

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

**ТЕХНОЛОГІЇ ДОПОВНЕНОЇ РЕАЛЬНОСТІ.
ПРАКТИКУМ**

Навчальний посібник

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра за освітньою
програмою «Технології друкованих і електронних видань»
спеціальності G20 Видавництво та поліграфія*

Укладачі: Д. І. Баранова, К. І. Золотухіна

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 655:004.92:004.932.72'1
Б15

Укладачі: *Баранова Дарина Іванівна, PhD, старший викладач*
Золотухіна Катерина Ігорівна, канд. техн. наук, доц.

Рецензент *Чепурна К.О., канд. техн. наук, доцент, доцент кафедри*
ТПВ НН ВПІ

Відповідальний
редактор *Палюх О.О., д-р техн. наук, доцент*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 27.08.2026 р.)
за поданням Вченої ради навчально-наукового Видавничо-поліграфічного інституту
(протокол № 1 від 11.09.2026 р.)

Б15 Технології доповненої реальності. Практикум [Електронний ресурс] : 2-ге видання, перероблене і доповнене : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Технології друкованих і електронних видань» спец. 186 (G20) Видавництво та поліграфія / КПІ ім. Ігоря Сікорського ; уклад.: Д. І. Баранова, К.І. Золотухіна. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 92 с.

У навчальному посібнику наведено перелік робіт, як індивідуальних завдань з дисципліни. Показано застосування теоретичного матеріалу до розв'язування поставлених практичних задач у відповідності до кожної роботи.

Призначений для здобувачів ступеня бакалавра НН ВПІ КПІ ім. Ігоря Сікорського за спеціальністю 186 (G20) Видавництво та поліграфія.

УДК 655:004.92:004.932.72'1

Реєстр. № НП 26/27-045. Обсяг 5,4 авт. арк.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
<https://kpi.ua>

Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП	5
Лабораторна робота №1. СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ ОНЛАЙН РЕСУРСІВ	7
Лабораторна робота №2. РОЗРОБЛЕННЯ ДЕКОРАТОРУ КІМНАТИ ЗА ДОПОМОГОЮ VLIPPAR	16
Лабораторна робота №3. РОЗРОБЛЕННЯ ІНТЕРАКТИВНОГО ПОСТЕРУ З ЕЛЕМЕНТАМИ ДОПОВНЕНОЇ РЕАЛЬНОСТІ.....	21
Лабораторна робота № 4. СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ У СЕРЕДОВИЩІ PROCESSING.....	28
Лабораторна робота №5. СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ З РІЗНИМИ ХАРАКТЕРИСТКАМИ (РУХ ПО ТРАЄКТОРІЇ, ВИКОРИСТАННЯ ТЕКСТУР, РІЗНОМАНІТТЯ ФОРМ) У СЕРЕДОВИЩІ PROCESSING	39
Лабораторна робота №6. СТВОРЕННЯ ІНТЕРАКТИВНОЇ ЛИСТІВКИ У СЕРЕДОВИЩІ PROCESSING	50
Лабораторна робота №7. СТВОРЕННЯ ЕЛЕМЕНТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ UNITY	57
Лабораторна робота №8. СТВОРЕННЯ ЕФЕКТІВ ЗА ДОПОМОГОЮ UNITY ТА ДОДАВАННЯ ЇХ ДО ЕЛЕМЕНТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ.....	62
Лабораторна робота №9. СТВОРЕННЯ ДОДАТКУ З ЕЛЕМЕНТАМИ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ UNITY	76
Лабораторна робота №10. ВИЗНАЧЕННЯ ВПЛИВУ ХАРАКТЕРИСТИК ЗАДРУКОВУВАНОГО МАТЕРІАЛУ НА ПОЛІГРАФІЧНЕ ВИКОНАННЯ AR-МАРКЕРІВ ТА ВІДТВОРЕННЯ НА ЕКРАНІ ЦИФРОВОГО КОНТЕНТУ.....	84

Лабораторна робота №11. ВИЗНАЧЕННЯ ВПЛИВУ ОСВІТЛЕННЯ НА БЕЗВІДМОВНІСТЬ МАРКЕРІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ПРОТЯГОМ ВІДПОВІДНОГО ТЕРМІНУ ЕКСПЛУАТАЦІЇ ПРОДУКЦІЇ...	86
Лабораторна робота №12. ВИЗНАЧЕННЯ ВПЛИВУ ОСОБЛИВОСТЕЙ ЗОБРАЖЕННЯ НА МОЖЛИВІСТЬ ЇХ ВИКОРИСТАННЯ ЯК AR-МАРКЕРІВ.....	88
РЕКОМЕНДОВАНИЙ ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ	90
ДОДАТОК А. ПРИКЛАД ОФОРМЛЕННЯ ТИТУЛЬНОГО АРКУШУ	92

ВСТУП

Навчальний посібник присвячений вивченню основних понять, особливостей, термінів, технологій, засобів контролю та програмних продуктів та особливостей роботи у них, що необхідні для створення якісних мультимедійних інтерактивних продуктів з елементами доповненої реальності для увиразнення контенту друкованої або електронної продукції.

Основна мета цієї дисципліни полягає у наданні студентам комплексного уявлення про:

- особливості розпізнавання образів при використанні різних технологій доповненої реальності;
- основні підходи до розроблення додатків з доповненою реальністю, що базуються на різних принципах та методах розпізнавання об'єктів доповненої реальності;
- основні тенденції та тренди на ринку продукції з доповненою реальністю;
- методи та засоби контролю функціональності та валідності додатків, особливості їх тестування та попередження впливу різних чинників на безвідмовність маркерів доповненої реальності.

Вивчення дисципліни призначено для формування таких предметних здатностей студентів:

- вибір відповідного програмного забезпечення, засобів та методів контролю, технологій доповненої реальності для розроблення мультимедійного продукту з елементами доповненої реальності у залежності особливостей подальшої експлуатації готового продукту, контент якої він буде увиразнювати;
- розуміння особливостей роботи програмного забезпечення, вебресурсів та бібліотек доповненої реальності, а також взаємозв'язку із бібліотеками комп'ютерного зору та принципів роботи у них, при розробленні відповідного мультимедійного продукту з елементами доповненої реальності;
- визначення особливостей наповнення розроблюваного AR-додатку у залежності від навичок цільової аудиторії, характеристик пристрою для розпізнавання, методів та принципів розпізнавання, що плануються втілити у проєкт;
- створення коректного маркеру доповненої реальності з врахуванням поставлених вимог до зображень-маркерів відповідно до обраної технології доповненої реальності та концепції поліграфічного продукту, у контент якої вони впроваджуватимуться;
- розуміння відмінностей між різними технологіями доповненої реальності та їх реалізації для увиразнення чи доповнення контенту відповідної продукції;
- вміння аналізувати поставлені завдання, розробляти план дій відповідно до поставлених завдань, проводити аналіз доцільності використання тих чи інших команд, знати всі можливі способи їх застосування;

— освоєння основ створення валідних маркерів доповненої реальності, опрацювання мультимедійного матеріалу, застосування мов програмування для додатків тощо.

Навчальний посібник дозволить освоїти особливості різних технологій доповненої реальності, вміти відрізняти їх між собою, набути знань щодо їх застосування в тій чи іншій ситуації для розроблення мультимедійних інтерактивних продуктів з елементами доповненої реальності для увиразнення контенту готової поліграфічної продукції; навчитися роботі у різноманітних додатках для розроблення елементів доповненої реальності; самостійно вирішувати конкретні завдання з вибору оптимальних технологічних рішень щодо створення якісних коректно працюючих додатків з елементами доповненої реальності та впроваджувати їх у контент поліграфічної продукції.

Посібник призначено для студентів денної та заочної форми навчання технічних спеціальностей. Його можна використати також для підготовки до занять, заліків, екзаменів студентам всіх форм навчання, які вивчають подібний матеріал. Курс відповідає нагальній ринковій потребі підготовки сучасних фахівців.

СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ ОНЛАЙН РЕСУРСІВ

Мета роботи: вивчення інтерактивних проєктів із елементами доповненої реальності. Створення об'єктів доповненої реальності за допомогою веб-ресурсу blippar.com

Теоретична частина

Доповнена реальність – технологія, що дозволяє впровадити віртуальні об'єкти (текст, ілюстрації, анімація, аудіо, відео тощо) у реальний світ людини шляхом сканування спеціального об'єкту-маркеру, що роздрукований на будь-якому задрукованому матеріалі чи відтворений на екрані монітору, за допомогою камери смартфона чи планшета чи будь-якого іншого пристрою з камерою та подальшого відтворення цифрового контенту на екрані цього пристрою.

Найбільш популярними видами даної технології є маркерна та безмаркерна. Даний практикум буде присвячений розробці елементу доповненої реальності, що базується на безмаркерній технології. У даному випадку маркер-зображення все таки є. Він являє собою складне деталізоване зображення, проте основою для розпізнавання слугують різноманітні дані – положення користувача у просторі, його геолокація, кут повороту пристрою тощо.

Одним із видів програмного забезпечення, що застосовується для створення безмаркерної доповненої реальності є **онлайн-платформи**. Вони дозволяють легко об'єднувати між собою віртуальні об'єкти та самостійно їх зашифровують у елемент доповненої реальності. Вони дають змогу працювати з певним набором безкоштовних функцій та певною кількістю «маркерів». Проте для більш деталізованого додатку слід активувати платну версію. До таких додатків можна віднести **BlippAR, Vuforia, RoAR, PlugXR** тощо. У даному практикумі будуть розглянуті принципи роботи на платформі The RoAR.

Середовище **BlippAR** є сучасним ресурсом для простого створення об'єктів доповненої реальності за допомогою інтерфейсу користувача. Середовище є простим та інтуїтивним і активно оновлюється та доповнюється новими можливостями та функціоналом, завдяки чому дозволяє реалізовувати складні та креативні ідеї та додавати інтерактивності друкованим матеріалам – зовнішній рекламі, постерам, книгам тощо.

Особливості виконання:

1. Підготовка до роботи

Метою роботи є створення інтерактивного додатку з елементами доповненої реальності із застосуванням онлайн-конструктору blippar. Для цього спочатку треба зареєструватися на сайті www.blippar.com (натиснути кнопку «**Signup**» у правому верхньому куті сайту) для створення свого акаунту.

Після проходження реєстрації буде наданий доступ до створеного акаунту (див. рис. 1), де будуть знаходитися вже створені коли-небудь проєкти (навіть якщо користувач тільки відкрив вікно редактора, але при цьому його нічим ненаповнив і вийшов) та кнопки створення нового проєкту. Якщо користувач ще ніколи не створювали проєктів у цьому

онлайн-конструкторі, то просто будуть наявні білі «слоти» під них. Далі треба натиснути на плюсик на одному з них. Як вже було зазначено вище як тільки користувач переходить до меню редактора проєкту, розроблюваний проєкт додається до акаунту.

*Якщо користувач випадково вийшов із редактору або ж вирішив повернутися до процесу редагування з часом, то тоді для цього треба зайти до свого акаунту, знайти необхідний проєкт, навести курсор на зображення використовуваного у проєкті маркеру і натиснути **Open Project**. Всі зміни автоматично зберігаються додатком.*

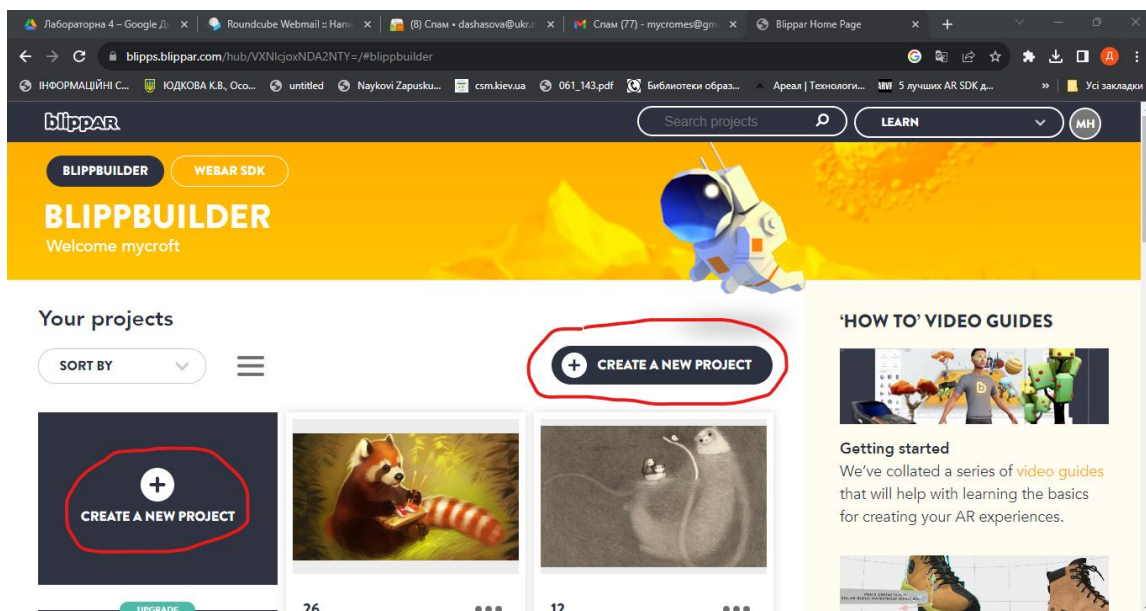


Рисунок 1 – Акаунт у BlippAR

Після створення нового проєкту відкриється нове вікно, де треба прописати назву проєкту та натиснути кнопку **«Continue»**.

Наступним кроком треба буде обрати варіант проєкту. Цей етап визначатиме що саме слугуватиме маркером доповненої реальності. Їх є 4 на вибір:

- **MARKER** – створення додатку на основі сканування плоских 2Д-зображень – для виконання завдання треба обрати **саме цей варіант (натиснути «Select Marker»)**;
- **FACE** – створення доповненої реальності на основі розпізнавання обличчя;
- **SURFACE** – створення додатку на основі сканування 3Д-моделей;
- **AROUND YOU** – створення додатку на основі сканування зовнішнього простору реального світу людини.

Останнім кроком перед створенням наповнення елемента доповненої реальності є вибір зображення-маркеру. Користувач може завантажити до 10 зображень у якості AR-маркерів для одного проєкту (для роботи достатньо **одного**). Тобто всі вони одночасно при їх скануванні будуть програвати один і той самий контент. Вони мають бути у форматі .jpg з розміром однієї з сторін до 1024 пікселів обсягом до 1 Мб. Зображення рекомендовано обирати кольорові та деталізовані – легше та швидше розпізнаються. У **випадку невідповідності додаток видасть повідомлення, що конкретно треба у зображенні підкорегувати**. Для завантаження треба натиснути на кнопку **«Browse and Upload»** та обрати підібране зображення. Також є можливість, щоб вебресурс самостійно згенерував маркер. Для цього треба натиснути на кнопку **«Autogenerate a marker»**. Після підвантаження зображення його можна видалити, натиснувши на кошик поруч, або підтвердити свій вибір, натиснувши **«Confirm»**. Після підтвердження вибору треба

почекати, поки додаток опрацює вибір та надасть доступ до редактора наповнення елементу доповненої реальності.

2. Робота в редакторі

Після успішного проходження попереднього етапу відкриється робочий простір редактору, що наведений на рисунку 2.

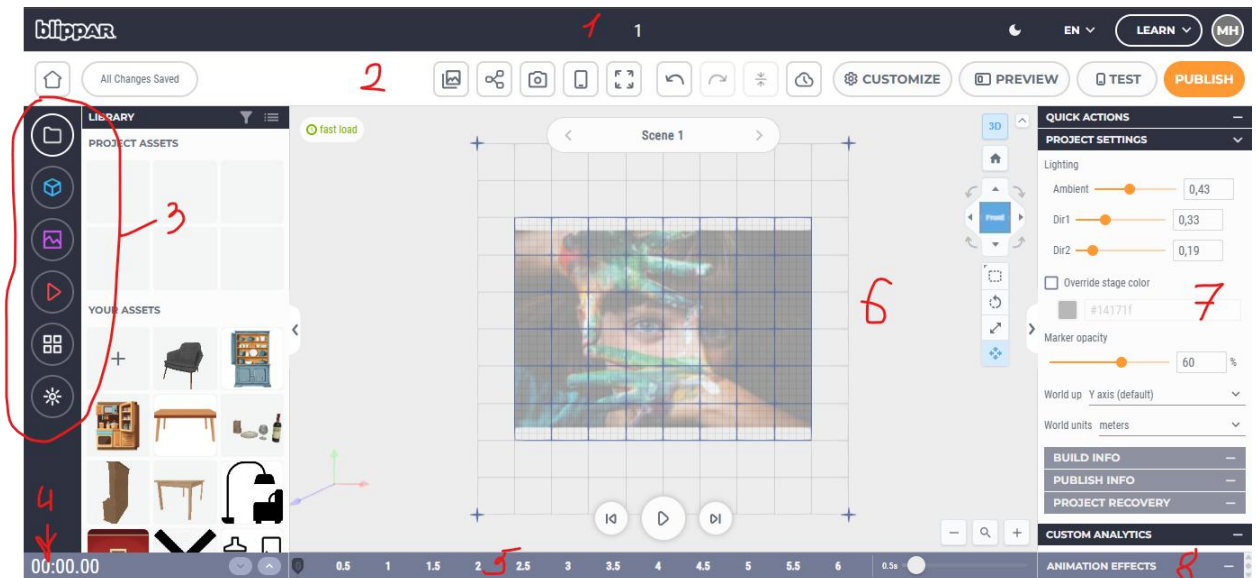


Рисунок 2 – Редактор наповнення елементу доповненої реальності

Він містить такі поля (див. рис. 2):

1. Рядок, що містить **лого онлайн-сервісу**, **назву проєкту**, додаткові документи для вивчення за потреби (**Learn**), кнопку перемикавання режиму (світла / темна тема) робочого простору редактора та мови та **кнопку акаунту**, де можна повернутися до акаунту та створити новий проєкт чи продовжити роботу з іншим, оформити підписку або вийти з акаунту.

