = self.style_combo.currentText()
        if style_text == "Dash":
            style = Qt.DashLine
        elif style_text == "Dot":
            style = Qt.DotLine
        elif style_text == "DashDot":
            style = Qt.DashDotLine
        else:
            style = Qt.SolidLine
        return self.width_spin.value(), style
```

BrushDialog — стиль заливки:

```
class BrushDialog(QDialog):
    def __init__(self, current_style=Qt.SolidPattern):
        super().__init__()
        self.setWindowTitle("Brush Presets")
        layout = QFormLayout()
        self.style_combo = QComboBox()
        self.style_combo.addItems(["Solid", "Dense1", "Dense4", "Cross"])
```

```
        layout.addRow("Fill Style:", self.style_combo)
        buttons = QDialogButtonBox(QDialogButtonBox.Ok |
QDialogButtonBox.Cancel)
        buttons.accepted.connect(self.accept)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)
        self.setLayout(layout)

    def get_values(self):
        style_text = self.style_combo.currentText()
        if style_text == "Dense1":
            style = Qt.Dense1Pattern
        elif style_text == "Dense4":
            style = Qt.Dense4Pattern
        elif style_text == "Cross":
            style = Qt.CrossPattern
        else:
            style = Qt.SolidPattern
        return style
```

Малювання фігур

Реалізуємо малювання фігур у функціях mousePressEvent, mouseMoveEvent, mouseReleaseEvent.

Підтримуються інструменти: Line, Rectangle, Ellipse, FillRectangle, FillEllipse.

```
def draw_rectangle(self, start_point, end_point):
    x1, y1 = start_point.x(), start_point.y()
    x2, y2 = end_point.x(), end_point.y()
    rect_x = min(x1, x2)
    rect_y = min(y1, y2)
```

```
    width = abs(x2 - x1)
    height = abs(y2 - y1)
    painter.drawRect(rect_x, rect_y, width, height)

def draw_ellipse(self, start_point, end_point):
    x1, y1 = start_point.x(), start_point.y()
    x2, y2 = end_point.x(), end_point.y()
    rect_x = min(x1, x2)
    rect_y = min(y1, y2)
    width = abs(x2 - x1)
    height = abs(y2 - y1)
    painter.drawEllipse(rect_x, rect_y, width, height)

def draw_filled_rectangle(self, start_point, end_point):
    rect = QRect(start_point, end_point)
    painter.setBrush(QBrush(self.fill_color, Qt.SolidPattern))
    painter.setPen(QPen(self.line_color, self.pen_width))
    painter.drawRect(rect)

def draw_filled_ellipse(self, start_point, end_point):
    rect = QRect(start_point, end_point)
    painter.setBrush(QBrush(self.fill_color, Qt.SolidPattern))
    painter.setPen(QPen(self.line_color, self.pen_width))
    painter.drawEllipse(rect)
```

Збереження та відкриття зображень

Використовуємо QFileDialog для збереження та відкриття зображень.

```
def save_image(self):
```

```
    file_path, _ = QFileDialog.getSaveFileName(self, "Save Image", "", "PNG
Files (*.png);;JPEG Files (*.jpg);;BMP Files (*.bmp)")
    if file_path:
        self.canvas.pixmap().save(file_path)

def open_image(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Open Image", "",
"Image Files (*.png *.jpg *.bmp)")
    if file_path:
        pixmap = QPixmap(file_path)
        self.canvas.setPixmap(pixmap)
```

Самостійне завдання

Додати інструмент FillCircle (залите коло).

Реалізувати кнопку Clear, яка очищає полотно.

Додати можливість змінювати розмір полотна вручну.

Додати вибір стилів через QComboBox у головному вікні як альтернативу діалогам.

2. **Висновок**
   Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 15.
Розробка застосунку з використанням технології віртуалізації в 3D-просторі засобами Unity, C#.
Частина 1. Створення 3D-простору.

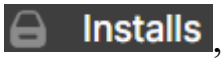
Мета роботи: Здобути навики використання ігрового движка UNITY при створенні програм з 3D інтерфейсом.

1. Підготувати середовище програмування до створення програмного застосунку:

Якщо у попередніх завданнях не було встановлено движок Unity:

Скачати редактор графічного движка Unity 2017.3.1f1 (64-bit) . !! Матеріали лабораторної роботи стосуються саме цієї версії. Завантажити встановлював можливо за посиланням: https://unity3d.com/get-unity/download/archive де обрати потрібну версію, або прямо завантажити встановлював https://unity3d.com/get-unity/download?thank-you=update&download_nid=49210&os=Win або скористатися доданим файлом: UnityDownloadAssistant-2017.3.1f1.exe

Запускаємо редактор графічного движка UNITY: Пуск -> Всі програми -> Unity 2017.3.1f1 (64-bit)-> Unity.exe. На запит ввести пароль та логін, можна зареєструватися (рекомендовано). Програмне забезпечення безкоштовне. Відмітьте лише при реєстрації персональний тип ліцензії та найменший прибуток.

Якщо версія Unity більш нова, то спочатку встановиться Unity Hub, в якому слід знайти пункт 

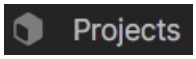
, де натиснути 

. Далі слідувати вказівкам майстра встановлення. Далі деякі команди та дії будуть можуть відрізнятись від версії Unity 2017.3.1f1, тому будуть застосовані додаткові зауваження.

СТВОРЕННЯ НОВОГО ПРОЕКТУ

Запускаємо редактор графічного движка UNITY: **Пуск -> Всі програми -> Unity 2017.3.1f1 (64-bit)-> Unity.exe** або **запускаємо Unity Hub [для нов. верс.]**.

Відкриваємо новий проект, натиснувши спочатку 

, або в **Unity Hub** --- 

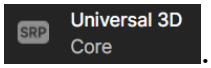
--- 

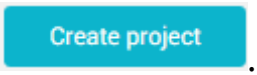
--- зазначити версію Unity 

, якщо їх кілька.

У полях, що з'являються задаємо бажане ім'я та папку для збереження вашого нового проекту.

Залишаємо також позицію 

--- або 

. Підтверджуємо все, натиснувши 

.

Терпляче зачекаємо кілька хвилин, доки проект завантажиться.

НАВІГАЦІЯ у вікні вигляду 3d сцени

Спочатку потренуємося.

Відмітимо інструмент головного меню 
. Це навігація вручну.

У вкладці **Вид Сцени** – **Scene** спробуємо, натиснувши **ЛКМ**, обережно порухати мишкою.

Цей інструмент також можна викликати, натиснувши у вкладці **Scene** на коліщатко мишки.

Тепер наближатимемо вид сцени чи віддалятимемо вид сцени у вкладці **Scene** обертаючи коліщатко мишки вперед і назад.

Можна також обертати вид. У вкладці **Scene** спробуємо, натиснувши тепер вже **ПКМ**, обережно порухати мишкою.

Спробуємо ще зміщуватись по сцені за допомогою звичайних клавіш ←↑→↓ на клавіатурі (але маючи активною вкладку **Scene**).

2. Виконання проєкту:

НАПОВНЕННЯ сцени базовими об’єктами

Додамо твердь земну на сцену: **Головне меню – GameObject – 3D object – Terrain**

У вікні ієрархії **Hierarchy** з’явилася позиція **Terrain**. Вчинимо подвійний клік на ній, щоб центрувати вид у **Scene**.

Тепер заповнимо нашу місцину текстурою.

Варіант 1.

Якщо викладач надав папку із додатковими матеріалами для виконання лабораторної роботи з пакетами ресурсів:

Імпортуємо текстури**: Головне меню – Assets(Активи) – Import Package– Custom Package –** оберемо у папці **Додаткові матеріали до лр.** файл **nature_textures.unitypackage –** у вікні з переліком компонент обираємо кнопку .

Після імпорту у вкладці **Project** з’явилася позиція **NatureStarterKit2**. Розкриємо її і побачимо різні текстури.

Варіант 2.

Якщо такої папки нема, то можливо самостійно завантажити потрібний ресурс:

Для версії Unity **2017.3.1f1** за допомогою меню можливо імпортувати деякі пакети ресурсів, в яких присутні заготовки текстур. У вкладці **Project --- пр. кн. мишки** на кореневу папку **Assets** ---- обираємо команду **Import Package** --- **Environment** --- трохи чекаємо завантаження --- у вікні, що з’явилося знімаємо галочку з кореневого елементу **Standard Assets** (тоді галочки знімуться з усіх інших також) --- знаходимо у списку папку **Terrain Assets з підпапкою Surface**

Textures --- **обираємо текстуру**, яку бажаємо, ставимо галочку --- **клікаємо на кнопку Import**.

Варіант 3.

У вкладці **Project** --- **пр. кн. мишки** на кореневу папку **Assets** --- обираємо команду **Create** --- **Folder** --- назвемо нову папку **MyTextures**.

Можливо дуже просто розмістити в проекті власну текстуру. Просто в Інтернеті знайти на власний смак файл зображення (бажано квадратний). Краще шукати текстури за запитом **Seamless Terrain Textures** (безшовні текстури для земної поверхні). Зберегти файл у папку проекту підпапку **Assets/MyTextures** , просто через **провідник Windows**.

Отже, файл **зображення** відображається **у вкладці Project у підпапках кореневої папки Assets**.

Клікнемо знову по **Terrain** у вкладці **Hierarchy**. У вкладці **Inspector** з’явилися налагодження елементу **Terrain**.

Варіант для Unity 2017.3.1f1

В **Інспекторі** виберемо інструмент  – знайдемо і клікнемо кнопку 

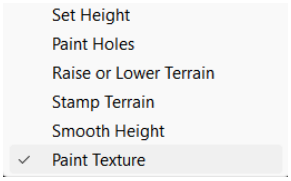
– оберемо команду **Add Texture** (Додавання текстури) – у вікні, що з’явилося – оберемо кнопку 

– далі оберемо текстуру **ground01** – підтвердимо додавання кнопкою 

.

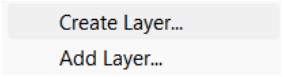
Варіант для Unity 6000.0.44f1

В **Інспекторі** виберемо інструмент  . Під ним виберемо тип дії: **Paint Texture** ---



Трохи нижче додамо у вікні Terrain Layers за допомогою кнопки

прошарок текстури:



та оберіть подвійним кліком завантажену текстуру, яка має відображатись в окремому вікні серед інших наявних текстур.

Наблизившись у вкладці **Scene** до об’єкту **Terrain**, можна впевнитись у наявності текстури на поверхні.

Додамо на сцену небеса.

Варіант 1.

Якщо викладач надав папку із додатковими матеріалами для виконання лабораторної роботи з пакетами ресурсів або завантажити з https://drive.google.com/file/d/1FisHLNfAu3oK3t2gAIBuyFb1rhI2WcHj/view?usp=sharing