2. **Поле меню** – містить такі кнопки (зліва направо по порядку):

- повернення до акаунту (**Return to dashboard**);
- поле з інформацією, що всі дії автоматично зберігаються – якщо у нього натиснути, то з'явиться чекбокс, де можна прибрати цю функцію, знявши галочку;
- Меню налаштування маркеру (**open marker manager**) – тут можна додати інше зображення у якості маркеру, видалити існуюче зображення-маркер (всі внесені зміни треба обов'язково підтвердити, натиснувши на **Done** для збереження змін);
- Меню створення подій та дій (**toggle action and events creator**);
- Прінтскрін робочого простору (**save image from current view**);
- Тестовий режим того, як розроблені елементи виглядатимуть на телефоні (**circle phone preview**) – натиснувши один раз на кнопку телефон буде представлений у вертикальному вигляді, ще раз – горизонтальний вигляд, ще раз – вихід з тестового режиму. Цей режим дозволяє тільки швидко переглянути загалом вигляд сцени без взаємодії з її елементами (кнопки не працюють, анімація невідтворюється автоматично) на уявному телефоні – просто візуальний перегляд та оцінювання;
- Розкриття на повний екран робочого простору (**toggle fullscreen view**), що містить наповнення сцени, що буде бачити користувач – для виходу з цього режиму треба натиснути **Escape**;

- Кнопки відміни дії та повернення до наступної дії (гарячі клавіші «**ctrl+Z**» та «**ctrl+Y**» відповідно);
- Вирівнювання обраних елементів – вирівнюються об'єкти відносно першого обраного об'єкту (**align selected items**). *Поки необрано декілька елементів ця кнопка недоступна. Для вибору декількох елементів треба обрати один з них (відносно нього відбуватиметься вирівнювання), затиснути **ctrl** та обрати інші необхідні елементи, які треба вирівняти. Після цього кнопка стає доступною і, натискаючи на неї, відкривається меню вирівнювання, у якому вже можна обрати відносно якої сторони першого обраного об'єкту треба здійснити цю операцію;*
- Налаштування проєкту (**Customize**)
- Попередній перегляд роботи проєкту (**Preview**) – попередній перегляд сцени з можливістю перегляду роботи анімацій, програвання музики, роботою кнопок – тобто дає можливість перегляду та перевірки роботи додатку;
- Тестування роботи проєкту з використанням згенерованого QR-коду (**Test**) – тестування всіх етапів роботи AR-додатку (як це все відбуватиметься при реальному використанні додатку) – від розпізнавання маркеру доповненої реальності до взаємодії користувача з кожною сценою;
- Публікація проєкту (**publish**) – оформивши підписку на вебресурс можна скористатися цією функцією та опублікувати цей проєкт для того, щоб він міг стати публічним та інші користувачі могли ним скористатися та випробувати.

3. Об'єкти, які можна додати до проєкту – це об'єкти, що користувач може додати до проєкту, їх бібліотека. Вони не є видимими для користувача розроблюваного проєкту, поки розробник їх не перетягне в область наповнення сцени проєкту (6). Ця область містить декілька видів об'єктів, які можна додати. У бібліотеку (**Library of assets**) можна додати аудіо (mp3), відео(mp4), зображення (jpg, png, gif), 3D-моделі (glb, glTF). Цей розділ складається з тих об'єктів, що вже є у проєкті (**Project Assets**) та тих, які користувач коли-небудь використовував при розробці своїх проєктів (**Your Assets**). Для додавання треба натиснути на + полі **Your Assets** цього розділу. Далі відкриється нове вікно, де можна або обрати об'єкти із бібліотеки сайту SketchFab, обравши відповідну вкладку з об'єктом, або свої, натиснувши на вкладку **Browse**, а потім **Browse and upload**. Новий завантажений об'єкт з'явиться у **Your Assets** і для роботи з ним треба його перетягнути у область наповнення сцени проєкту (6) (крім аудіо – воно підвантажується іншим чином – через задання об'єктам параметру **Action/Audio** (при натисканні на об'єкт програватиметься аудіо) або через кнопку **Customize** (тоді музика додається до всього проєкту загалом і буде програватися при його завантаженні)). Всі коли-небудь підвантажені об'єкти будуть постійно у бібліотеці та будуть доступні для всіх майбутніх проєктів.

Також такі об'єкти як зображення, тривимірні моделі та відео винесені в окремі розділи цієї області та містять кнопку додавання об'єктів до системи «+», відповідні об'єкти, що підвантажив користувач, та об'єкти, що є в системі, розділені на тематичні групи та які теж можна додавати до проєкту, перетягнувши їх у робочий простір (6). Також серед об'єктів окремим розділом винесено **бібліотеку віджетів (widgets)** – тут можна додати додаткові наявні у системі елементи, наприклад, текст, графік, карта. Із цього поля для роботи знадобиться текст **Text**. Його треба затиснути та перетягнути у робочий простір сцени, щоб із ним працювати. Так як простір додатку, створеного у бліпарі є тривимірним, то окремим об'єктом, який можна додати до проєкту, є світло. Є різні варіанти світла та можна будь-який додавати до проєкту для вибудовування більш реалістичної картинки для споживачів.

4. Поле налаштування анімації об'єктів (animation layers) – це поле дозволяє зробити статичні елементи сцени динамічними, а також налаштувати тривалість їх анімації, задати точки початку та кінця обраної дії та обрати варіант самої анімації. Якщо розгорнути

це поле, то можна побачити всі створені об'єкти обраної сцени. Разом з цим полем розгортається часова шкала (5) та бібліотека анімацій (8).

5. **Часова шкала** – необхідна для налаштування анімації об'єкту.

6. **Робочий простір сцени** – те, що користувач буде бачити, коли відповідна сцена провантажить. Простір додатку, розроблений за допомогою BlipAR, є 3D-простором, а тому можна розміщати об'єкти на різних рівнях, формуючи таким чином тривимірний ресурс.

Зверху цього поля знаходиться редактор сцен (див. рис. 3) – видно назву сцени та можливість перемикавання між сценами в межах редактора за допомогою стрілок.

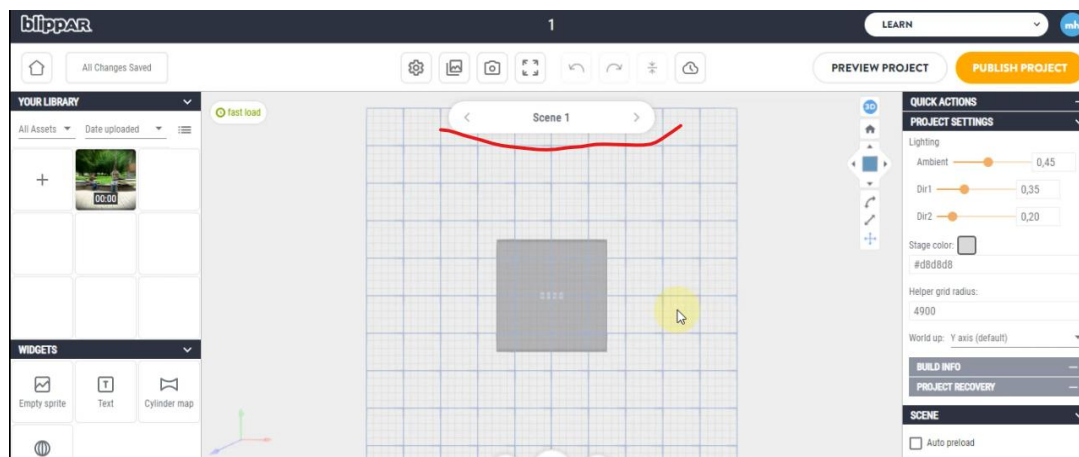


Рисунок 3 – Робота зі сценами

Якщо натиснути на назві сцени, то можна перейти у сам редактор сцен (див. рис. 4). Створити нову сцену можна наступним чином – треба натиснути у полі **Add New Scene**, дати назву сцені та натиснути **Enter**. Сцена, яка стоїть найвище буде першою зустрічати користувача. Для зміни порядку сцен можна або натискати на стрілочки ліворуч від назви (зміна порядку на один рівень), або зажати сцену ЛКМ та перетягнути її у відповідне місце. Сцену також можна скопіювати або видалити (з правої сторони від назви сцени). Сцена, з якою користувач працює зараз виділяється жовтеньким контуром. Переходити до роботи з іншою сценою можна або тут, обравши іншу сцену, або натиснути на стрілочку поруч з назвою сцени у робочому просторі (див. рис. 3)

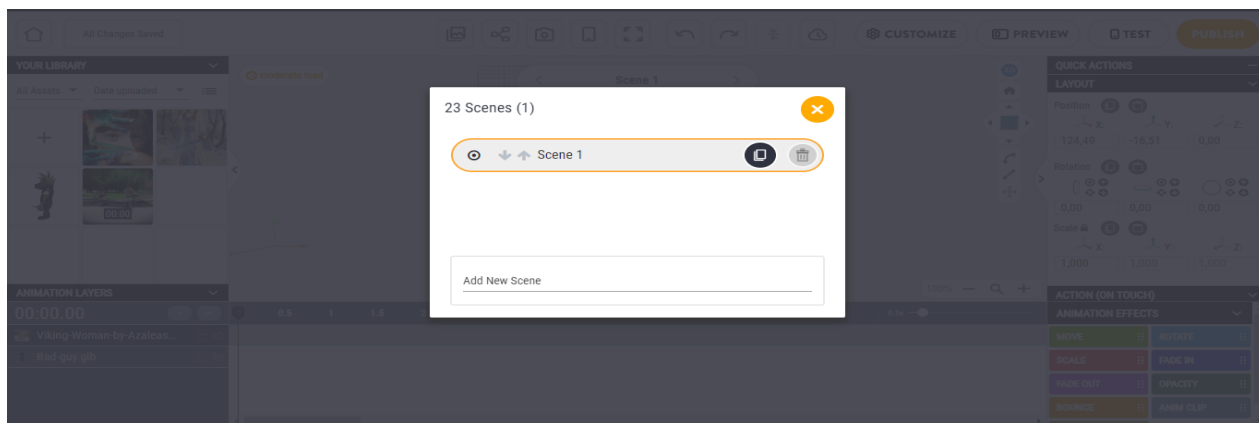


Рисунок 4 – Редактор сцен

У центрі поля робочого простору сцени знаходиться засвітлене зображення маркеру, якого не буде видно користувачеві (тому якщо дане зображення має бути присутнім у

додатку, то треба у робочий простір підвантажити теж саме зображення, що і для маркеру та перетягти його у робочий простір) та простір, що символізує поле перегляду об'єктів при використанні готового проєкту (всі елементи у цьому просторі незарізатимуться межами екрану телефону та будуть відповідного розміру для сприйняття). Проте загалом чітких меж розміщення об'єктів немає (для відслідковування результату виконання операцій чи розміщення елементів можна користуватися режимами тестування чи попереднього перегляду) – за ці межі можна виходити. У цей простір додаються шляхом перетягування необхідних об'єктів з бібліотеки або віджетів. Можна скільки завгодно раз витягати один й той ж самий об'єкт, просто перетягаючи його у робочий простір. Об'єкти також можна копіювати (ctrl+C/ctrl+V).

Із правої сторони (див. рис. 5) цього простору є інструменти налаштування об'єктів (переміщення, поворот, масштабування) та вибору профілю перегляду сцени (збоку, спереду, зверху тощо).

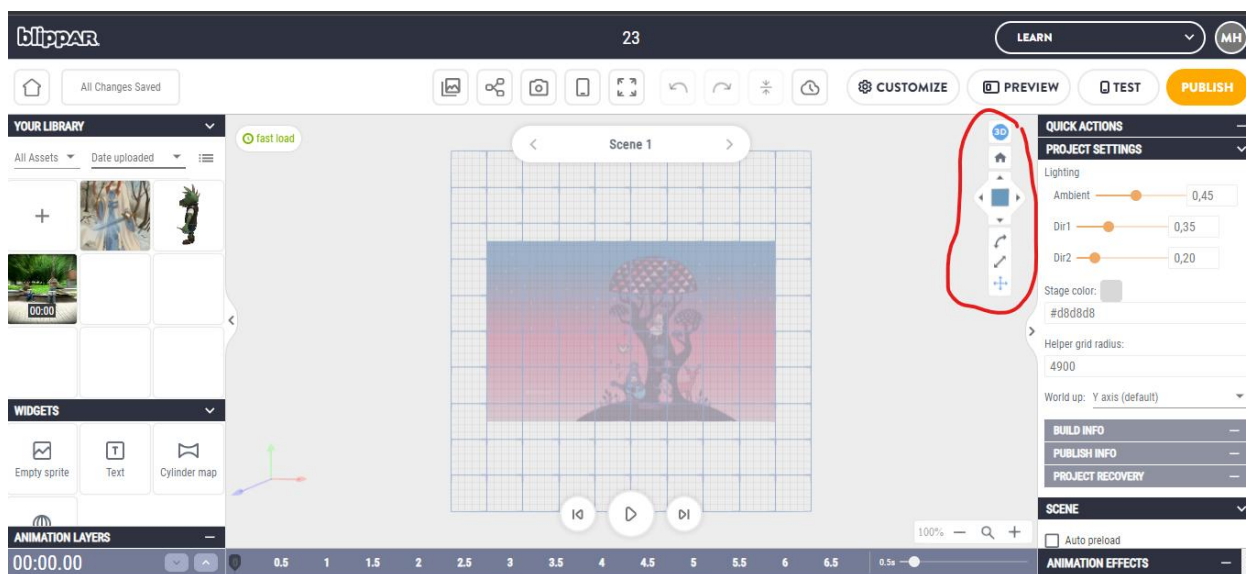


Рисунок 5 – Інструменти редагування положення об'єкту та всієї сцени загалом

7. Поле налаштування параметрів об'єктів – при виборі об'єкту у робочому просторі сцени в цьому полі можна підкорегувати параметри цього обраного об'єкту – масштабування, положення, кут повороту відносно осей X, Y, Z, параметри інтерактивності Action (задають поведінку об'єкта, якщо на нього натисне користувач) тощо, а також ті, що характерні для конкретних об'єктів (наприклад, для тексту характерними є параметри шрифту – гарнітура, кегль, колір тексту та його фону, накреслення, вирівнювання тощо)

8. Бібліотека анімацій від додатку (animation effects) – розкриваються разом із полем налаштування анімацій (5). Для того, щоб додати анімацію об'єкту, треба відкрити **Animation Layers** (5) та перетягнути на часову шкалу відповідного елемента необхідний ефект та налаштувати його зміну та тривалість (підвівши курсор до правої частини доданої анімації, затиснувши ЛКМ та потягнувши у відповідну сторону або налаштувавши характеристики, натиснувши ЛКМ на додану анімацію, у полі **Move settings**, що розкриється) (див. рис. 6). У кожного з ефектів є певна початкова зміна параметру, але його завжди можна налаштувати під себе. Для цього необхідно поставити курсор у відповідному місці ефекту і змінити характеристику, яку змінює дана анімація, формуючи таким чином ключову точку. Програма буде прораховувати як від однієї ключової точки перейти до іншої, створюючи таким чином певну анімацію зміни параметру. Наприклад, анімація Move змінює положення елемента, Rotate – кут повороту, Scale – масштаб тощо. **До одного об'єкту можна додавати скільки завгодно анімацій.** Вони можуть йти паралельно,

попередньо та з певною паузою між ними. Також кожній анімації можна задати певну затримку її початку.

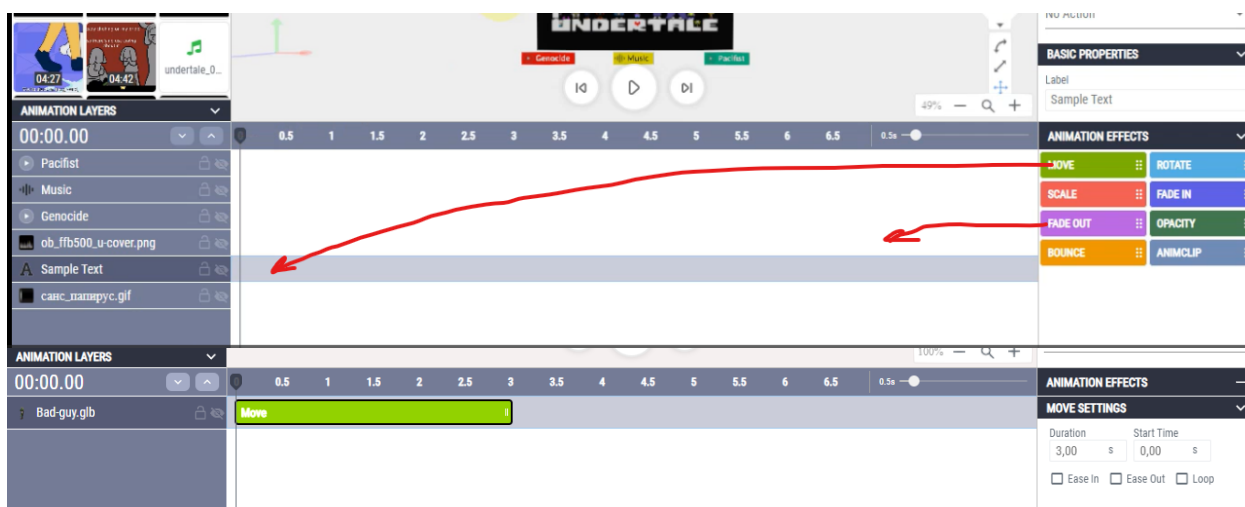


Рисунок 6 – Приклад налаштування анімації об'єктів

Також можна створити кнопки переходу між сценами, щоб користувач теж міг між ними перемикатися. Кнопки створюються шляхом задання зображенням або тексту характеристики **Action**, що перетворює цей елемент у кнопку. Кнопки є різні:

- **Audio** – при натисканні на цей об'єкт почне програватися задана музика
- **Video** – при натисканні на цей об'єкт почне програватися задане відео
- **Website** – перехід по посиланню
- **Call** – здійснити дзвінок
- **Email** – відправка повідомлення
- **Product Page** – перехід по посиланню на сторінку з товаром
- **Twitter** – перехід на сторінку твіттеру
- **Youtube** – перехід на Youtube
- **Go to scene** – кнопка переходу між сценами (при виборі цього варіанту треба буде прописати назву сцени, на яку відбудеться перехід, коли користувач додатку натисне на цей елемент).