Головне меню Unity – Assets(Активи) – Import Package– Custom Package – виберемо у папці завантажений файл **–** у вікні з переліком компонент обираємо кнопку **Import.**

Варіант 2.

Можемо імпортувати небесний пакет з магазину Unity. **У пункті меню Windows** можна знайти підпункт **Asset Store**. Ми потрапимо у **магазин** різноманітних заготовок Unity (можливо, попередньо потрібно буде зареєструватися).

Ввести у полі пошуку слово **Sky**

Встановити **Фільтри пошуку**:

- Виставивши в **Pricing** тип **FREE Assets** (безкоштовні),
- в **Unity Versions** версію Unity (у нашому випадку це буде **2017.x** або **6000.x**),

Вибрати заготовку за бажанням та виконати для неї потрібні дії для завантаження **(Дотримуйтесь інструкцій майстра завантажувача пакетів – можливо кілька разів необхідно буде підтверджувати завантаження та імпорт на різних рівнях)**:

наприклад, **Add to My Assets**… **Open in Unity** … **Download** … **Import** …

Якщо **Package Manager** не відкрився, то відкрити його через головне меню Unity --- **Window** --- **Package Manager** , далі … **Download** … дочекатися завантаження 

… 

… ще раз на кнопку **Import** …

Отже, **імпортовані елементи** з’являються **у вкладці Project** (у одній з папок, що має назву імпортованого пакета).

Просто знайдемо у вкладці **Project у пункті із назвою завантаженого пакету** позицію типу **Sky** , розкриємо її, виберемо об’єкт для завантаження, наприклад:

або 

(це може бути файл текстури або матеріалу тощо)

і перетягнемо його на небосхил у вкладці **Scene**.

!! Якщо текстура неба не відображається у вкладці **Scene**, то у цій вкладці **на панелі інструментів** (зверху вкладки) обрати пункт **Toggle skybox, fog, …** та виставити **галочку** навпроти пункту **Skybox**.

Пообертаємося у вкладці **Scene** і оцінимо додане.

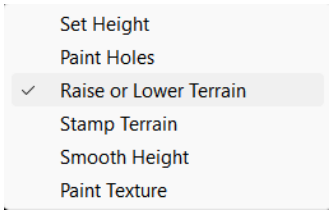
ДОДАВАННЯ ЛАНДШАФТУ

Додамо до земної поверхні ландшафтну кривизну.

Виділимо позицію **Terrain** (наша місцина) у вікні **Hierarchy**.

Варіант для Unity 2017.3.1f1

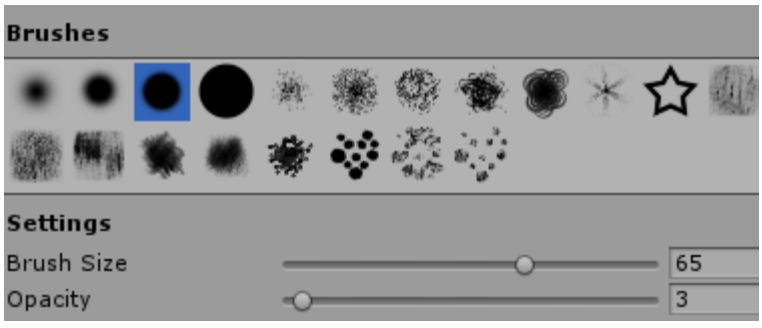
В інспекторі знайдемо інструмент 

В **Інспекторі** виберемо інструмент  . Під ним виберемо тип дії: **Raise or Lower Terrain** 

Повернемося у вікно **Sceen –** трохи наблизимося до поверхні об’єкту **Terrain** (земної поверхні) – наводимо на точки поверхні і спостерігаємо зону дії інструмента відмічену розмитим колом  – багаторазово клікаючи мишкою та затримуючи **ЛКМ** на точках поверхні формуємо пагорби.

Рекомендуємо зробити вищі пагорби по краях земної поверхні.

Можливо налагодити інструмент ландшафту обравши тип зони, величину зони, тощо.



Можна знижувати (згладжувати) створений ландшафт, якщо затиснути **Shift** на клавіатурі і натискати мишкою на форми рельєфу.

Додамо до земної поверхні об’єкти природи: дерева та кущі.
Імпортуємо текстури та об’єкти**:**

Варіант 1:

Якщо такі надав викладач або завантажити https://drive.google.com/file/d/155QbmBelwKJsxNrGT2ieJ_0OOKrB8qGs/view?usp=sharing

Головне меню Unity – Assets(Активи) – Import Package– Custom Package – виберемо у папці завантажений файл **–** у вікні з переліком компонент обираємо кнопку **Import.**

Після імпорту у вкладці **Project** позиція **NatureStarterKit2** поповнилася новими об’єктами. Розкриємо її і побачимо різні текстури та об’єкти – оберемо і розкриємо підпункт **Nature**

Варіант 2:

Пошук безкоштовних asset-ів **Trees та Bushes в AssetStore**, зкачування та імпорт.

Коли пакети об’єктів природи завантажено,
– перетягуємо різні об’єкти **tree** та **bush** на бажані точки земної поверхні у вікні **Scene**.

Після формування власного світу, є бажання помандрувати ним.
Тому необхідно додати гравця.

ДОДАВАННЯ ОБ’ЄКТУ-ГРАВЦЯ на сцену

Іноді у простих випадках додають лише імітацію гравця.

Клікнемо на вкладці **Scene**. Наблизимося десь до середини нашої земної поверхні. Додамо об'єкт-замінник гравця: **Головне меню – GameObject – 3D object – Capsule**

У вікні ієрархії **Hierarchy** з'явилася позиція **Capsule**. Вчинимо подвійний клік на ній, щоб центрувати вид у **Scene**. Якщо об'єкт не на земній поверхні, поставимо його туди. Клікнувши в вікні **Hierarchy** об'єкт **Capsule –** у вікні Inspector задамо координату **Y = 0** – ще раз подвійно клікнемо в вікні **Hierarchy** об'єкт **Capsule** для його локалізації у вікні **Scene.**

Дещо піднімемо **Capsule** над земною поверхнею потягнувши за його вертикальну вісь мишкою, так, щоб **Capsule** був трохи вище поверхні землі.

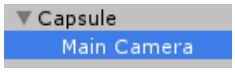
Додамо для цього об'єкта дію гравітації.

У вікні **Hierarchy** клікнемо об'єкт **Capsule –** перейдемо до вікна **Inspector** та натиснемо Add Component **–** оберемо пункт **Physics** –підпункт **Rigid body** (жорстке тіло).

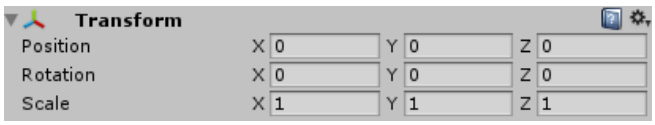
Запустимо гру ▶ – поспостерігаємо, як **Capsule** падає на землю – вимкнемо Режим Гри, натиснувши ще раз ▶.

Заставимо основну камеру відстежувати гравця.

У вікні **Hierarchy** об'єкт камери **Main Camera** перетягнемо у об'єкт **Capsule**, так, щоб **Main Camera** стала підпунктом в **Capsule:**



Як бачимо камера десь поодаль від **Capsule**, тому потрібно задати їй нові координати. Тепер її координати будуть відносно **Capsule** тому можемо сміливо для неї виставити нульові координати: виділимо **Main Camera** в **Hierarchy** і у вікні Inspector виставимо:



Одразу ж спостерігаємо у вікні **Game** зміну загального вигляду.

Далі додамо скрипт руху нашого гравця на сцені. Скрипт в Unity – це шматок програмного коду.

ДОДАВАННЯ до ОБ'ЄКТА-ГРАВЦЯ можливостей руху

Спочатку додамо вбудовану систему контролю руху гравця **Character Controller.**

У вікні **Hierarchy** клікнемо об'єкт **Capsule –** перейдемо до вікна **Inspector** та натиснемо Add Component **–** оберемо пункт **Physics – Character Controller**. У налагодження цього компонента поки заглиблюватись не будемо.

Тепер додамо скрипт переміщення для **Capsule.**

У вікні **Hierarchy** клікнемо об'єкт **Capsule –** перейдемо до вікна **Inspector** та натиснемо Add Component **–** оберемо пункт **New Script –** підтверджуємо створення кнопкою **Create and Add –** нова позиція **New Behavior Script** з'явилася для **Capsule** у вікні **Inspector** знизу **–**обираємо в цій позиції позначку ⚙ (*Unity 2017*) або ⋮ (*Unity 6000*)**–** обираємо команду **Edit script –** чекаємо кілька майкрософтовських хвилин **–** і автоматично переходимо в **Visual Studio** для редагування.

Заготовка коду виглядає так.

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

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {
```

```
        }

        // Update is called once per frame
        void Update () {

        }
}
```

Це код на C#, але саме в цьому випадку майже відсутня різниця з класичним C++.

Зовсім трохи теорії.

Секція **using** – це аналог **using namespace** разом з **#include** в C++.
Заготовка коду описує клас **NewBehaviourScript**, який унаслідується від стандартного базового класу **MonoBehaviour** бібліотеки Unity.
Наша задача – це описати члени класу: методи та поля.
Описані члени класу будуть належати об'єкту нашої сцени в Unity.
Об'єкт класу **NewBehaviourScript**, буде автоматично створений Unity за описаним нами класом і підключений до об'єкту сцени.
Описані в цьому класі методи будуть взаємодіяти з іншими об'єктами та компонентами.
У заготовці вже є стандартний метод **Start ()**. Його Unity викликатиме один раз при запуску сцени.
У заготовці є також стандартний метод **Update ()** – його Unity викликатиме по кожному тіку таймера кадрів анімації (тобто з кадровою частотою).
Start () та **Update ()** не описані. Далі ми це зробимо.

Нарешті заповнимо запропоновану заготовку корисним кодом.

Після рядка **public class NewBehaviourScript : MonoBehaviour {**

оголосимо пусту змінну типу **CharacterController** так:

```
private CharacterController MyCharacterController;
```

Тип цієї змінної дозволяє нам зв'язати її з раніше доданим до об'єкту **Capsule** компонентом **CharacterController.**

Пропишемо це все в середину функції **void Start ()**, тобто між **{** сюди **}**

```
MyCharacterController = GetComponent<CharacterController>();
```

Тепер створимо змінні горизонтального та вертикального переміщення. Також зв'яжемо їх з вхідними сигналами **Input** движка Unity по осям (**Axis**). Пропишемо їх вже в середину функції **void Update ()**, тобто між **{** сюди **}**

```
float MyHorizMove = Input.GetAxis("Horizontal");
float MyVertMove = Input.GetAxis("Vertical");
```

На кожному кадрі в ці змінні-поля зчитуватимуться ненульові значення, якщо натиснуті кнопки клавіатури ←→ (горизонтальна вісь – **Horizontal**) ↑↓ (вертикальна вісь – **Vertical**).

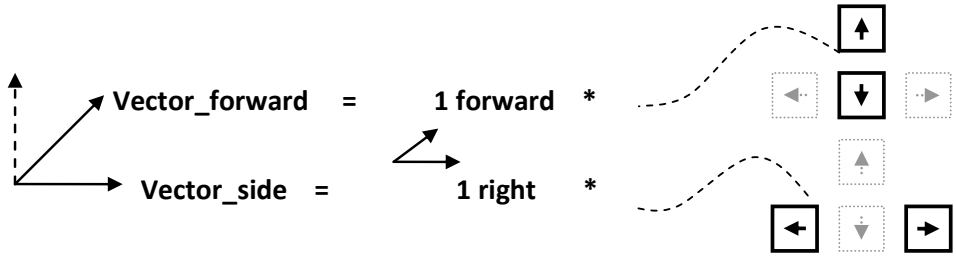
Далі до вже доданого поля **private CharacterController MyCharacterController;** додамо поля:

```
private Vector3 MyForvardVectMove; // вектор руху вперед
private Vector3 MySideVectMove;  // вектор руху у сторону
```

Ці поля-змінні будуть зберігати значення векторів руху у тривимірному просторі.
В функції **void Update ()** вони будуть перераховуватись залежно від значень **MyVertMove** та **MyHorizMove** які отримали від кнопок.

```
MyForvardVectMove = transform.forward * MyVertMove;
MySideVectMove = transform.right * MyHorizMove;
```

Тут векторам присвоюються базові вектори **transform.forward** та **transform.right** але помножені на значення натиснутих кнопок. Зрозуміло, що нулі породять нульові вектори.



Але це тільки визначено кількість руху по осям. Тепер додамо функції передачі руху. Їх наступними запишемо у метод **Update ()**

```
MyCharacterController.SimpleMove(MyForvardVectMove);
MyCharacterController.SimpleMove(MySideVectMove);
```

Збережемо скрипт, натиснувши на  у **Visual Studio**.

Повернемося у **Unity**.

Для *Unity 6000.0.44f1*

Головне меню Unity --- Project Settings --- Player --- Other Settings ---

Спробуємо запустити програму, натиснувши  .

Спробуємо «походити».

Якщо нашому віртуальному об’єкту-гравцю це не вдалося, або вдалося не так, то нашвидкуруч можемо виправити ситуацію так.

Вимикаємо режим виконання, натиснувши  – у вікні **Hierarchy** відмічаємо **Capsule** переходимо в **Inspector --- Add Component --- Phisics --- Rigidbody** далі шукаємо вньому позиції **Freeze Posision** та **Freeze Rotation** і залочуємо всі осі повороту:

Запускаємо програму  – «ходимо».

!! Іноді необхідно перевірити для об’єкта Capsule у вкладці Inspector в налагодженнях **Character Controller** такий компонент як **Min Move Distance**. Бажано, щоб він мав значення в межах від **0** до **0.001**. Також іноді відсутність руху долається перенесенням рядка MyCharacterController = GetComponent<CharacterController>(); з функції Start() першим рядком у функцію Update().

Так має бути краще, але чи можливо швидше.

ДОДАВАННЯ до ОБ’ЄКТА-ГРАВЦЯ швидкості руху

Повертаємося у **Visual Studio**.