3. Перевірка проробленої роботи

Як вже було описано вище, є декілька можливостей попереднього перегляду роботи:

1. Тестовий режим того, як розроблені елементи виглядатимуть на телефоні (**circle phone preview**) – чисто візуально без перегляду їх інтерактивності
2. Попередній перегляд роботи проєкту (**preview**) – попередній перегляд додатку з можливістю перегляду всіх сцен, переходів між ними роботи анімацій, програвання музики, роботою кнопок – тобто дає можливість перегляду та перевірки роботи додатку;
3. Тестування роботи проєкту з використанням згенерованого QR-коду (**test**) – тестування всіх етапів роботи AR-додатку (як це все відбуватиметься при реальному використанні додатку) – від розпізнавання маркеру доповненої реальності до взаємодії користувача з кожною сценою;

Кожен раз, коли буде потреба перегляду виконаної роботи, можна скористатися режимами тестування. Наприклад, можна скористатися режимом тестування з QR-кодом, щоб протестити повністю весь шлях розпізнавання маркеру. Для цього треба натиснути на кнопку **Test** у правій верхній частині вікна редактора (біля кнопки **Publish**). Далі вебресурс перекидає у нове вікно як на рисунку 7.

PREVIEW PROJECT

Scan the QR code below using your smart device.



Then point your device camera at the marker →
For the best AR experience, scan a printed marker.

Preview expires in:
19m 39s

Your PREVIEW project URL is:
<https://ar.blippar.com/556789070>

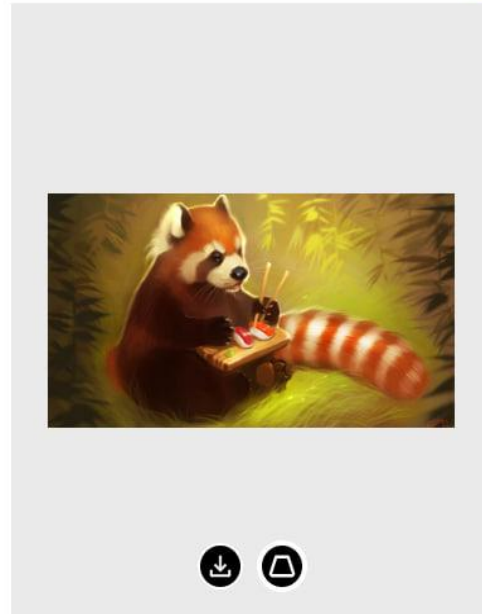


Рисунок 7 – Приклад меню тестування

Далі треба відсканувати QR-code та перейти по посиланню. Після завантаження елементів відкриється вікно у браузері як на рисунку 8.

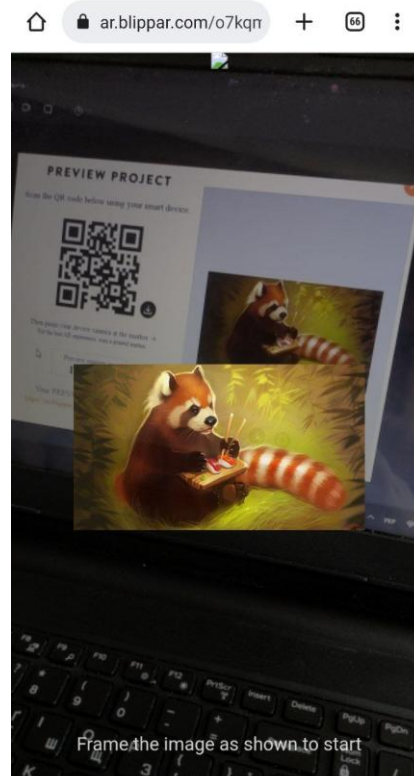


Рисунок 8 – Меню у браузері

Далі потрібно зображення, що з'являється на екрані телефону накласти на зображення на екрані монітору чи ноутбуку у тому вікні, де було проскановано QR-код і далі екрані телефону буде програватися зашифрований контент.

Хід роботи

1. Визначитися з концепцією та метою створення додатку доповненої реальності.
2. Розпланувати наповнення сцен додатку – додаток має містити мінімально 3 сцени.
3. Визначити цільову аудиторію додатку та особливості сприйняття нею інформації, а також шляхи розповсюдження додатку (наприклад, додаток буде для книжок відповідного до задуму направлення або для пакування, або ж для розповсюдження на сайтах тощо)
3. Створити додаток у середовищі бліппару – створити сцени, їх наповнення, додати анімацію до об'єктів (мінімальна кількість анімацій для усього проєкту – 2-3 анімованих об'єкти). За потреби протестити виконання пророблених маніпуляцій за допомогою тестових режимів. Слід не забувати про логіку викладу інформації, щоб користувач міг зрозуміти та за потреби провзаємодіяти із інтерактивними елементами – щоб він розумів що це, наприклад, кнопка і на неї можна натиснути.
4. Створити протокол, у який занести опис проробленої роботи, описати задум та концепцію проєкту, навести скріни проробленої роботи, опис сцен, зображення маркер. Зробити скріни з тестового режиму готового додатку. Зробити висновки стосовно проведеної роботи.
5. Записати додатково відео роботи маркеру під час його тестування.

Контрольні запитання:

1. Що таке доповнена реальність?
2. Які є види доповненої реальності та в чому між ними різниця?
3. Чим доповнена реальність відрізняється від віртуальної?
4. Які переваги та недоліки веб-ресурсу blippar?
5. Як створюються сцени у середовищі blippar?
6. Як створити анімаційні елементи у середовищі blippar?

Лабораторна робота №2

**РОЗРОБЛЕННЯ ДЕКОРАТОРУ КІМНАТИ
ЗА ДОПОМОГОЮ BLIPPAR**

Мета роботи: вивчення особливостей створення подій та реакцій для об'єктів для додатків із доповненою реальністю. Створення декоратору кімнати за допомогою веб-ресурсу blippar.com

Теоретична частина

Доповнена реальність – технологія, що дозволяє впровадити віртуальні об'єкти (текст, ілюстрації, анімація, аудіо, відео тощо) у реальний світ людини шляхом сканування спеціального об'єкту-маркеру, що роздрукований на будь-якому задрукованому матеріалі чи відтворений на екрані монітору, за допомогою камери смартфона чи планшета чи будь-якого іншого пристрою з камерою та подальшого відтворення цифрового контенту на екрані цього пристрою.

Найбільш популярними видами даної технології є маркерна та безмаркерна. Даний практикум буде присвячений розробці елементу доповненої реальності, що базується на безмаркерній технології. У даному випадку маркер-зображення все таки є. Він являє собою складне деталізоване зображення, проте основою для розпізнавання слугують різноманітні дані – положення користувача у просторі, його геолокація, кут повороту пристрою тощо.

Одним із видів програмного забезпечення, що застосовується для створення безмаркерної доповненої реальності є **онлайн-платформи**. Вони дозволяють легко об'єднувати між собою віртуальні об'єкти та самостійно їх зашифровують у елемент доповненої реальності. Вони дають змогу працювати з певним набором безкоштовних функцій та певною кількістю «маркерів». Проте для більш деталізованого додатку слід активувати платну версію. До таких додатків можна віднести **BlippAR, Vuforia, RoAR, PlugXR** тощо. У даному практикумі будуть розглянуті принципи роботи на платформі The RoAR.

Середовище **BlippAR** є сучасним ресурсом для простого створення об'єктів доповненої реальності за допомогою інтерфейсу користувача. Середовище є простим та інтуїтивним і активно оновлюється та доповнюється новими можливостями та функціоналом, завдяки чому дозволяє реалізовувати складні та креативні ідеї та додавати інтерактивності друкованим матеріалам – зовнішній рекламі, постерам, книгам тощо.

Особливості виконання

Дана робота буде присвячена розробленню декоратору кімнати, що базується на використанні доповненої реальності, що використовує варіант технології доповненої реальності як а Real AR, який полягає у тому, що маркеру як такого немає, але при цьому розпізнаються як тригерні елементи об'єкти реального світу, зокрема поверхні, на які системою розміщуються відповідні цифрові об'єкти при розпізнаванні цих об'єктів.

Спочатку при створенні нового проєкту у BlippAR після задання його назви з'явиться нове меню, у якому треба обрати один з наступних варіантів – Surface або Around You (див. рис. 1).

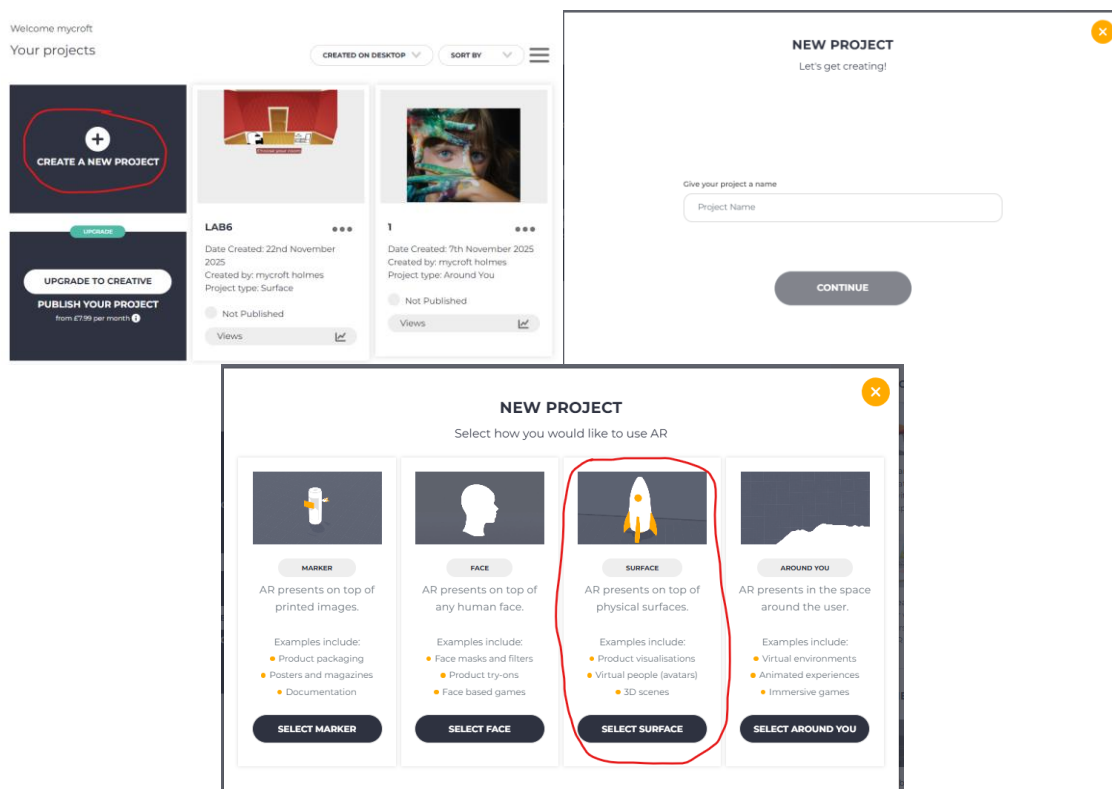


Рисунок 1 – Створення проєкту

Проект, що базується на методі **Surface**, дозволить сканувати плоску поверхню (підлога, стіл, стілець, книжка тощо) та розставляти на ній об'єкти, зашифровані у вигляді доповненої реальності. Проект, що базується **Around You** дозволяє розставляти об'єкти у загальному просторі людини без сканування та розпізнавання плоских поверхонь, створюючи певним чином схожий до віртуальної реальності простір.

Далі необхідно наповнити проєкт відповідними запланованими об'єктами, зокрема 3D-об'єктами, використовуючи ліву область меню редактора, де знаходяться вкладки з об'єктами, які можна додавати у проєкт (див. рис. 2).

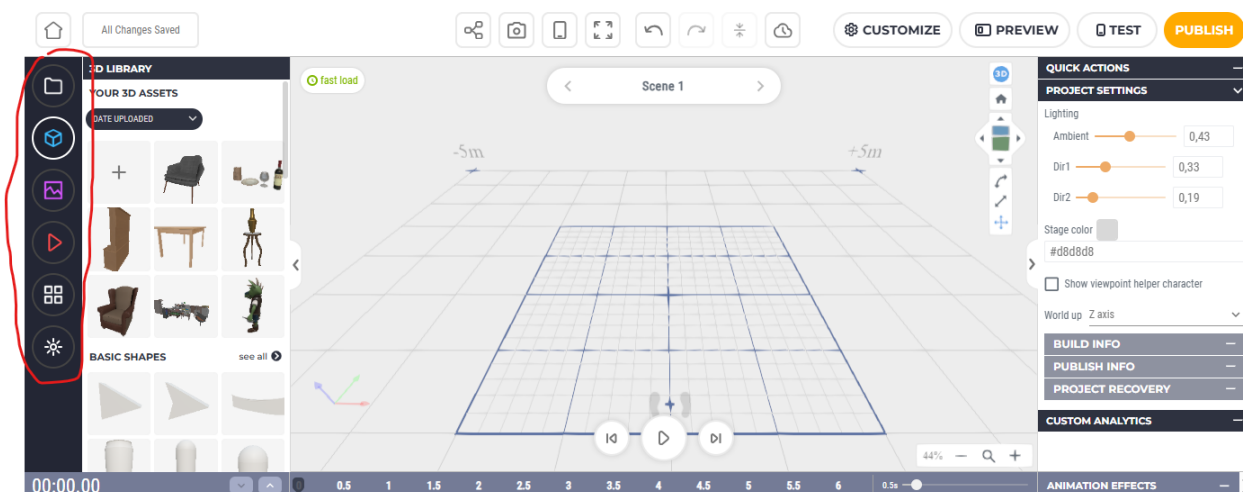


Рисунок 2 – Меню об'єктів

Налаштовувати параметри цих об'єктів можна у правій частині редактора, обравши відповідний об'єкт.

Для роботи необхідно буде створити кнопки, які додаватимуть відповідний 3D-об'єкт у сцену. Для цього необхідно додати необхідний 3D-об'єкт у сцену та кнопку, налаштувати їх параметри. Далі перейти у меню **Toggle Actions and Events Creator** (див. рис. 3). У цьому меню можна налаштовувати поведження об'єктів, якщо на якийсь об'єкт натиснуто (**On Tap**) або ж коли сцену завантажено (**On Scene Start**).

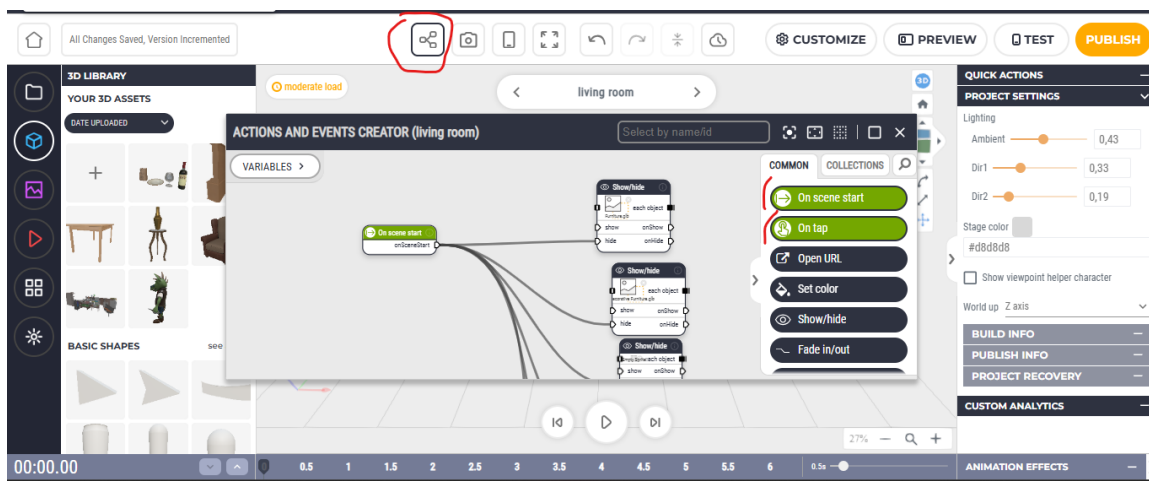


Рисунок 3 – Меню **Toggle Actions and Events Creator**

Далі треба створити два стани – один для **On Tap** та один для **On Scene Start**. Для цього треба на них натиснути і вони з'являться у вільному полі. Далі треба додати якусь дію, яка відбудеться при цьому – у даному випадку **Show/Hide** – теж натиснувши на ній (див. рис. 4).

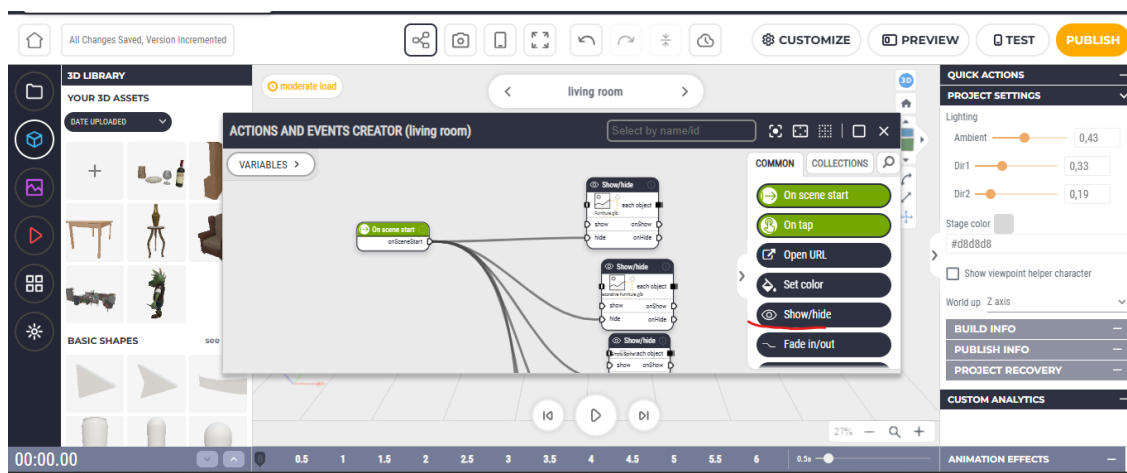


Рисунок 4 – Дія Show/Hide

Після додавання треба зробити зв'язок.