Додамо до раніше доданих полів класу **NewBehaviourScript** ще одне – це змінна швидкості ходи:

```
public float MyWalkSpeed;
```

Тут зберігатиметься значення швидкості. Специфікатор дозволить нам побачити поле задавання швидкості прямо в **Inspector** у позиції скрипта **NewBehaviourScript**.

Скоректуємо рядки визначення векторів переміщення, які у методі **Update ()**:

```
MyForvardVectMove = transform.forward * MyVertMove * MyWalkSpeed;
MySideVectMove = transform.right * MyHorizMove * MyWalkSpeed;
```

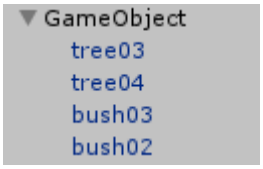
Запускаємо програму ▶ – виконуємо переміщення – порівнюємо з попереднім – пробуємо задати швидкість в Inspector для відміченого в Hierarchy об’єкту **Capsule** у полі **MyWalkSpeed**.

ВПОРЯДКУВАННЯ ОБ'ЄКТІВ У ВІКНІ HIERARCHY

Упорядкуємо безлад у вікні **Hierarchy**.

Створимо пустий об'єкт: **Головне меню – Game object – Create empty**

У **Hierarchy** з'явилася позиція **Game Object**. Перетягнемо туди позиції **tree** та **brush**.



Якщо закрити об'єкт **Game Object**, то структура стане більш впорядкованою.

3. **Висновок**
   Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.

Лабораторна робота 16.
Розробка застосунку з використанням технології віртуалізації в 3D-просторі засобами Unity, C#.
Частина 2. Додавання функцій інтерактивності та GUI.

Мета роботи: Здобути навики використання ігрового движка UNITY при створенні програм з 3D інтерфейсом.

1. Продовжити редагування проєкту, виконаного в попередній л.р.

ВІДКРИТТЯ ПРОЕКТУ

Для Unity 2017.3.1f1 Запускаємо редактор графічного движка UNITY: **Пуск -> Всі програми -> Unity 2017.3.1f1 (64-bit)-> Unity.exe**,
або через Unity Hub відкриваємо попередній проект, натиснувши спочатку

.

Обираємо папку проекту, який зберігали на попередній лабораторній роботі
Терпляче зачекаємо кілька хвилин, доки проект завантажиться.

Продовжимо наповнювати проект функціональністю.

РОЗШИРЕННЯ РУХОМОСТІ ОБ'ЄКТА-ГРАВЦЯ

Рухаючи об'єкт гравця у режимі гри, ми бачимо, що можливо здійснювати рух лише вперед/назад та вправо/вліво. Повернутися наш гравець поки не здатен. Потрібно передбачити додаткові команди у його скрипті.

У вікні **Hierarchy** виділимо **Capsule**. Для об'єкта **Capsule** у **Inspector** виділимо десь знизу позицію скрипта **NewBehaviourScript**. У цій позиції справа є кнопка або . Натисиснувши на неї – обираємо команду **Edit script –** чекаємо – автоматично переходимо в **Visual Studio** для редагування.

Скрипт вже частково нами написаний.

Допишемо після рядка змінної **public float MyWalkSpeed;**

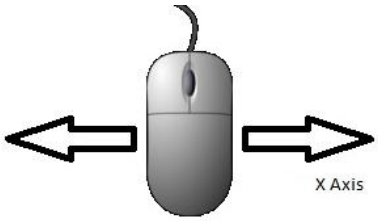
рядок:

```
public float MouseSensitivity;
```

Специфікатор **public** створить у вікні **Inspector** однойменне поле **MouseSensitivity з** доступним для редагування значенням.

Спробуємо задати чутливість мишки в **Inspector** для відміченого в **Hierarchy** об'єкту **Capsule** у полі **MouseSensitivity**. Нехай це буде число **3** (**Hierarchy – Capsule – Inspector – NewBehaviourScript – MouseSensitivity – 3**).

Тепер введемо до програми дані про переміщення мишки взовж **oci X**.



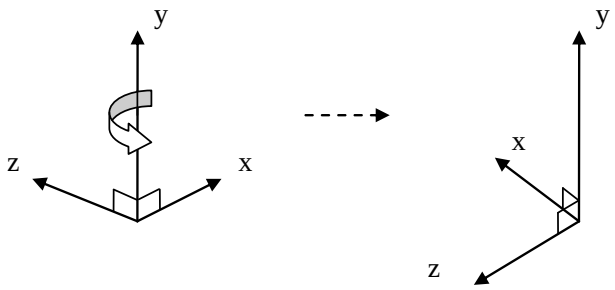
У метод **void Update () { … }** допишемо з кінця:

```
float mouseX = Input.GetAxis("Mouse X");
```

Ми отримали величину переміщень мишки вправо/вліво і зберегли це значення у локальну змінну **mouseX**. Практика показує, що ці значення треба дещо посилити (збільшити). Тому помножимо їх на число **MouseSensitivity** (його задали раніше) і збережемо їх у теж локальну проміжну змінну **RotAmountY**.

```
float RotAmountY = mouseX * MouseSensitivity;
```

Назву **RotAmountY** – величина повороту по **осі Y**, обрано тому, що ми бажаємо, щоб об'єкт гравець здійснював повороти саме по **осі Y**. Ця вісь в Unity – вертикаль.



Особливість у тому, що повертатиметься весь об'єкт-гравець разом з **осями X**, та **Z** (що видно на рисунку).

Далі оголосимо змінну 3-вимірний вектор на ім'я **targetRot** та присвоїмо йому поточні кути об'єкта гравця (кут по осі X, по Y, по Z):

```
Vector3 targetRot = transform.rotation.eulerAngles;
```

eulerAngles – кути Ейлера: Ейлер довів, що будь-яке кутове положення об'єкта можна задати через кути повороту його навколо власних осей координат.

Вибираємо з вектора **targetRot** тільки той кут, що нас цікавить **targetRot.y** та додаємо до нього ту частинку, що задана з мишки:

```
targetRot.y += RotAmountY;
```

Для уникнення зайвих рухів незадіяну **вісь Z** обнулимо командою:

```
targetRot.z = 0;
```

Тепер передамо задані у векторі кути у наш об'єкт командою:

```
transform.rotation = Quaternion.Euler(targetRot);
```

Quaternion – кватернион Гамільтона: Гамільтон розробив спеціальні математичні операції у векторній формі для виконання дій повороту по осям. В Unity поворот по осям Ейлера здійснюється за допомогою **Quaternion.Euler(вектор кутів повороту по 3-м осям)**

Збережемо всі зміни за допомогою інструменту в меню Visual Studio  .

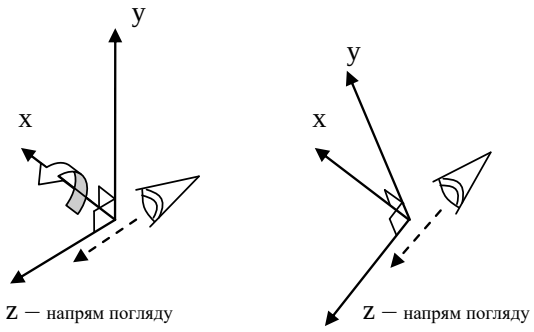
Запустимо гру і перевіримо поворотну рухомість вправо та вліво.

Зверніть увагу, що **Main Camera**, яка приєднана до **Capsule** повертається разом з ним, оскільки вона тепер є частиною **Capsule**.
Також зверніть увагу, що після повороту об’єкт-гравець може продовжити рухатись прямо і вже за новообраним кутом повороту.

Але для більшої реальності іноді хочеться «подивитись» на землю або небо. Поки в нашій реалізації світу це не можливо.

ВСТАНОВЛЕННЯ РУХОМОСТІ ГОЛОВНОЇ КАМЕРИ (MAIN CAMERA)

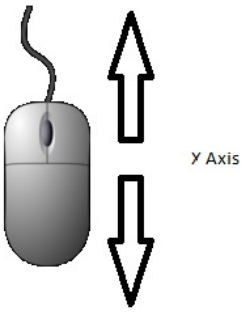
Рух «погляду» донизу та рух доверху зімітуємо повертаючи камеру, яку ми приєднали до об’єкту. Повертати будемо навколо **осі X**.



Відмітимо в **Hierarchy** об’єкт **Main Camera –** для нього в **Inspector –** Add Component **–** оберемо пункт **New Script –** підтверджуємо створення кнопкою **Create and Add – New Behavior Script 2** з’явилося у вікні **Inspector**

знизу – або – **Edit script** – чекаємо – і автоматично переходимо в **Visual Studio** для редагування.

Аналогічно до поворотів вправо/вліво повороти вгору/вниз теж керуватимуться мишкою, але вже рухами по іншій осі.



Заготовка коду для **Behavior Script 2** виглядає так:

пропишемо у метод **void Update () { … }** такі рядки:

```
float mouseY = Input.GetAxis("Mouse Y");
float RotAmountY = mouseY * MouseSensitivity;
Vector3 targetRot = transform.rotation.eulerAngles;
targetRot = transform.rotation.eulerAngles;
targetRot.x -= RotAmountY;
targetRot.z = 0;

transform.rotation = Quaternion.Euler(targetRot);
```

Також додамо після рядку public **class NewBehaviourScript2 : MonoBehaviour {**

рядок оголошення змінної чутливості мишки:

```
public float MouseSensitivity;
```

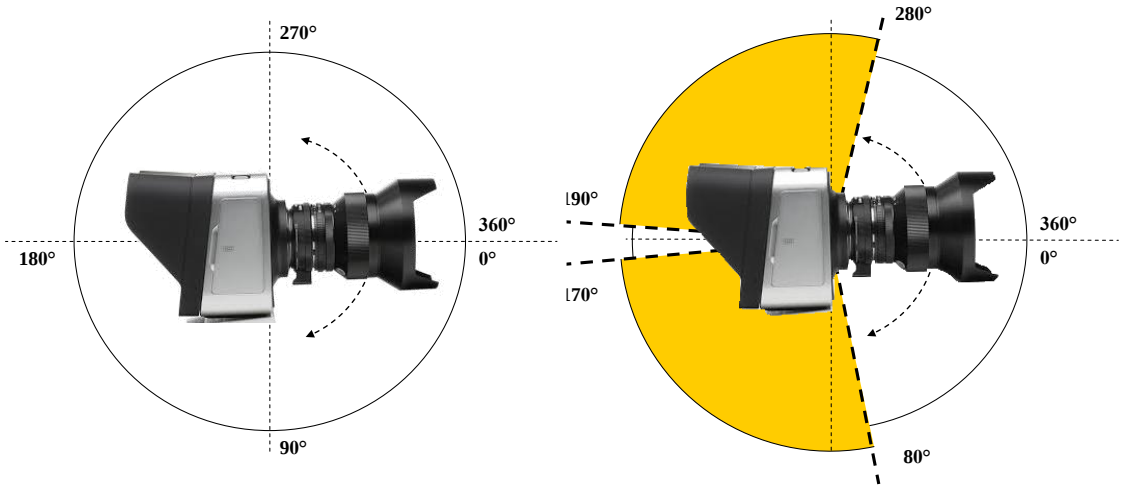
Збережемо всі зміни за допомогою інструменту в меню Visual Studio .

Спробуємо задати чутливість мишки в **Inspector** для відміченого в **Hierarchy** об’єкту **Main Camera** у полі **MouseSensitivity**. Нехай це буде число **3** (**Hierarchy – Capsule – Inspectore – NewBehaviourScript – MouseSensitivity – 3**).

Запустимо гру і перевіримо поворотну рухомість вгору та вниз.

Програючи сцену ви обов'язково помітите особливість поворотів в Unity: при крайніх положеннях вгору та донизу відбувається мерехтливе переключення повороту відносно позицій відносно осі Y, або перевертання виду сцени догори.

Подивимось на градусні міри крайніх положень камери, які ми можемо задати (лівий рисунок):



Обмежимо рухи камери так, щоб вона не входила у зворотні зони (правий рисунок).

Перед рядком **transform.rotation = Quaternion.Euler(targetRot);**

додамо фрагмент контролюючих команд

```
if ((targetRot.x > 190)&&(targetRot.x < 280))
    {
       targetRot.x = 280;
    }
else if ((targetRot.x > 80)&&(targetRot.x < 170))
    {
       targetRot.x = 80;
    }
```

Зберігаємо. Запускаємо. Перевіряємо.

Як бачимо у вікні **Scene**, при повороті вигляду вниз/вгору рухається тільки **Main Camera**, а об’єкт **Capsule** залишається нерухомим.