Спочатку треба від стану **On Scene Start (onSceneStart)** протягнути лінію, затиснувши відповідний трикутник біля стану **onSceneStart**, до відповідного трикутника у лівій частині дії **Show/Hide** (якщо треба, щоб об'єкт невідображався при завантаженні сцени, то до трикутника біля **hide**). Також потрібно обрати об'єкт у дії **Show/Hide** – 3D-об'єкт (див. рис. 5).

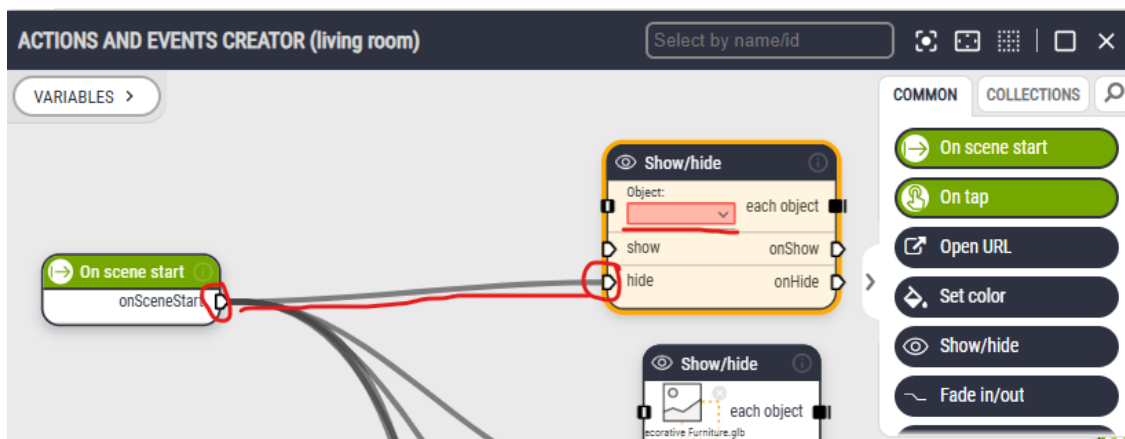


Рисунок 5 – Створення зв'язку

Далі потрібно створити додавання об'єкту при натисканні на кнопку (див. рис. 6). Для цього треба для стану On Tap створити зв'язок до дії **Show/Hide** – від стану **On Tap/onTap** до дії **Show/Hide / show**. У стані **On Tap** треба обрати в об'єкті кнопку, а в дії **Show/Hide** – той же **3D-об'єкт**. У стані **On Tap** має стояти галочка **enabled**, щоб кнопка працювала.

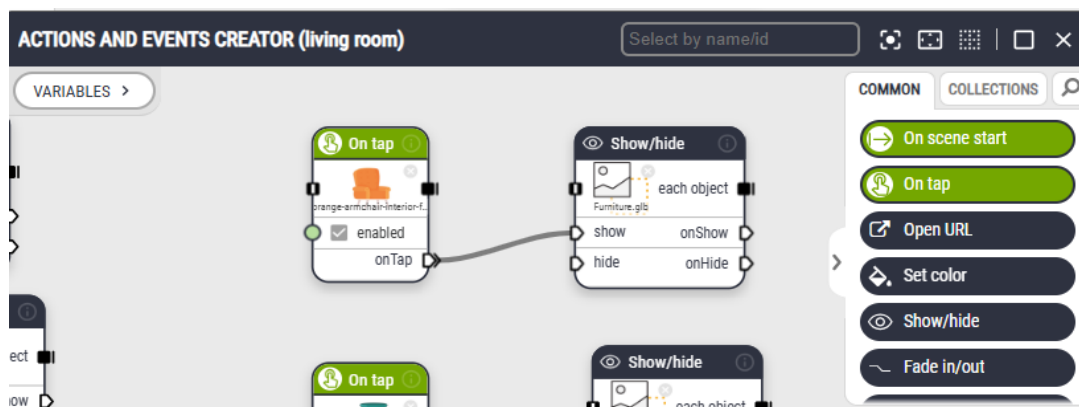


Рисунок 6 – Прив'язування об'єкту до кнопки

Це меню запам'ятовує попередні дії і може при створенні повторної дії чи стану підв'язати об'єкт з попереднього стану чи дії. Для вибору нового об'єкту треба натиснути на хрестик біля об'єкту і потім можна обрати новий об'єкт.

Також можна подібним чином створювати випадаюче меню, тільки з більшою кількістю об'єктів для взаємозв'язків.

Хід роботи

1. Створити додаток, який матиме головне меню та мінімум дві сцени, де буде можливість розставляти 3D-меблі або інші об'єкти по кімнаті, з можливістю переходу між сценами та переходу до головного меню.
2. Створити у сценах кнопки, які дозволять додавати об'єкт до сцени.
3. Створити випадаюче меню по кнопці.
4. Створити протокол з результатами проробленої роботи.

Контрольні запитання:

1. Алгоритм створення кнопки, що додає об'єкт до сцени
2. Алгоритм створення випадаючого меню у BlipAR.
3. Алгоритм створення кнопок переходу між сценами.

РОЗРОБЛЕННЯ ІНТЕРАКТИВНОГО ПОСТЕРУ З ЕЛЕМЕНТАМИ ДОПОВНЕНОЇ РЕАЛЬНОСТІ

Мета роботи: вивчення особливостей різних дій та подій у середовищі VliprAR, створення інтерактивного постера чи картини з можливостями вебресурсу

Теоретичні відомості

У вебресурсі VliprAR є різноманітні стани для об'єктів, які можна задавати для виконання ними додаткових дій при взаємодії з ними або зі сценою загалом. На рисунку 1 наведено меню станів та дій, що можливі до застосування для об'єктів сцени.

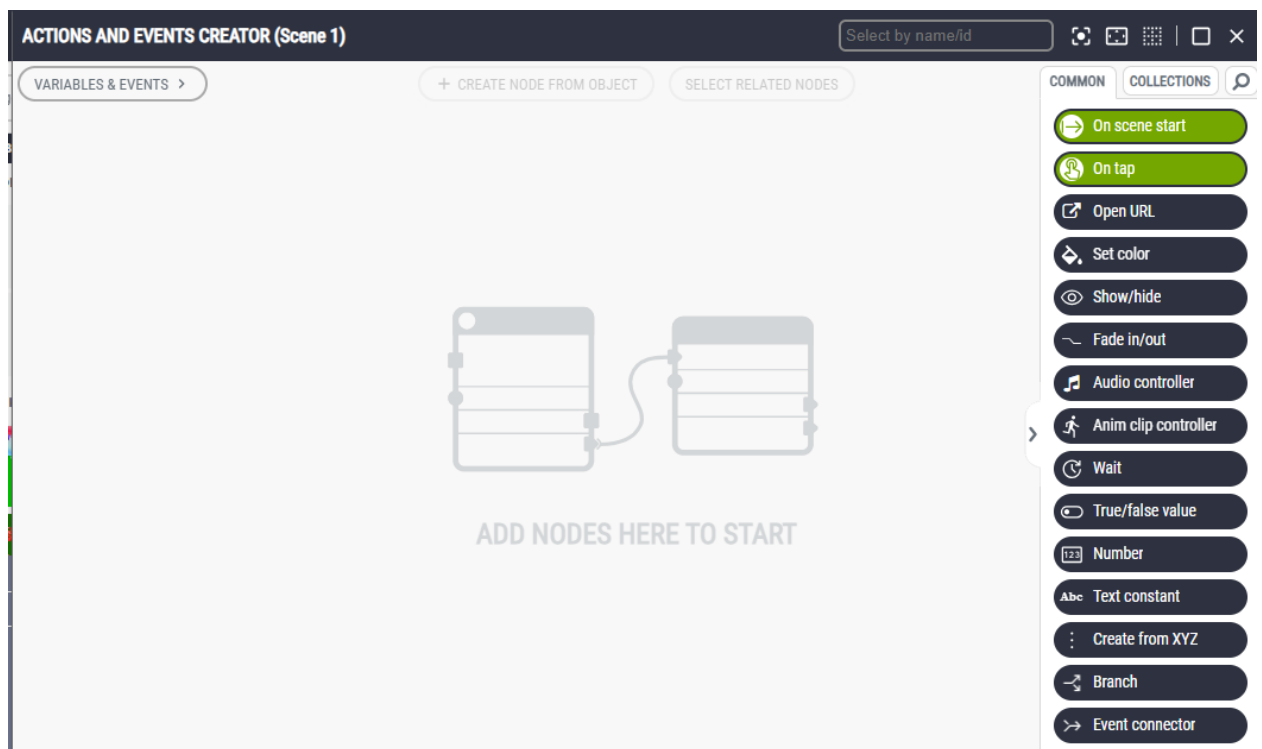


Рисунок 1 – Меню дій та станів

Існує два моменти взаємодії з об'єктами, коли з ними може відбутися якась дія – коли на них натиснуто (On tap) та коли сцена тільки завантажилася (On scene Start). Наприклад, можна зробити так, щоб об'єкти були невидимі при завантаженні сцени, а потім якщо натиснути на якусь кнопку, то вони стають видимими.

Кожна дія складається в загальному з трьох частин, що наведені на рисунку 2.

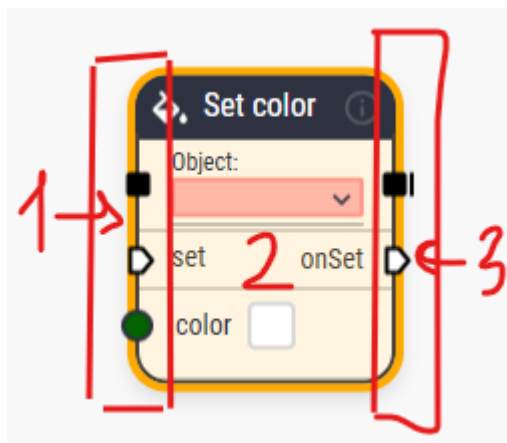


Рисунок 2 – Приклад оформлення дії

Зокрема можна виділити наступні зони (див. рис. 2):

1. Зона вхідних умов – в загальному містить два блоки – вхід самої дії (сюди необхідно побудувати зв'язок із іншою дією чи подією On tap чи On Scene Start), який має форму **білого п'ятикутника** та має назву відповідно до обраної дії (якщо дія одночасно може мати два варіанти виконання, наприклад, видимість об'єкта (видимий або невидимий), то таких входів буде два – один на один варіант виконання дії, інший – на другий) та вхід якоїсь характеристики, яку можна задати у **зоні 2** у залежності від обраної дії, який має форму **кольорового кола** (в залежності від дії може бути декілька параметрів, для кожного свій вхід). Також, якщо дія пов'язана з об'єктом, то є додатковий вхід для задання об'єкта (має форму **чорного квадрата**).

2. Зона задання параметрів дії – дана зона відрізняється для різних дій, але в загальному тут можна задавати різні параметри для дій – обирати об'єкт, над яким проводитиметься дія (із наявних у сцені, що можна обрати у випадаючому списку), прописувати url, задавати колір тощо.

3. Зона вихідних умов – ця зона містить виходи такої ж форми, що й входи, від яких можна побудувати зв'язки до входів інших дій чи подій. У дій, що пов'язані із змінами об'єктів є вихід у формі квадрата, тобто ці дії можуть стати об'єктами для інших дій – від виходу у формі квадрата із **зони 3** таких дій можна провести зв'язок до входу у формі квадрата у **зоні 1** інших дій. Також для всіх дій є вихід у формі білого п'ятикутника із назвою відповідної дії (якщо у дії два таких входи у зоні 1, то буде два виходи у зоні 3 під кожен запланований варіант розвитку подій), тобто кожна дія може бути подією, яка буде причиною виконання іншої дії.

У загальному пов'язувати можна входи і виходи **ОДНАКОВОЇ** форми та кольору.

Зв'язок створюється шляхом або натискання ЛКМ на виході однієї дії і потім вході іншої (при цьому між ними створюється лінія), або затискання ЛКМ на виході однієї дії та перетягуванням курсору до відповідного входу іншої (після цього курсор треба відпустити). Щоб видалити зв'язок, треба видалити дію чи подію, від або до якої він йде.

До дій, які можна застосувати до об'єктів, можна віднести наступні:

1. Open URL – відкриття певного посилання, яке додається у відповідному рядку цієї дії (див. рис. 3). Єдиним параметром, що можна задати у цій дії, є рядок, де можна прописати посилання сторінки, на яку можна буде перейти, наприклад, якщо натиснути на об'єкт чи кнопку.

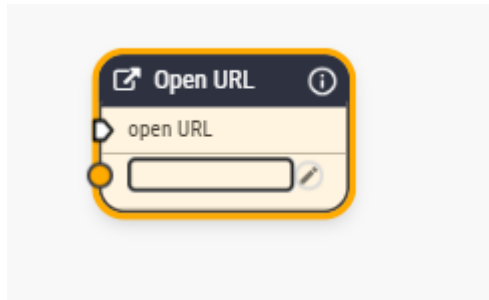


Рисунок 3 – Дія Open URL

2. **Set Color** – об’єкт (всі його текстури та матеріали) перефарбовується системою у обраний колір, наприклад, при натисканні на об’єкт чи на спеціально розроблену кнопку (див. рис. 4). У цієї дії можна задати об’єкт, із яким відбуватиметься дія, та колір, у який об’єкт перефарбується.

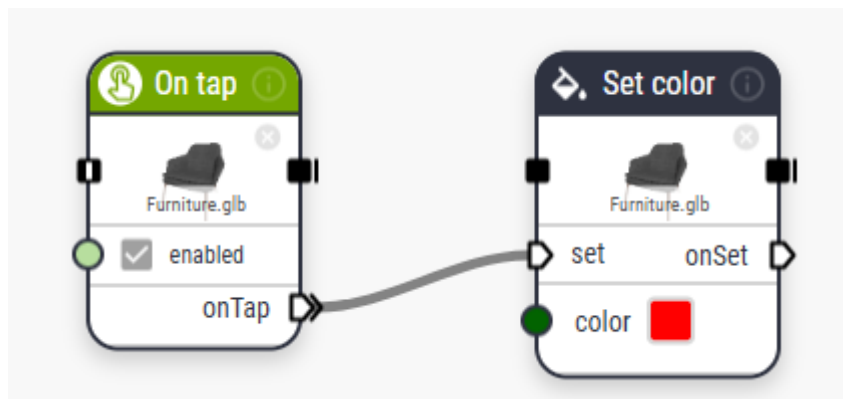


Рисунок 4 – Дія Set Color

У випадку, як продемонстровано на рисунку 4, об’єкт Крісло змінить колір на червоний, якщо користувач на нього натисне.

3. **Show / Hide** – зміна видимості об’єктів при натисканні на них або інші об’єкти, наприклад, кнопку або ж при завантаженні сцени (див. рис. 5). У цьому випадку можна задати лише об’єкт, що буде видимим чи невидимим в залежності від обраного входу при побудові зв’язку.



Рисунок 5 – Дія Show / Hide

У випадку, як продемонстровано на рисунку 5, об'єкт Крісло буде показуватися користувачеві, якщо він натисне на цей же об'єкт.

4. Fade in / out – плавне проявлення або зникнення об'єкта за певний проміжок часу. У даному випадку можна задати об'єкт, що проявлятиметься або зникатиме у залежності від обраного входу для зв'язку, та час, за який ця дія відбуватиметься (див. рис. 6).

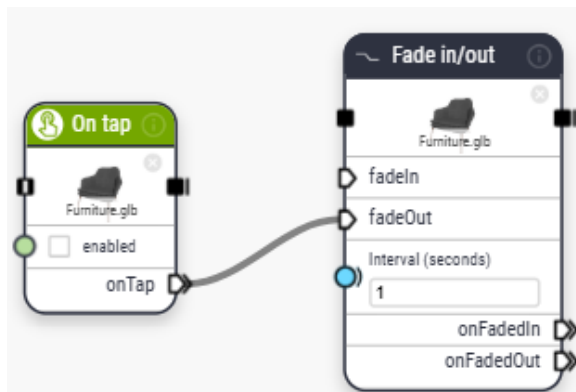


Рисунок 6 – Дія Fade in / out

У випадку, як продемонстровано на рисунку 6, якщо користувач натисне на об'єкт Крісло, то воно за 1 секунду має плавно зникнути.

5. Audio Controler – будь-який об'єкт може контролювати параметри аудіоінформації, що завантажене в систему (див. рис. 7). Для цієї дії як вже було зазначено обов'язково треба підвантажити аудіоінформацію у бібліотеку редактора. Для неї можна задати об'єкт, що контролюватиме музику, сам аудіотрек, гучність та зацикленість програвання за допомогою чекбоксу. Об'єкт може почати програвати звук, поставити його на паузу або ж припинити програвання звуку.

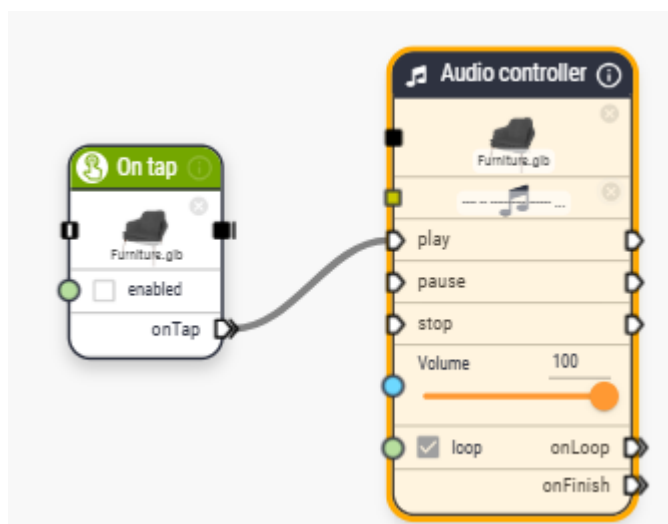


Рисунок 7 – Дія Audio Controler

У випадку, як продемонстровано на рисунку 7, якщо користувач натисне на об'єкт Крісло, то музика, задана у дії, почне програватися із гучністю 100% та буде зацикленою.

6. Anim Clip Controller – при використанні цієї дії об'єкт контролює програвання вбудованої анімації себе ж або іншого об'єкта (див. рис. 8), тобто тих, що розроблені при створенні об'єкта в іншій програмі, наприклад, при створенні анімованого 3D-об'єкта у Blender. Ця дія не контролює анімації, що є в редакторі, тобто якщо до об'єктів у відповідному меню знизу додати анімацію, то дана дія ці анімації **НЕКОНТРОЛЮЄ**. У даному випадку для цієї дії можна задати об'єкт, що має анімацію, саму анімацію, що контролюватиметься (обрати з випадаючого списку у параметра **Clip** якусь конкретну чи всі) та заикленість анімації. Об'єкт, що буде виступати у якості контролера цієї анімації, може програвати цю анімацію або ж зупиняти її відтворення.

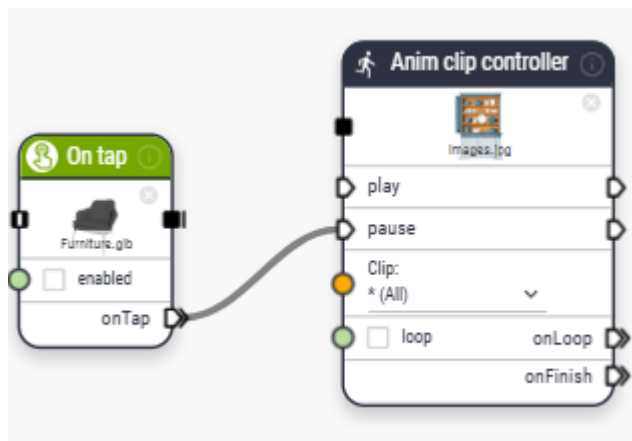


Рисунок 8 – Дія Anim Clip Controller

У випадку, як продемонстровано на рисунку 8, якщо користувач натисне на об'єкт Крісло, то об'єкт Зображення (якби у нього була вбудована анімація) став би статичним, оскільки б усі його анімації перестали б програватися.

7. Додаткові допоміжні функції – не є самостійними діями, але є доповненням до інших дій для побудови більш складних та автоматизованих зв'язків (див. рис. 9).

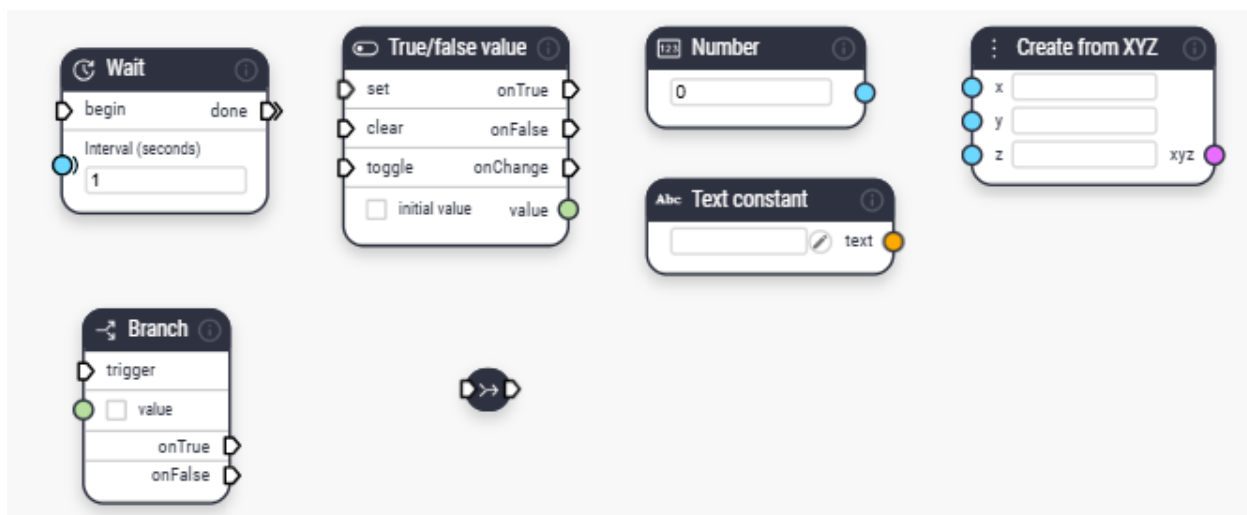


Рисунок 9 – Допоміжні функції

На рисунку 9 представлені наступні допоміжні функції:

- **Wait** – функція, яка дозволяє задати затримку для виконання якоїсь із дій.

- **True/False value** – дозволяє задати умову перевірки виконання якоїсь дії за допомогою булевих операцій на істинність (**Set** – істинно, **Clear** – хибно, **Toggle** – змінює значення на протилежне). Також ця функція має параметр **initial value** – початкове логічне значення при запуску сцени. Якщо встановлено галочку – значення стартує як **True**, якщо ні – як **False**. Тобто можна задати, що поки кнопка, наприклад, не натиснута, то виконуватиметься якась одна дія, наприклад, показ об'єкту, яка закладена, наприклад, в параметр Істинно (вихід OnTrue), якщо ж натиснути на кнопку, то виконуватиметься дія, яка закладена в параметр Хибно, наприклад, показ іншого об'єкту (вихід OnFalse). Таким чином можна формувати більш складні зв'язки. Є ще вихід OnChange – цей вихід програє закладену в нього дію при будь-якій зміні логічного значення істинності.
- **Number** – функція, що дозволяє числове значення, що можна використовувати для інших дій.
- **Text constant** – теж саме, але можна задати якийсь текст.
- **Create from XYZ** – дозволяє задати координати вектора та потім передати отримані значення до інших дій, що можуть використати цей вектор для виконання.
- **Branch (розгалуження)** – ця функція перевіряє логічне значення (True або False) і запускає одну з двох гілок подій. Наприклад, якщо музика є ввімкнутою, то система її вимкне і, навпаки, якщо вона вимкнута, то ввімкне.
- **Конектор**  - необхідний для зв'язку між діями і подіями. Він використовується, щоб передати виконання від однієї дії до іншої; зробити схему більш акуратною (особливо якщо стрілки перетинаються); розбити довгий ланцюжок на логічні частини.

Є також крім базових вузлів подій, дій та функцій (вкладка COMMON) колекція шаблонів у сусідній вкладці у цьому ж меню (вкладка COLLECTIONS), що наведено на рисунку 10.

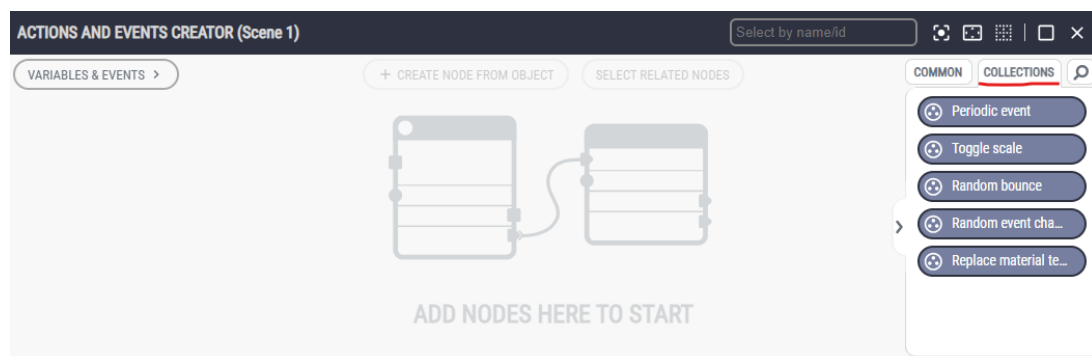


Рисунок 10 – Меню Дій та подій

Це вже набір готових шаблонів, які можна використовувати для своєї сцени. Вони мають початкові вузли та зв'язки між ними, а далі вже користувач може налаштовувати їх під себе та додавати відповідні дії та події. Зокрема тут є наступні шаблони:

1. Periodic event – запускає подію чи дію через певний інтервал часу. Наприклад, кожні 2 секунди відтворювати анімацію, кожні 5 секунд змінювати текст, регулярно перевіряти якусь умову тощо.

2. Toggle scale – перемикає масштаб об'єкта між двома значеннями. Наприклад, було 100% – стало 120%, наступне натискання – знову 100%. Часто використовують для збільшення кнопки при натисканні, ефекту «виділення»; відкриття/закриття об'єкта.

3. Random bounce – створює ефект випадкового підстрибування. Об'єкт трохи підстрибує, висота й час можуть бути випадковими, рух виглядає більш менш природнім. Добре підходить для предметів, які мають привертати увагу.

4. Random event chance – запускає подію лише з певною ймовірністю. Наприклад при натисканні на кнопку може відбутися якась дія, а може й не відбутися і ймовірність того, що дія відбудеться складатиме 40%. Цей шаблон можна використати, наприклад, якщо треба створити якийсь елемент несподіванки.

5. Replace material texture – замінює текстуру матеріалу 3D-об'єкта. Сам 3D-об'єкт не змінюється – змінюється лише його текстура (матеріал).

Хід роботи

1. Обрати зображення для інтерактивної картини чи постеру. Описати цільову аудиторію та призначення розроблюваного мультимедійного продукту

2. Наповнити сцену чи сцени необхідним контентом, протестувати результат.

3. Застосувати мінімум 3 дії з меню Дій та Подій редактора, а також мінімум один шаблон до об'єктів

4. Створити протокол з результатами проробленої роботи.

Контрольні запитання:

1. Які існують події, що можуть впливати на об'єкти сцени?

2. Які існують дії, що можуть впливати на об'єкти сцени?

3. Які існують функції, що можуть впливати на дії та події та їх виконання?

4. Які шаблони є в системі BlippAR, що можна використати для впливу на об'єкти на сцені?

5. Як створити зв'язки між діями та подіями?

СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ У СЕРЕДОВИЩІ PROCESSING

Мета роботи: вивчення основ розроблення інтерактивних проєктів з об'єктами доповненої реальності у середовищі Processing за допомогою мови програмування java.

Теоретична частина

Маркер доповненої реальності – це основа для використання маркерної технології. Це чорно-біле або кольорове просте або деталізоване зображення, при наведенні на яке камери скануючого пристрою відбувається відтворення віртуальних об'єктів. Маркер може бути надрукований або ж розміщений в електронному вигляді на сторінці сайту. Маркери бувають різні.

Одним із найпоширеніших є **простий маркер**, оскільки його легко створювати та вводити у контент реальної продукції. Є ряд характеристик, яким має відповідати такого роду маркер: він має бути чорно-білим, інваріантним до поворотів, квадратним та мати товсту чорну рамку, яка займатиме 10% його площі. Основні алгоритми розпізнавання, застосовувані у даному випадку – це розпізнавання кутів, країв, точок та контурів. Цей вид маркеру має найбільші показники стабільності та легкості при використанні.

Наступний найбільш застосовуваний вид маркеру є так званий «**безмаркерний**» тип. Не дивлячись на назву маркер все-таки є – це деталізоване зображення. Воно має мати велику кількість деталей і бути дуже якісним, кольоровим та інваріантним до поворотів. Розпізнавання базується на зміні градієнту, розпізнавання кутів зображення або ж локаційних координатах. До основних причин широкого застосування можна віднести легкість введення таких зображень у реальний світ через відсутність необхідності створення чорної рамки навколо зображення.

Комбінація маркерів застосовується для технології, що базується на розпізнаванні 3Д-об'єктів, наприклад для представлення продукції у реальних умовах. Тобто спочатку створюється 3Д-модель об'єкту, потім розробляється код, і при наведенні камери на реальний об'єкт відтворюється контент.

Frame Marker застосовується для шифрування логотипів чи слоганів компанії для їх індивідуальної ідентифікації. Такі маркери мають бути маленькими, інваріантними до поворотів, квадратними, уніфікованими та мати контраст між рамкою та наповненням.

До основних параметрів класифікації маркерів можна віднести наступні: основа для розпізнавання, метод розпізнавання, колір маркерів, складність рисунку маркеру, сфера використання тощо.

Особливості виконання

Мета роботи полягатиме у тому, щоб за допомогою програмного забезпечення Processing з використанням додаткової бібліотеки створити елемент доповненої реальності. Цей елемент складатиметься з маркеру доповненої реальності та його цифрового наповнення. Цифрове наповнення – це перші літери ПІБ студента, де кожна літера сформована за допомогою кубиків. Також можна додати функцію обертання до літер.

Для спрощення формування літер у цифровому вигляді за допомогою кубиків можна розробити макет на папері для прорахунку необхідної кількості кубиків та їх розташування. Кількість та розміщення кубиків в межах однієї літери – на розсуд студента. Слід керуватися коректністю формування літер та їх впізнаваністю у такому вигляді. Приклад наведений на рисунку 1.

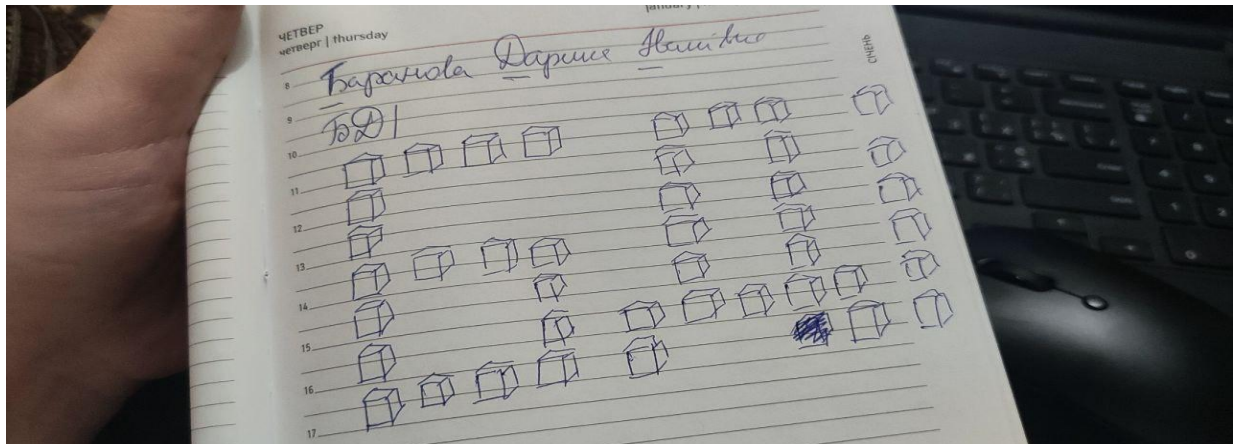


Рисунок 1 – Схема літер

Загалом весь процес розроблення елементу доповненої реальності можна поділити на три етапи – розроблення зображення-маркеру, розроблення коду додатку та його тестування і внесення коректив за потреби.

1. Розроблення зображення-маркеру доповненої реальності

Для роботи знадобиться простий маркер. Цей маркер має бути квадратним та чорно-білим. Він складається з товстої та помітної чорної рамки (чітких меж розміру рамки немає, проте вона має займати не менше 10% площі маркеру та при цьому має бути помітним рисунком маркеру в її межах) та рисунку маркеру, що складається з білих та чорних пікселів (фон та фігури / текст / векторні елементи / їх поєднання відповідно) (див. рис. 2).

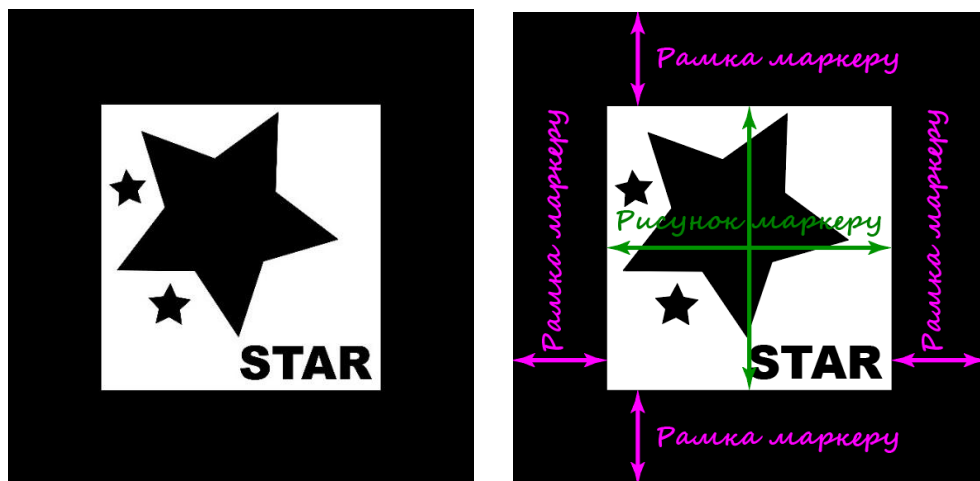


Рисунок 2 – Складові частини маркеру

Рисунок маркеру має бути не симетричними (тобто коло, наприклад, у якості темної частини рисунку маркеру не підходить або поєднання та розміщення елементів теж не повинно формувати симетрію). Ця несиметричність рисунку забезпечує інваріантність маркеру до поворотів, що дозволяє камері зчитувати їх під різними кутами. У такому

варіанті центр фокусування для камери під різними кутами не змазується, що забезпечує стабільність процесу розпізнавання.

Розмір маркеру можна обирати будь-який. Рекомендований мінімальний розмір маркеру для розпізнавання – 20-25 мм.

Приклади працюючих маркерів наведені на рисунку 3.