СТВОРЕННЯ ОБ’ЄКТІВ-АРТЕФАКТІВ ДЛЯ ЗБОРУ ЇХ ОБ’ЄКТОМ-ГРАВЦЕМ

Додамо просто кубики, що обертатимуться.
Hierarchy – центруємо вигляд на **Capsule** подвійним кліком на пункті **Capsule –** додамо об’єкт-артефакт: **Головне меню – GameObject – 3D object – Cube**.

Якщо з**амість Cube** застосувати власний об’єкт, то іноді до нього потрібно додати **Add Component --- Physics --- Capsule Collider або Box Collider**.

Об’єкти можливо зливаються і тільки-но доданий об’єкт дещо занурений у землю, тощо. Тому клікнемо на **Cube** у **Hierarchy** --- потім на інструмент  – активується лише координатні **осі Cube** – трохи піднімемо його над поверхнею, потягши **мишкою** за **вісь Y** – розташуємо об’єкт у потрібному місці на поверхні, потягши мишкою за **інші осі**.

Додамо скрипт обертання до доданих кубиків.
Редагуємо метод **void Update () { … }** додаванням рядків

```
transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime);
```

Метод **.Rotate** дозволяє задати швидкості обертання об’єкту по осям. Тут на кожному кадрі складові кутової швидкості закладені у векторі **Vector3(15, 30, 45)** будуть помножуватись на **Time.deltaTime – час між останніми двома останніми кадрами**, це дає можливість створити рівномірну швидкість обертання незалежно від частоти кадрів (стандартний підхід в Unity).

Можемо запустити гру і переглянути зміни.

Створимо можливість об’єкту-гравцю збирати наші кубики-артефакти.

В момент зіткнення об’єкт-гравець повинен ідентифікувати, що саме з кубиком-артефактом відбулось зіткнення.
Для цього присвоїмо кубику тег – **tag** (мітку): виділимо в **Hierarchy** об’єкт **Cube –** у **Inspector** десь зверху клікнемо**–** оберемо команду **Add Tag –** у вікні, що з’явилося, додамо **тег** кнопкою **+.** **–** замінимо назву **New Tag** на **Pick_Up –** знову клікнемо на **Cube** в **Hierarchy –** знову клікнемо у полі Tag Untagged **–** оберемо з випадаючого списку підготовлену нами позицію **Pick_Up**.

Для «відчування» програмою зіткнення, потрібно налагодити **компоненти зіткнення**.
виділимо в **Hierarchy** об’єкт **Cube –** у **Inspector** знайдемо компонент зіткнення **Box Collider –** у його параметрі **Is Trigger** ставимо галочку ☑.

Виділимо в **Hierarchy** об’єкт **Capsule.**
У скрипті **NewBehaviourScript** об’єкта-гравця **Capsule** після вже наявного опису методу

```
void Update ()
        {
        …
        }
```

пропишемо метод

```
void OnTriggerEnter(Collider other)
        {

        }
```

Цей стандартний метод викликатиметься Unity у момент зіткнення об’єкту-гравця з кубиком-артефактом.

У метод **void OnTriggerEnter(Collider other) { }** пропишемо команди

```
if (other.gameObject.CompareTag ("Pick_Up"))
        {
            other.gameObject.SetActive (false);
         }
```

other. – це посилання на об'єкт, з яким відбувається зіткнення.
.CompareTag – метод, який поверне значення **true**, якщо для об'єкта, у нашому випадку того, з яким відбулося зіткнення, **тег** рівний тому, що вказаний як параметр в дужках;
.SetActive (false) – такий метод з параметром **false** деактивує об'єкт, у нашому випадку той, з яким відбулося зіткнення.

Запускаємо гру та пробуємо стикатися з кубиком-артефактом.

Для швидшого виконання розрахунків движком експерти рекомендують додати до кубика компонент **RigidBody** і активувати його властивість **Is Kinematic**, виставивши відповідну галочку.

Додамо такі кубики у різні місця нашого світу. Для цього :

Просто у вікні **Project** клікнемо пункт **Create** та створимо **New Folder** (нову папку) – переіменуємо її на **MyPrefabs** (мої заготовки) – перетягнемо туди об'єкт **Cube** з вікна **Hierarchy** – рухаючись по поверхні у вікні **Scene**, перетягуємо з папки **MyPrefabs** заготовку **Cube** у бажані місця розташування на земній поверхні.

Запускаємо гру та збираємо кубики-артефакти.

СТВОРЕННЯ ЛІЧИЛЬНИКА ЗБИРАНИХ АРТЕФАКТІВ

Відкриємо для редагування скрипт об'єкта гравця: **Hierarhy – Capsule – Inspector – NewBehaviourScript – або – Edit script**.

Оголосимо ще одну змінну **після** рядка **public float MouseSensitivity;**

```
private int count;
```

Ініціалізуємо змінну нулем, додавши рядок у опис методу **Start() {…}** цього скрипта.

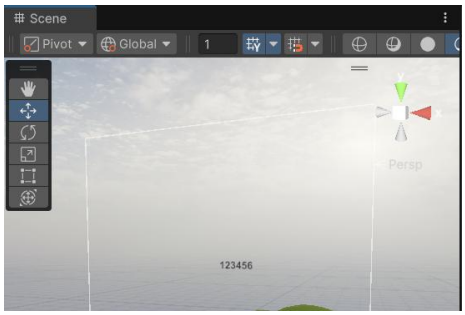
```
count = 0;
```

Тепер додамо команду спрацювання лічильника, тобто збільшення на 1, у процедуру (метод) **void OnTriggerEnter(Collider other)** , який оброблює контакт з артефактом:

```
count++;
```

Потрібно цю інформацію виводити користувачу на екран.
Створимо об’єкт тексту на основному екрані гри: **Hierarhy.**
Перейдемо до головного меню **Unity --- Game Object --- UI --- Legacy – Text**.
Одразу з’являється нова позиція у переліку **Hierarhy**, це позиція **Canvas.Text**.

Побачити візуально елемент **UI на Scene** можливо, клікнувши подвійно на **Canvas** у вкладці **Hierarchy**:

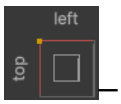


Клікнемо на позицію **Text** у вікні **Hierarhy –** перейдемо до вікна **Inspector –** можемо змінити колір тексту у позиції **Color** (дуже бажано для контрастного

відображення у режимі гри). Можливо також змінити стиль тексту у позиціях **Font**.