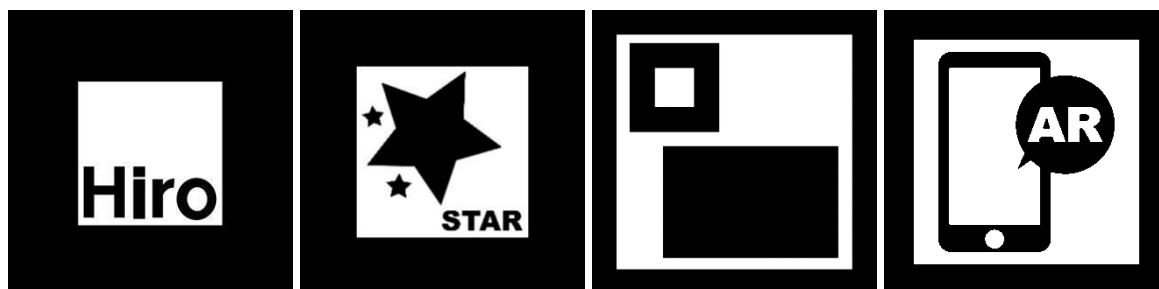


Рисунок 3 – Приклади маркерів

Далі маркер треба створити та зберегти у форматі **.jpg**. Якщо для тестування результату маркер буде друкуватися, то треба зберегти його у колірній системі CMYK, якщо ж маркер буде демонструватися з екрану смартфона – у колірній моделі RGB.

*Маркер краще всього створювати у векторному редакторі, наприклад, Adobe Illustrator, проте студент може скористатися будь-якою зручною програмою, яка дозволяє зберігати зображення у форматі .jpg. У випадку використання Adobe Illustrator їх треба експортувати у цей формат (**File / Export** та у новому вікні що з'явиться обрати .jpg)*

2. Створення коду елементу доповненої реальності

Спочатку треба встановити необхідні файли. Для цього треба:

1. Завантажити файл архіву «**Processing.zip**» (у цьому архіві міститься ще 2 архіви – один самого «**Processing**» та другий спеціальної бібліотеки для роботи з доповненою реальністю «**nyar4psg**»).

2. Створити папку та розархівувати «**Processing.zip**» у цю папку (слід прослідкувати, щоб у назві папки та у шляху до неї **НЕ БУЛО** кириличних символів).

3. Розархівувати «**processing-2.2.1-windows64.zip**» (містить папку «**processing-2.2.1**» – так папкою і розархівувати і так прослідкувати, щоб назві папки та у шляху до неї **НЕ БУЛО** кириличних символів).

4. Розархівувати «**nyar4psg-2.0.0.zip**» (у архіві папка «**nyar4psg**» – так папкою і розархівувати і так прослідкувати, щоб назві папки та у шляху до неї **НЕ БУЛО** кириличних символів).

5. Скопіювати папку «**nyar4psg**» в **processing-2.2.1\modes\java\libraries**

Важливо! Якщо невідключена, або відсутня веб-камера, то бібліотека не працюватиме.

Далі треба відкрити **Processing.exe** у папці «**processing-2.2.1**» – це версія **Portable**, а тому не потребує встановлення. Чисто зовнішньо робочий простір програми схожий на блокнот з можливістю запуску попереднього перегляду прописаного коду, вибору мови програмування для перевірки програмою синтаксису та поля для перегляду повідомлень програми про можливі помилки при запуску тестового режиму (див. рис. 4).

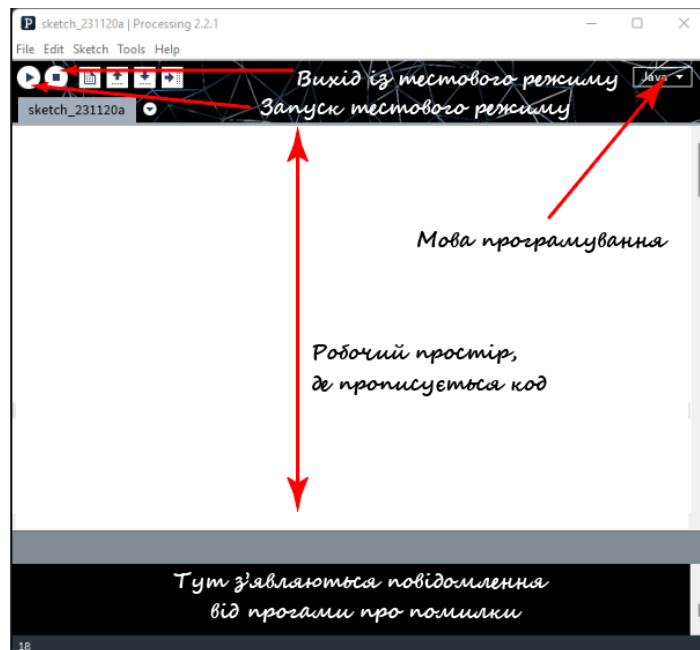


Рисунок 4 – Робочий простір програми

Основні 2 моменти при написанні коду – це не забувати ставити крапку з комою після кожної операції (крім рядків, що задають опис якоїсь підпрограми (що починаються з функції **void**) або початок умов (**if**) чи циклів) та закривати дужки – це дозволить зменшити можливість появи повідомлення про помилку. При збереженні проєкту слід використовувати **ЛИШЕ ЛАТИНИЦЮ** в назві та перевіряти, щоб у шляху до папки проєкту не було кирилиці. Також латиницю слід використовувати, називаючи файл маркеру.

У загальному будь який код складається з наступних частин:

1. підключення необхідних бібліотек;
2. Оголошення необхідних глобальних змінних (для цього існують різні функції, все залежить від того для чого ці змінні слугуватимуть чи яке значення можуть приймати);
3. `void setup () { ... }` – формування вікна сцени, тут прописуються розміри вікна сцени, підвантажуються дані у змінні, включається за потреби аудіо чи відео разом із завантаженням сцени, а також включається камера;
4. `void draw () { ... }` – створення системою об'єктів, що будуть у сцені та їх характеристик та поведінки і система їх постійно створює, оскільки оновлюється постійно екран, тому сюди заноситься все, що має бути в сцені постійно у кожному кадрі або характеристики, які постійно мають формувати параметри дій над об'єктами;
5. далі можуть бути інші функції та їх опис, що або задані у `void draw ()` та їх немає у синтаксисі мови Java, наприклад, створення літери з коробки `void drawBox () { ... }` (спочатку така функція прописується у `void draw ()`, наприклад `drawBox ()`, а потім нижче вже поза межами цього блоку йде вже опис цієї функції, щоб система розуміла що потрібно зробити, коли вона дійде до зчитування саме цього рядка) або вони є і вони певним чином визначають поведінку об'єктів і потрібно прописати що саме, наприклад, поведінку об'єктів при натисканні ЛКМ (до таких функцій-обробників дій від клавіатури чи мишки варто вносити ті дії та параметри, що мають викликатися в певний момент часу, наприклад, при натисканні на кнопку програватиметься музика).

Алгоритм прописування коду:

1. Підключення бібліотеки, що дозволяє працювати з вебкамерою (має бути обов'язково камера, інакше бібліотека доповненої реальності непрацюватиме – студент

розробляє доповнену реальність, що буде розпізнаватися камерою ноутбуку або підключеною до нього вебкамерою) за допомогою функції **import**

```
import processing.video.*;
```

2. Наступний рядок – підключення стокової бібліотеки, що дозволить створювати елементи доповненої реальності (**nyar4psg**) за допомогою функції **import**

```
import jp.nyatla.nyar4psg.*;
```

3. У випадку додавання можливості вертіння літер навколо своєї осі наступним рядком необхідно задати кут повороту для кубів за допомогою функції **float**, задаючи змінну **angBox**, що відповідатиме за кут повороту літери, з початковим кутом у **0 градусів**

```
float angBox=0;
```

4. Далі треба задати підпрограму, що відповідатиме за створення робочого простору, у якому розміщуватимуться літери. Це виконується за допомогою функції **void** із заданням методу **setup()**. У цій підпрограмі початково задається тільки розмір простору за допомогою функції **size (x,y,P3D)** – задається розмір простору по довжині (**x**) та висоті (**y**) у пікселях та відмічається для системи, що користувач працює у 3Д-просторі, щоб літери були об'ємними, а також тому, що у подальшому буде скануватися 3Д-простір при розпізнаванні маркерів

```
void setup()  
{  
size (640,480,P3D);  
}
```

У прикладі написано розмір 640 x 480 пікселів – можна робити більше, але камера потім буде розпізнавати простір саме такого розміру, тому щоб непереробляти варто початково поставити такий розмір.

5. Наступні рядки коду – **підпрограма оновлення графічного вікна** – це поле для опису процесу розпізнавання маркеру, тому повністю буде заповнене вже перед етапом тестування AR-додатку. Єдине, що варто задати на даному етапі – це виклик підпрограми побудови коробок (літер) – **drawbox ()**.

```
void draw()  
{  
drawbox ();  
}
```

6. Далі задається опис підпрограми **drawbox ()**. Кожна літера для зручності задається окремо – так для кожної літери можна задати свої розміри коробок, їх колір, контур та кут повороту. Це виконується за допомогою функцій **pushMatrix()** та **popMatrix()**, що символізують початок та кінець побудови літери. Тут треба задати всі параметри, що пов'язані з оформленням кубів, із яких формуватиметься літера, а також їх розміщення:

- колір коробки можна задати за допомогою функції **fill(R,G,B)**, де R,G,B – значення кольору, що змінюється в межах від 0 до 255. *Якщо задати fill(0,0,0) – об'єкт буде чорного кольору, fill(255,255,255) – білого кольору, fill(255,0,0) –*

червоний, *fill(0,255,0)* – зелений, *fill(0,0,255)* – синій. Змінюючи значення 3 складових можна сформувати різні відтінки та їх поєднання;

- якщо треба задати контур для кубика, то це можна зробити за допомогою функції **strokeWeight()** – задає товщину лінії, де у дужках прописується товщина лінії – та **stroke(R,G,B)**, де R,G,B – значення кольору, що змінюється в межах від 0 до 255, для задання кольору контуру;
- кубик задається функцією **box()**, де у дужках прописується кубик. Якщо у дужках писати тільки одне число, наприклад *box(10)*, то буде формуватися саме куб із розміром однієї сторони у 10 пікселів. Якщо прописати у дужках довжину ширину та висоту, наприклад, *box(20,10,30)*, то буде формуватися паралелепіпед із довжиною 20, шириною 10 та висотою 30 пікселів;
- для задання повороту всієї літери можна використати функцію **rotateX(angle)** / **rotateY(angle)** / **rotateZ(angle)** для повороту літери відносно Осі X, Y чи Z відповідно. Якщо у дужках прописати «-», то поворот виконуватиметься у зворотньому напрямку. Так як кут трошки раніше був заданий через змінну **angBox**, то у дужках треба її і написати, наприклад, **rotateY(-angBox)** – рух навколо осі Y проти годинникової стрілки. Також треба задати суміщення куту повороту **angBox += значення**, щоб літера почала обертатися. Значення, що прописується після знаку «=», задає швидкість обертання і воно має бути дуже маленьким, наприклад, у прикладі прописано **angBox += 0.01**, чим меншим буде число від даного значення, тим швидше обертатиметься літера;
- зміщення кубика один відносно одного задається за допомогою функції **translate(x, y)**. Перший кубик кожної літери початково треба змістити на певну відстань по x то у в межах розміру робочого простору, розмір якого заданий був вище – так щоб було комфортно з ним далі працювати, оскільки початково куб будується у лівому верхньому куті. При цьому варто за потреби проводити тестування роботи коду, натискаючи на кнопку плеї, щоб підналаштовувати положення коробки під себе, змінюючи значення у цій функції. Наступна коробка вже буде будуватися відносно попередньої (тепер вона початок відліку – її верхній лівий кут на передній грані) за допомогою функції **translate (x, y, z)**. Потім задаємо новий кубик з тим же розміром, що і у попереднього. Наприклад, такі рядки коду

```
translate(180, 100);  
box(40);  
translate (50,0,0);  
box(40);
```

означатимуть, що спочатку буде побудовано кубик розміром 40 на відстані 180 по осі X та 100 по осі Y від лівого верхнього кута екрану, після цього на відстані 50 по осі X від нього буде побудований наступний кубик.

На основі вище сказаного наведено приклад прописання коду для функції **drawBox()** для однієї літери:

```
void drawbox()  
{  
  pushMatrix();  
  fill(255,0,255);  
  strokeWeight(1);  
  stroke(0,0,0);  
  translate(180, 100);  
  rotateY(-angBox);  
  angBox += 0.01;  
  box(20);  
  translate(-30,0,0);
```

```

box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(30,0,0);
box(20);
translate(30,0,0);
box(20);
translate(30,0,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
popMatrix();
}

```

Результат виконання наведений на рисунку 5

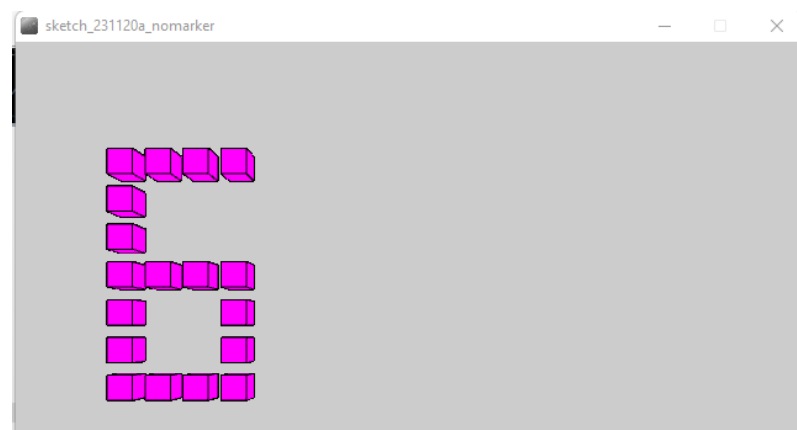


Рисунок 5 – Результат виконання прикладу коду

Рядок **popMatrix()** дає змогу зрозуміти програмі при зчитуванні кожного рядка при програванні коду, що побудова літери закінчилася і початок відліку знову повертається у лівий верхній кут. Для наступних 2 літер треба знову задати усі необхідні параметри у блоці **pushMatrix()** / **popMatrix()** (2 окремі блоки) відповідно до задумки.

Приклад побудованого ПІБ наведений на рисунку 6.

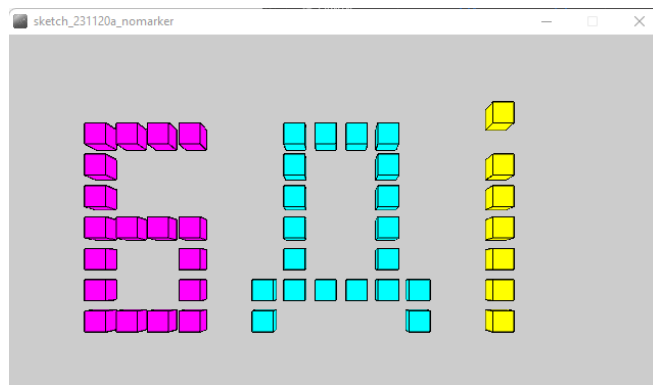


Рисунок 6 – Результат побудови 3 літер

3. Розроблення коду для створення доповненої реальності

Після корегування літер можна перейти під налаштування камери та програмування створеного зображення як маркеру доповненої реальності (*треба зберегти проєкт, якщо до цього ще цього не було зроблено і перекинути файл зображення-маркеру у папку з проєктом*). За це відповідає ряд функцій, наведених у таблиці 4.1.

Таблиця 4.1 – Функціонал

Команда	Опис команди
Оголошення змінних, підключення бібліотек	
Capture cam;	оголошення об'єкту cam класу Capture, отримання зображення з камери;
MultiMarker nya;	зادання об'єкту, що буде маркером
Підпрограма налаштування поля перегляду	
cam=new Capture(this,640,480);	створення об'єкту Capture розміром 640 на 480 (як робочий простір)
nya=new MultiMarker (this, width, height, "camera para.dat", NyAR4PsgConfig.CONFIG PSG);	створення об'єкту MultiMarker
nya.addARMarker("patt.hiro",80);	вибір маркера (має називатися у лапках так, як називається файл зображення-маркеру)
cam.start();	запуск камери
Підпрограма оновлення графічного вікна	
void draw() {	підпрограма оновлення графічного вікна
if (cam.available() != true) { return; }	перевірка доступності камери, вихід якщо недоступна
cam.read();	прочитання одного кадру з камери
nya.detect(cam);	передача зображення з камери в об'єкт пуа
background(0);	очищення графічного екрану
nya.drawBackground(cam);	виведення в графічне вікно поточного кадру з камери
if (nya.isExistMarker(0) != true) { return; }	перевірка наявності маркера на кадрі – якщо є, то програма переходить до побудови літер (drawbox()), якщо ні, то програма сканує наступний кадр
Початок розпізнавання маркеру	
nya.beginTransform(0);	початок трансформування фігури
drawbox();	виклик підпрограми побудови кубів
nya.endTransform (); }	завершення трансформування фігури

Приклад загального кінцевого коду наведений нижче
import processing.video.*;

```

import jp.nyatla.nyar4psg.*;
//float angBox=0;
Capture cam;
MultiMarker nya;
void setup()
{
size (640,480,P3D);
cam=new Capture(this,640,480);
nya=new MultiMarker (this, width, height, "camera_para.dat", NyAR4PsgConfig.CONFIG_PSG);
nya.addARMarker("patt.hiro",230);
cam.start();
}
void draw()
{
if (cam.available() != true)
{
return;
}
cam.read();
nya.detect(cam);
background(0);
nya.drawBackground(cam);
if (nya.isExistMarker(0) != true)
{
return;
}
nya.beginTransform(0);
drawbox();
nya.endTransform ();
}
void drawbox()
{
pushMatrix();
fill(255,0,255);
strokeWeight(1);
stroke(0,0,0);
translate(180, 100);
//rotateY(-angBox);
//angBox += 0.01;
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(30,0,0);
box(20);
translate(30,0,0);
box(20);

```

```

translate(30,0,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
popMatrix();
pushMatrix();
fill(0,255,255);
strokeWeight(1);
stroke(0,0,0);
translate(250, 280);
//rotateY(angBox);
//angBox += 0.01;
box(20);
translate(0,-30,0);
box(20);
translate(30,0,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(30,0,0);
box(20);
translate(30,0,0);
box(20);
translate(30,0,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(0,30,0);
box(20);
translate(-30,0,0);
box(20);
translate(-30,0,0);
box(20);
translate(90,0,0);
box(20);
translate(0,30,0);
box(20);
popMatrix();
pushMatrix();
fill(255,255,0);

```

```
strokeWeight(1);
stroke(0,0,0);
translate(480, 280);
//rotateY(angBox);
//angBox -= 0.01;
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-30,0);
box(20);
translate(0,-50,0);
box(20);
popMatrix();
}
```

Якщо усе працює (програма не видає помилок і запускає тестовий режим із запуском веб-камери), то в певних межах маркеру можна помітити відповідні літери та їх анімацію.