Змінимо розташування тексту.

Знову для об'єкту **Text** переглянемо вікно **Inspector** – клікаємо на 

або 

– затиснемо на клавіатурі **Alt** і клікнемо зверху/зліва позицію  .
Тепер текст буде у лівому верхньому кутку головного екрану.

Запрограмуємо виведення даних у доданий елемент **Text**.
В Hierarchy для об'єкту **Capsule** переходимо у вікно **Inspector** – **NewBehaviourScript** –  **або**  **– Edit script**.
Додаємо область для роботи з елементами **UI**: на початку файлу додаємо рядок

```
using UnityEngine.UI;
```

У місці оголошення полів класу додаємо 1 нове поле типу **Text** із назвою **countText**:

```
public Text countText;
```

Специфікатор **public** зробить це поле видимим у позиції компонента NewBehaviourScript об'єкту **Capsule** і ми зможемо помістити туди посилання на об'єкт **Text**, тобто зв'язати змінну `countText` з об'єктом **Text.**

Далі ініціалізуємо значення **Text** за допомогою скрипта.
Додамо рядок до методу **Start() {…}:**

```
countText.text = "Count: 0";
```

Метод **.text** дозволяє отримати доступ до тестового вмісту поля **countText.**

Будемо змінювати **Text** при кожному захопленні артефакту.

Додамо рядок до методу **OnTriggerEnter(Collider other) {…}**:

```
countText.text = "Count: " + count.ToString ();
if (count >= 12)
{
      countText.text = "You Win!";
}
```

Метод **.ToString()** дозволяє нам перетворити число **count** у рядок.

Тепер передамо посилання на об’єкт **Text** у поле **countText.**

У вікні **Hierarchy** клікаємо спочатку об’єкт **Capsule**, так щоб у **Inspector** з’явились властивості та перелік компонент об’єкту **Capsule –** прокручуємо список компонент до компоненту **NewBehaviourScript –** тут має бути поле з іменем **countText** (якщо нема такого поля, то запускаємо та виключаємо режим гри).

У вікні **Hierarchy**, натиснувши та не відпускаючи мишку з об’єкта **Text**, перетягуємо його у поле **countText**.

Одразу ж після можна запустити програму і протестувати її функціональність.

УРІЗНОМАНІТНЕННЯ ВМІСТУ СЦЕНИ

У пункті меню Windows можна знайти підпункт Asset Store. Зайшовши на вкладку (зручно правою кнопкою на заголовку вкладки викликати команду Maximized), ми потрапимо у магазин різноманітних заготовок Unity (попередньо потрібно буде зареєструватися).

Виставивши версію Unity та тип оплати FREE, можливо обрати заготовку за бажанням та виконати її Download.

Пропонується урізноманітнити сцену архітектурними чи природними елементами.

Шукаємо --- обираємо пакет з бажаними моделями ----

Add to My Assets… **Open in Unity** … **Download** … **Import** …
Якщо **Package Manager** не відкрився, то відкрити його через головне меню Unity --- **Window** --- **Package Manager** , далі … **Download** … дочекатися завантаження 

… 

… ще раз на кнопку **Import** …

Отже, **імпортовані елементи** з’являються **у вкладці Project** (у одній з папок, що має назву імпортованого пакета).

Після витягування об’єкту на сцену часто потрібно задавати йому координати, властивість твердого тіла, коєфіцієнти розмірів (Scale), тощо.

СТВОРЕННЯ РІВНІВ ТА ПЕРЕХОДІВ МІЖ НИМИ

У вкладці Hierarchy виділити назву сцени та за допомогою випадаючого меню обрати команду збереження сцени під іншим ім’ям. Зберегти сцену з іншим іменем у підпапку проекту Assets/Scenes (це і буде сцена нашого нового рівня).

За необхідності повернутись до попередньої сцени можна знайти її у пункті Assets у вкладці Project в редакторі Unity. Подвійним кліком активувати її та перейти до редагування.

Через пункт **Build Settings (що у пункті меню File)**, використавши кнопку **Add Open Sceens**, додати поточну нову сцену до формувача проекту.

Особливість, якщо виконувати зміни **Terrain** на новій сцені, то зміни відіб'ються на сцені, з якої породжено нову. Тому рекомендується перед виконанням змін нової сцени видалити на ній **Terrain**, та додати наново.

На кожній сцені призначити об'єкту-гравцю певний Tag: можна створити новий унікальний **Tag** через **Add Tag** (так само як це було виконано для кубиків-артефактів).

Створити будь-який об'єкт на сцені, що виконуватиме роль порталу у іншу сцену.
Відмітити для цього об'єкта **IsTrigger** у пункті **…Collider** у вкладці **Inspector**.

Додати до об'єкту-порталу новий скрипт. У скрипті додати команди:
1) Інструкцію використання компоненту **using UnityEngine.SceneManagement** до вже присутніх **using System.Collections**, **using System.Collections.Generic**, **using UnityEngine**.
2) Функцію обробки спрацювання тригера (як це вже виконувалось).
У ній команда-функція для переходу на потрібну сцену виглядає як **SceneManager.LoadScene("ім'я сцени")**

!! не забути зберегти зміни у скрипті.

Запустити гру та одразу ж вийти з режиму гри, щоб був оброблений новий скрипт.

Перейти до вигляду сцени, до об’єкту-порталу, до його скрипта у вкладці Inspector.
Там має з’явитися нове поле для введення (поки пусте) з іменем нового поля, доданого до класу скрипта. Записати у нього точне ім’я сцени, на яку планується перехід з поточної сцени.

Запустити гру та перевірити перехід.

!! При додаванні нових сцен шляхом клонування з існуючих, може так статися, що одні і тіж об’єкти на різних сценах пов’язані, тобто зміни одного викликають зміни іншого. У такому випадку просто клонувати такий об’єкт на сцені (через випдаюче меню пункт Duplicate), з видаленням зв’язаного, і виконувати вже зміни на клоні.

В результаті ми маємо отримати гру з різними рівнями та порталами переходів з одних рівнів на інші, та назад до першого рівня (мінімальна кількість рівнів - 2).

2. **Висновок**
   Зробити висновок про те, які функції необхідно застосувати для того, щоб виконати заданий алгоритм роботи застосунку.