Хід роботи

1. Описати цільову аудиторію та призначення розроблюваного мультимедійного продукту
2. Створити зображення-маркер. Описати його основні елементи
3. Розробити код програми для зчитування маркеру доповненої реальності. Протестувати отриманий код, зробити скрін результату зчитування.
4. Створити протокол, де описати етапи роботи, елементи коду та занести скрін результату виконання. Зробити загальні висновки по роботі.

Контрольні запитання:

1. Що таке маркер доповненої реальності?
2. Які існують види маркерів та в чому між ними різниця?
3. Що необхідно для розпізнавання маркерів?
4. Які є принципи розпізнавання маркерів?
5. Як задати параметри шрифту у Processing?
6. Як задати поворот об'єктам у Processing?

СТВОРЕННЯ ОБ'ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ З РІЗНИМИ ХАРАКТЕРИСТИКАМИ (РУХ ПО ТРАЄКТОРІЇ, ВИКОРИСТАННЯ ТЕКСТУР, РІЗНОМАНІТТЯ ФОРМ) У СЕРЕДОВИЩІ PROCESSING

Мета роботи: вивчення особливостей розроблення об'єктів різної форми для наповнення елемента доповненої реальності, можливостей накладання текстур на примітиви, задання їх руху по траєкторії, прив'язка об'єктів до маркера з використанням мови програмування Java у середовищі Processing.

Особливості виконання

1. Створення фігур різної форми

У середовищі Processing через спеціально розроблені функції можна задавати тільки паралелепіпеди або квадрати (**box()**) та сфери або еліпсоїди (**sphere()**). Проте можливість створення інших простих фігур також є, код яких описаний у таблиці 1.

Таблиця 1 – Створення фігур

Команда	Опис команди
Конус	
<code>void draw() { drawCone(10,100);}</code>	підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення конусу із радіусом 10 і висотою 100
<code>void drawCone(float r,float h){</code>	Оголошення функції створення конусу, де <i>r</i> – це змінна для задання радіусу, <i>h</i> – це змінна для задання висоти
<code>int sides=40;</code>	Задання кількості граней конуса, тобто скільки трикутних полігонів формуватимуть поверхню конуса. Чим більше значення, тим менш помітними є трикутники, тим більш гладка поверхня
<code>beginShape(TRIANGLE_FAN);</code>	Початок побудови фігури
<code>vertex(0,-h/2,0);</code>	Задання верхівки конуса, від якої формуватимуться всі прямокутники
<code>for(int i=0;i<=sides;i++){ float a=TWO_PI*i/sides; float x=cos(a)*r; float z=sin(a)*r; vertex(x,h/2,z); }</code>	Формування циклу для виконання побудови трикутників по колу для формування поверхні конуса (цикл виконуватиметься на 1 раз більше, ніж кількість трикутників, для замкнення кола – остання грань буде повторювати першу). Float <i>a</i> – задання кута, на який треба перейти при побудові наступної точки основи, Float <i>x</i> і Float <i>z</i> – задання координат кожної точки на основі конуса. Vertex () – побудова цих точок та поєднання їх з вершиною
<code>endShape();</code>	Кінець створення фігури
Циліндр	
<code>void draw() { drawCylinder(30,100);}</code>	підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення циліндру із радіусом 30 і висотою 100
<code>void drawCylinder(float r, float h) {</code>	Оголошення функції створення циліндру, де <i>r</i> – це змінна для задання радіусу, <i>h</i> – це змінна для задання висоти
<code>int sides = 40;</code>	Задання кількості граней циліндра (прямокутників)
<code>beginShape(QUAD_STRIP);</code>	
<code>for(int i=0;i<=sides;i++){ float a=TWO_PI*i/sides; float x=cos(a)*r; float z=sin(a)*r; vertex(x,-h/2,z); vertex(x,h/2,z);} endShape();</code>	Цикл працює так само, як і для конуса, але так як у циліндра дві основи, то й функції vertex () треба дві для їх формування

Кінець таблиці 1

Команда	Опис команди
Тор	
void draw() { drawTorus(90,25,40,20); }	підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення тору із радіусом 90 зовнішнього кола, радіусом внутрішнього 25, кількістю сегментів зовнішнього кола 40 та кількістю сегментів внутрішнього 20
void drawTorus(float R, float r, int seg1, int seg2){	Оголошення функції створення циліндру, де R – великий радіус, r – малий радіус, seg1 – кількість частин великого кола, seg2 – кількість частин малого кола
for(int i=0;i<seg1;i++){ float a1 = TWO_PI*i/seg1; float a2 = TWO_PI*(i+1)/seg1; beginShape(QUAD_STRIP); for(int j=0;j<=seg2;j++){ float b = TWO_PI*j/seg2; float x1 = (R+r*cos(b))*cos(a1); float y1 = (R+r*cos(b))*sin(a1); float z1 = r*sin(b); float x2 = (R+r*cos(b))*cos(a2); float y2 = (R+r*cos(b))*sin(a2); float z2 = r*sin(b); vertex(x1,y1,z1); vertex(x2,y2,z2);}} endShape();	У цілому цикл схожий на попередні – будуються точки зовнішнього кола, внутрішнього кола та поєднуються за допомогою граней, формуючи прямокутники у передбаченій кількості для кожного кола

Змінюючи параметри у рядках кодів для фігур можна їх модифікувати в інші, наприклад, якщо у конусі задати меншу кількість трикутників-полігонів, що формують його поверхню, наприклад, поставити 5, то можна створити піраміду, а з циліндра можна створити різноманітні 3D-фігури з різними багатокутниками в основі.

Також можна створювати складні фігури, прописуючи рядки коду для їх створення. Програмний код деяких наведених у таблиці 2.

Таблиця 2 – Створення складних фігур

Команда	Опис команди
Серце	
void draw() { drawHeart3D();}	підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення серця
void drawHeart3D(){	Оголошення функції створення серця
beginShape(); for (float t=0; t<TWO_PI; t+=0.08) { float xx = 16*pow(sin(t),3); float yy = -(13*cos(t)- 5*cos(2*t)-2*cos(3*t)- cos(4*t)); vertex(xx*2, yy*2, depth/2); } endShape(CLOSE);	Створення передньої сторони: 1. Все розпочинається функцією побудови форми beginShape() 2. Далі формується цикл побудови точок передньої сторони, що триватиме, поки програма не пройде ціле коло (2π) – починається все з 0 градусів, крок 0.08 радіану, тобто програма виконуватиме побудову невеликими відрізками для більшої плавності та деталізації лінії 3. функція float xx знаходить значення координати X для кожної точки циклу і в подальшому дозволить формувати півкола серця 4. функція float yy знаходить значення координати Y для кожної точки циклу і в подальшому дозволить формувати: 13*cos(t) – висота серця 5*cos(2*t) – також для формування верхніх півкіл 2*cos(3*t) – краща згладженість серця - cos(4*t) – гострий низ серця - перед всім цим ставиться мінус, щоб серце не було перевернутим 5. функція vertex () – формуватиме вершини серця протягом проходження циклу та саму фігуру загалом 6. endShape(CLOSE) – закінчення побудови фігури, CLOSE – поєднує останню точку з першою

Продовження таблиці 2

Команда	Опис команди
Серце	
<pre>beginShape(); for (float t=TWO_PI; t>=0; t-=0.08) { float xx = 16*pow(sin(t),3); float yy = -(13*cos(t)-5*cos(2*t)- 2*cos(3*t)-cos(4*t)); vertex(xx*2, yy*2, -depth/2);} endShape(CLOSE);</pre>	Те саме, але для побудови зворотної поверхні
<pre>beginShape(QUAD_STRIP); for (float t=0; t<=TWO_PI+0.08; t+=0.08) { float xx = 16*pow(sin(t),3); float yy = -(13*cos(t)-5*cos(2*t)- 2*cos(3*t)-cos(4*t)); vertex(xx*2, yy*2, depth/2); vertex(xx*2, yy*2, -depth/2); } endShape();</pre>	Побудова бічної частини, щоб серце було об'ємним, у цілому те саме, що й для основ, є дві функції vertex (), оскільки треба поєднати 2 поверхні, QUAD_STRIP) – означає, що потрібно побудувати смугу з полігонів-прямокутників, у циклі до загального кута у 2 π додається 0,08, щоб не було щілин між останнім та першим полігонами.
Зірка	
<pre>void draw() { drawStar(); }</pre>	Підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення об'ємної зірки
void drawStar() {	Оголошення функції створення серця
float R = 100;	Задання радіуса зовнішнього кола, по якому створюватимуться точки-вершини зірки (від центра до кінців зірки)
float r = 45;	Задання радіуса внутрішнього кола, по якому створюватимуться точки-вершини зірки (від центра до западин зірки)
float depth = 30;	Глибина розміщення вершин
<pre>beginShape(); for (int i=0; i<10; i++) { float angle = i*TWO_PI/10 - HALF_PI; float radius; if (i%2==0) radius = R; else radius = r; float x = radius*cos(angle); float y = radius*sin(angle); vertex(x, y, depth/2); } endShape(CLOSE);</pre>	Створення передньої сторони зірки 1. Все розпочинається функцією побудови форми beginShape() 2. Далі формується цикл побудови точок передньої сторони, що триватиме, поки програма не побудує всі необхідні вершини (у 5-кінечної зірки 10 точок вершин – 5 по довгому радіусу R та 5 по короткому r) 3. Функція float angle встановлює кут, на який відхилятиметься уявний промінь при побудові наступної вершини, наприклад перша вершина буде проставлена на промені, що відходить від центра побудови під кутом 36 градусів тощо. HALF_PI необхідно прописати, щоб верхній промінь зірки відкладався нагору, щоб вона не була перекручена 4. Функція float radius задає змінну для визначення по якому радіусу відкладатиметься вершина – зовнішнього кола (кінці зірки) R чи внутрішнього (западини) r 5. Створення умови if (i%2==0) необхідно, щоб точки будувалися послідовно – спочатку відкладалася вершина по довгому радіусу R, а потім по короткому r (щоб вони чергувалися і будувалася зірка) 6. Функції float x та float формують координати X та Y точок-вершин 7. Функції vertex () – формуватиме зірку 8. endShape(CLOSE) – закінчення побудови фігури, CLOSE – поєднує останню точку з першою
<pre>beginShape(); for (int i=0; i<10; i++) { float angle=i*TWO_PI/10-HALF_PI; float radius; if(i%2==0) radius=R; else radius=r; float x=radius*cos(angle); float y=radius*sin(angle); vertex(x,y,-depth/2); } endShape(CLOSE);</pre>	Створення зворотної сторони зірки – в цілому все так само, єдине що при побудові вже самої зірки у функції vertex() глибина розміщення вершин задається з від'ємним значенням, щоб всі вони розташовувалися в межах однієї зворотної площини

Кінець таблиці 2

Команда	Опис команди
Зірка	
<pre>beginShape(QUAD_STRIP); for (int i=0; i<=10; i++) { float angle=(i%10)*TWO_PI/10- HALF_PI; float radius; if(i%2==0) radius=R; else radius=r; float x=radius*cos(angle); float y=radius*sin(angle); vertex(x,y, depth/2); vertex(x,y,-depth/2); } endShape();</pre>	Створення бічної сторони зірки – в цілому все так само, як і для інших сторін зірки, є дві функції vertex (), оскільки треба поєднати 2 поверхні, QUAD_STRIP – означає, що потрібно побудувати смугу з полігонів-прямокутників, при визначенні кута переходу до загального кута у 2π домножається $i\%10$, щоб не було щілин між останнім та першим полігонами.
Багатокутник	
<pre>void draw() { drawPolygon(3); }</pre>	Підпрограма оновлення графічного вікна / створення програмою наповнення сцени – виклик підпрограми створення об'ємного рівностороннього багатокутника, у даному випадку трикутника
<pre>void drawPolygon(int sides) {</pre>	Оголошення функції створення серця
<pre>float R = 100;</pre>	Визначення радіусу кола, у яке вписується основа об'ємного багатокутника, у даному випадку розмір 100
<pre>float depth = 30;</pre>	Визначення глибини розміщення вершин багатокутника
<pre>float[] x = new float[sides]; float[] y = new float[sides];</pre>	Функції для формування масивів координат X та Y точок, які будуть вершинами багатокутника
<pre>for (int i = 0; i < sides; i++) { float angle = i * TWO_PI / sides - HALF_PI; x[i] = R * cos(angle); y[i] = R * sin(angle); }</pre>	Визначення координат точок, що будуть вершинами багатокутника – отримані координати будуть заповнювати комірки відповідного масиву, сформованого вище
Багатокутник	
<pre>beginShape(); for (int i = 0; i < sides; i++) { vertex(x[i], y[i], depth/2); } endShape(CLOSE); /</pre>	Формування передньої сторони фігури
<pre>beginShape(); for (int i = 0; i < sides; i++) { vertex(x[i], y[i], -depth/2); } endShape(CLOSE);</pre>	Формування зворотньої сторони фігури
<pre>beginShape(QUAD_STRIP); for (int i = 0; i <= sides; i++) { int j = i % sides; vertex(x[j], y[j], depth/2); vertex(x[j], y[j], -depth/2); } endShape();</pre>	Формування бічної сторони фігури

Проаналізувавши отримані коди, можна зробити висновок, що у загальному більшість складних фігур формуються шляхом опису передньої, зворотньої сторони та бічної, що поєднує дві основи в єдине ціле. Параметри функцій можна змінювати, отримавши при цьому цікавіший результат.

2. Створення руху об'єкту по траєкторії та обертання фігур

Об'єкти у Processing можуть рухатися по певній траєкторії, це може бути лінія, еліпс, спіраль або інші складніші форми кривої траєкторії. Також об'єкти можуть обертатися навколо себе по одній з осей X, Y чи Z. Також вони можуть рухатися по траєкторії, наприклад, по колу навколо іншого об'єкту. Програмний код різних рухів наведений у таблиці 3.

Таблиця 3 – Рух об'єктів

Команда	Опис команди
Обертання навколо себе по одній з осей	
... Float ang=0 ... rotateY(ang); ang += 0.03;	Спочатку задається змінна та початковий кут повороту об'єкта, у даному випадку ang=0 градусів. Далі у void draw () перед створенням відповідного об'єкта задається функція rotate () і вказується вісь, відносно якої обертатиметься об'єкт, а в дужках прописується задана змінна, у даному випадку об'єкт буде обертатися навколо осі Y на кут ang. Для того, щоб цей рух почався, треба задати зміщення (ang += 0.03;)
Збільшення / зменшення (пульсація об'єкта)	
... float pulse = 1 + 0.2 * sin(frameCount * 0.1); ... scale(pulse);	Спочатку задається змінна для формування пульсації об'єкту через формулу $1 + 0.2 * \sin(\text{frameCount} * 0.1)$, що дозволяє робити плавну зміну розміру об'єкту від 100% до 80%. 0.2 – це амплітуда пульсації, чим більша, тим більш помітною є зміна розмірів. 0.1 – визначає швидкість пульсації, чим менше це число, тим меншою є швидкість пульсації. Далі у void draw () перед створенням відповідного об'єкта задається функція scale(pulse), щоб запустити цю зміну розмірів при виконання програми.
Рух по прямій	
... float yPos = 20; float speed = 1.5; ... translate (0, yPos, 0); yPos += speed;	Функція зміщення об'єкта по певній осі, у даному випадку по осі Y, починаючи з певного значення, що можна задати через змінну до void setup () {}, наприклад, float yPos = значення (якщо буде з мінусом, то рух буде до низу). А вже при створенні об'єкта у void draw () задати це зміщення по прямій через функцію translate () та зміну змінної (у даному випадку по швидкості). Так само об'єкт може рухатися по осі X та по осі Z у додатньому та від'ємному напрямку, тоді треба відповідну координату задати через змінні, а всі інші будуть 0. Для того, щоб об'єкт саме рухався, необхідно задати йому зміщення, наприклад, на величину швидкості 1,5
Рух по еліпсу (орбіта, кільце тощо)	
... Float ang=0 ... x=round(r*cos(PI*ang/180)) ; y=round(r*sin(PI*ang/180)); translate(x,y);	Початково задається кут об'єкта з якого він починатиме рух, далі у необхідний момент задається визначення координат X та Y для визначення розміщення об'єкта на еліпсоїдній траєкторії, останнім кроком додається зміщення об'єкта по цим точкам, щоб він почав рухатися, через функцію translate ()
Рух по спіралі	
... Float angle1=0 float r1 = 70; float yPos = 20; ... translate(cos(angle1)*r1, yPos, sin(angle1)*r1);	Функція зміщення об'єкта по траєкторії у вигляді спіралі з певним радіусом та під певним кутом, що задаються додатково через функцію float(), у даному випадку радіус траєкторії 70, початковий кут 0. Якщо об'єкт рухається по осі Y, як у даному випадку, то координата Y задається через певну змінну із значенням, а X та Z визначається за відповідними формулами для формування кола – $\cos(\text{angle1}) * r1$ та $\sin(\text{angle1}) * r1$, якщо по осі X – X задається числом, Y – через $\cos(\text{angle1}) * r1$, Z – через $\sin(\text{angle1}) * r1$, і так само для Z – X через косинус Y, через синус, Z – координата. Якщо для координат, що описуються формулами поміняти місцями sin і cos, то спіраль буде зміщена по фазі на 90 градусів
Рух по синусоїді	
... float xPos = -400; float z1 = sin(xPos * 0.03) * 120; ... translate(xPos, -120, z1); xPos += 2;	Задання руху по траєкторії – якщо об'єкт рухається по осі X та здійснює синусоїдальні коливання по осі Z, то треба задати початкове значення для координати X через змінну, а також задати синусоїдальне коливання по осі Z через змінну, у даному випадку z1, координата Y залишається незмінною протягом руху об'єкту, щоб уникнути додаткових стрибків (у даному випадку - 120) або 0, якщо немає необхідності зміщувати траєкторію відносно центра побудови. Для руху об'єкту по створеній траєкторії необхідно задати суміщення по осі X (xPos += значення)

Кінець таблиці 3

Команда	Опис команди
Рух по синусоїді	
<pre>... float xPos = -400; float y2 = sin(xPos * 0.03) * 120; ... translate(xPos, y2, 0); xPos += 2;</pre>	Задання руху по траєкторії – якщо об’єкт рухається по осі Y (вертикально) та здійснює синусоїдальні коливання по осі X, то треба задати початкове значення для координати X через змінну, а також задати синусоїдальне коливання по осі Y через змінну, у даному випадку y2, координата Z залишається незмінною протягом руху об’єкту, тому у даному випадку їй можна присвоїти значення 0. Якщо потрібно задати ще й рух вправо при цьому, то необхідно задати суміщення по осі X (xPos += значення)

Якщо є необхідність, наприклад, руху об’єктів навколо інших об’єктів, то потрібно задавати їм однаковий кут чи координату, яка відповідає за рух, наприклад, якщо одна сфера рухається навколо іншої по спіралі, а разом вони рухаються вниз, то у них має бути однакова координата для осі Y. Якщо наприклад сфера рухається по еліпсу навколо іншої, то у них має бути один і той самий кут для задання кривої еліпсоподібної траєкторії. Наприклад код для задання руху однієї сфери навколо іншої (як можна побачити один і той самий кут angle)

```
float angle = 0;
void setup()
{
  size(800, 600, P3D);
}
void draw()
{
  background(220);
  lights(); // підсвітлює сцену
  translate(width/2, height/2); // розміщує початок відліку у центрі робочого простору
  angle += 0.03; // для обох сфер швидкість руху буде однаковою, якщо треба різною, то треба задати цей
показник в межах pushMatrix(); ... popMatrix(); для кожної сфери
  pushMatrix();
  rotateY(angle);
  sphere(40);
  popMatrix();
  float R = 150;
  float x = R * cos(angle);
  float z = R * sin(angle);
  pushMatrix();
  translate(x, 0, z);
  rotateY(angle);
  sphere(20);
  popMatrix(); }
```

Рух по спіралі однієї сфери навколо іншої з одночасним спуском обох до низу (тут однакова координата Y, щоб обидва об’єкти рухалися донизу одночасно)

```
float angle1 = 0;
float selfAngle = 0;
float yPos = -250;
float r1 = 120;
void setup() {
  size(800, 600, P3D);
}
void draw() {
  background(220);
  lights();
  translate(width/2, height/2);
  angle1 += 0.05; // рух по спіралі
```

```

selfAngle += 0.12; // обертання малої сфери
yPos += 0.5;      // спуск системи
// коли анімація доходить до кінця, то потім плавно починається зверху екрану знову
if (yPos > 300) {
  yPos = -300;
}
pushMatrix();
translate(0, yPos, 0);
sphere(35);
popMatrix();
pushMatrix();
translate(cos(angle1) * r1, yPos, sin(angle1) * r1);
rotateY(selfAngle);
sphere(18);
popMatrix();
}

```

3. Оформлення елементів

У Processing є різні можливості для оформлення як об'єктів, так і для організації та оформлення робочого простору.

Робочому простору можна задати розмір, тривимірність чи двовимірність за допомогою функції **size(ширина, висота, P3D)**. У дужках прописується розмір робочого простору та за потреби, якщо об'єкти мають рухатися в 3D просторі, то також це вказується, якщо ж 2D – то просто вказується розмір простору. Також всім об'єктам, що є у робочому просторі можна задати ступінь згладжування країв за допомогою функції **smooth(значення)**. У дужках прописується ступінь згладжування – від 2 до 16 (чим більше число, тим менша «пікселізація» країв об'єктів). Якщо, наприклад, непотрібне згладжування, то треба використати функцію **noSmooth()**. Якщо не прописувати значення, то програма автоматично вибере значення. Ці всі функції задаються у **void setup ()** для формування робочого простору.

Також робочому простору можна задати колір, як і об'єктам. Це робиться у **void draw()** за допомогою функції **background(значення)**. Ця функція працює так само, як **fill ()** для об'єктів (див. лабораторна №4). Можна задавати одне значення від 0 до 255, тоді таке значення буде присвоєне всім трьом каналам одночасно, або ж задавати кожному з каналів своє, щоб формувати кольорові відтінки. Також можна задати картинку як фон. Для цього треба спочатку задати змінну для збереження цієї картинки через **PImage** змінна до моменту оголошення побудови робочого простору, а вже у **void setup ()** після функції **задання розмірів робочого простору** прописати рядок

змінна = loadImage("назва картинки.jpg");

а потім у **void draw()** прописати

background(значення);

image(змінна,0,0,width,height); // width,height – розміри вікна, автоматично беруться із **size()**, якщо розміри зображення співпадають з розміром вікна, то просто пишеться **image(змінна,0,0);**

Зображення має зберігатися в тій же папці, що й проєкт. Назва змінної і картини – **АНГЛІЙСЬКОЮ**.

Також у Processing є функції налаштування освітлення сцени, щоб робити об'єкти більш деталізованими, розставляти тіні чи робити підсвічування.

Для створення стандартного освітлення сцени використовується функція **lights()**. Система автоматично проставляє та налаштовує освітлення.

Також можна задати створення, наприклад, розсіяного світла за допомогою функції **ambientLight()**. Воно світить з усіх боків одночасно і об'єкти не мають різких тіней. Так само, як і для фону і об'єктів, тут можна задавати один колір для всіх каналів одночасно від 0 до 255 та для кожного з трьох каналів окремо.

Також можна задати направлене світло, як сонце, за допомогою функції **directionalLight(R, G, B, x, y, z)**, де **R, G, B** – значення по кожному каналу від 0 до 255, **x, y, z** – напрямок світіння джерела.

Є ще точкове джерело світла, як лампочка. Його можна задати за допомогою функції **pointLight(R, G, B, x, y, z)**. Тут **x, y, z** – це саме розміщення самого джерела. Чим далі об'єкт від джерела, тим слабше він освітлюється.

Ще один варіант – це світіння прожектора, тобто світло створюється в межах конуса. Для цього використовується функція **spotLight(R,G,B, X, Y, Z, x, y, z, l, k)**. Тут **R, G, B** – значення по кожному каналу від 0 до 255, **X, Y, Z** – розміщення самого джерела **x, y, z** – напрямок світіння джерела, **l** – ширина конуса, в межах якого світить джерело (задається як π і його похідні – $\pi/2$, $\pi/4$ тощо), **k** – визначає, як розподіляється яскравість усередині конуса (чим ближче до 1, тим більш рівномірним є світло).

Всі варіанти освітлення вказуються у void draw() ОДРАЗУ після задання кольору простору. Це допоможе системі прораховувати освітлення з кожним оновленням сцени у кожному кадрі. Можна задавати декілька джерел і комбінувати їх між собою для створення цікавішої картинки для користувача. Зазвичай вказують **lights()**, а всі інші вже по задумці, якщо треба.

Можна також вимкнути всі джерела освітлення за допомогою функції **noLights()**. Його можна вказувати не одразу, а перед тим об'єктом, який не повинен освітлюватися. Все що буде йти **ПІСЛЯ** рядка **noLights()**; освітлюватися не буде, **поки не буде прописане нове джерело освітлення**.

Щодо об'єкту, то йому також можна задавати різноманітні параметри оформлення. Можна задавати колір об'єкту чи створювати для об'єктів текстуру, а також задавати для них контур. Для задання заливки фігури можна скористатися функцією **fill()**, а для задання контуру – **stroke()**, де в дужках вказується або одне значення для 3 каналів одночасно, або для кожного окремо, або конкретний колір, якщо такий є в системі (див. лабораторна робота №4). Також можна задавати прозорість об'єктам як четвертий параметр цих функцій (альфа-канал). У системі він змінюється також від 0 до 255. Тому, наприклад, рядок **fill(255,0,0,50)** означатиме, що об'єкт, що йде після цього рядка буде червоного кольору з прозорістю приблизно 20% ($50 \approx 20\%$). Ці функції прописуються у **void draw () ПЕРЕД** створенням об'єкту. Якщо контур непотрібен, то потрібно прописати рядок **noStroke()**; Якщо не потрібна заливка – **noFill()**;

Для лінії ще можна задати товщину лінії через **strokeWeight(значення)**.

Якщо потрібно, щоб ці характеристики відносилися до конкретного об'єкта, якщо їх декілька, то як і у лабораторній №3 потрібно скористатися **pushMatrix ();...popMatrix()**;, інакше вони застосовуються до всіх об'єктів, що написані після них (див. лабораторна №4).

Якщо потрібен градієнт, то як такої функції у Processing немає, але можливість для створення є, наприклад, можна створити декілька об'єктів з різним кольором та ступенем прозорості з однаковими координатами центра.

Текстура створюється наступним чином – спочатку до задання простору треба прописати змінну для задання текстури через функцію **PImage**, потім у **void setup ()** задати вкладення у цю змінну зображення, що буде текстурою для об'єкта (зображення має бути в тій же папці, що й проєкт), а потім задати параметри об'єкту із заданням текстури і вже у **void draw ()** створити цей об'єкт через **shape** (змінна), наприклад, треба створити Місяць, тоді рядки коду будуть наступні

```
import processing.video.*;
PShape moon; // створення змінної для задання об'єкту Місяць, використовується, якщо треба
               створювати об'єкт саме з текстурою, якщо треба створювати таким чином декілька
               об'єктів, то можна всі одразу прописувати в один рядок через кому (PShape moon,
               sun, earth...;)
```

```

PImage moontexture; // так само можна декілька змінних для задання текстур для різних об'єктів
                               прописувати в один рядок
void setup() {
size (640,480,P3D);
moontexture = loadImage("moon.jpg"); // підвантаження значення (зображення) у задану змінну для
                               текстури
moon = createShape(SPHERE, 5); // задання змінної для задання об'єкту як сфери з певним радіусом (у
                               даному випадку 5)
moon.setTexture(moontexture); // накладання текстури на об'єкт
moon.setStroke(false); // за потреби можна задати, щоб не промальовувалися каркасні лінії сфери
}
void draw() {
...
pushMatrix();
shape(moon); // створення об'єкту, заданого у void setup()
...
popMatrix();
}

```

Також можна задати як об'єкт буде взаємодіяти зі світлом. Для цього можна скористатися такими функціями:

1. **ambient(R,G,B,alfa)** – це колір об'єкта при розсіяному (загальному) освітленні. Дозволяє регулювати матовість об'єкту.

2. **specular(R,G,B,alfa)** – для створення кольору світлих ділянок, тобто відблисків. Створює металічні або глянцеві об'єкти.

3. **shininess(значення)** – дозволяє регулювати параметри цього блиску, а саме наскільки він буде концентрованим (м'яким розсіяним чи більш чітким). 1 – м'який розсіяний відблиск, 100 – концентрована яскрава цятка. Чим більше за 100, тим більш «відполірованим» буде об'єкт

4. **emissive(R,G,B,alfa)** – дозволяє об'єкту самостійно випромінювати світло певного відтінку.

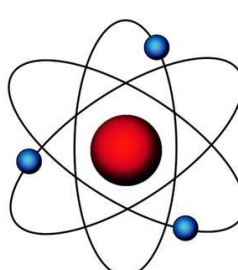
Хід роботи

1. Відповідно до варіанту по списку групи обрати із таблиці (див. табл. 4) для виконання відповідний варіант завдання. Визначитися із об'єктами та особливостями виконання описаних процесів.

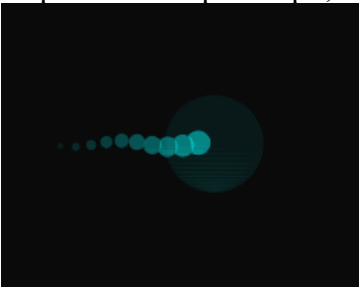
Таблиця 4 – Варіанти завдань

Варіант	Опис завдання
1	Створити модель Сонячної системи, де у центрі є Сонце, навколо нього обертаються 8 планет, а навколо Землі – додатково Місяць. Слід врахувати, що планети мають обертатися з різною швидкістю як навколо Сонця, так і навколо себе, а також різного розміру. На планети потрібно накласти текстуру.
2	Створити ефект ліхтарика – 2-3 об'єкти різної форми, кольору та розміру крутяться навколо іншого та освітлюють його. Траєкторія – на власний розсуд
3	Створити ефект лупи – одна напівпрозора сфера чи іншої форми об'єкт рухається по прямій і, доходячи до іншого об'єкту, збільшує його у розмірах

Продовження таблиці 4

Варіант	Опис завдання
4	Створити 2 об'єкти – серце та конус – обертаються навколо власної вісі та рухаються по колу з різним радіусом. Один об'єкт рухається у вертикальній площині, інший – в горизонтальній (на власний розсуд). Траєкторії їх руху не перетинаються.
5	Створити куб, на одній з вершин якого горизонтально розміщено паралелепіпед, що обертається відносно вертикальної вісі. На краях цього паралелепіпеда розташовано дві сфери різних розмірів, що обертаються навколо своїх осей. Зробити один об'єкт з контуром та заливкою, один з текстурою, один такий, що світиться, та один напівпрозорий.
6	Створити 3 об'єкти різних розмірів – зірка та два на власний розсуд. Зірка рухається вниз. Два інші об'єкти рухаються вниз навколо неї по спіралі з різним діаметром та обертаються відносно власної вісі.
7	Створити 2 об'єкти – циліндр та піраміду, один з об'єктів рухається вправо по синусоїді в горизонтальній площині, а інший – аналогічно у вертикальній площині. Обидва об'єкти «пульсують».
8	Створити 2 сфери. Одна зі сфер є більшою в діаметрі та рухається по спіралі вниз, інша – рухається по синусоїді вправо. Обидві сфери обертаються відносно власної вісі. На обидві сфери накласти текстуру
9	Створити 2 паралелепіпеди, довжина яких втричі більша за ширину та 2 сфери. Паралелепіпеди обертаються відносно осі x по довгому краю в різні сторони. Перша сфера рухається навколо першого паралелепіпеда вниз по спіралі, друга – навколо другого вгору по спіралі. Обидві сфери обертаються навколо власної вісі. Всім об'єктам задати різні характеристики матеріалу – матовість, блиск, світіння тощо
10	Створити 2 об'єкти – серце та 3Д-октаedr – різних розмірів, один з яких рухається вліво по синусоїді в горизонтальній площині, а інший – вгору по спіралі. Обидва сфери обертаються відносно власної вісі з різною швидкістю. Траєкторії їх руху не перетинаються.
11	Створити 2 серця різних розмірів, які рухаються на зустріч одна одній по спіралі. Сфери рухаються з різними швидкостями. Обидві сфери обертаються відносно власної вісі.
12	Створити 1 паралелепіпед, довжина якого втричі більша за ширину та 2 сфери різних діаметрів. Паралелепіпед обертається відносно осі x по довгому краю. На певній відстані від паралелепіпеда обертається більша зі сфер, а менша обертається навколо більшої. Більша сфера рухається по спіралі у горизонтальній площині. Обидві сфери обертаються відносно власної вісі.
13	Створити 4 сфери. Одна зі сфер є більшою в діаметрі та являє собою ядро атома. Три інші сфери являють собою електрони та обертаються навколо ядра за траєкторіями відповідно до рисунку. Орбіти мають бути видимими, сфери мають бути різного кольору
	
14	Створити 6 сфер однакових розмірів та зірку більшого розміру. Розташувати ці сфери по колу на однаковій відстані відносно зірки, забезпечити рух цих сфер по колу та відносно своєї вісі. Зірка має пульсувати та підсвічуватися

Кінець таблиці 4

Варіант	Опис завдання
15	Створити ефект зоряного неба – зробити темний фон, розкидати по ньому сфери або зірки різного розміру, що пульсують та підсвічуються, а також комету у вигляді сфери, що пролітає по траєкторії, наприклад, як на рисунку 
16	Створити ефект маятника – зірку, яка тримається на нитці і починає свій рух зверху справа і з певним часом її швидкість затухає та вона зупиняється.

2. Створити зображення-маркер (особливості розроблення – див. лабораторна робота №4). Описати його основні елементи

3. Розробити код програми для зчитування маркеру доповненої реальності (структура коду програми – див. лабораторна робота №4). Описати що кожен рядок означає. Протестувати отриманий код, зробити скрін результату зчитування.

4. Створити протокол, де описати етапи роботи, елементи коду та занести скрін результату виконання. Зробити загальні висновки по роботі.

Контрольні запитання:

1. Які фігури можна створити у програмному середовищі та які функції для цього потрібні?
2. Які варіанти освітлення сцени можна реалізувати у програмі?
3. Як створити рух по спіралі?
4. Як можна створити характеристики матеріалу для об'єктів?
5. Чим відрізняється задання текстури для фону та для об'єкту?

Лабораторна робота №6

**СТВОРЕННЯ ІНТЕРАКТИВНОЇ ЛИСТІВКИ
У СЕРЕДОВИЩІ PROCESSING**

Мета роботи: створення інтерактивної AR-листівки з функціями керування клавішами миші, використанням 3D об'єктів та аудіофайлів у середовищі Processing

Особливості виконання

1. Імпорт 3D-моделей

У середовищі Processing є можливість імпорту із зовнішніх джерел не тільки 2D-зображень, але й 3D-моделі у форматі .obj. Для цього треба прописати наступний код

```
PShape змінна;  
void setup()  
{  
  size(900,700,P3D);  
  змінна = loadShape("назва.obj"); // має бути в тій же папці, що і проєкт  
}  
void draw()  
{  
  pushMatrix();  
  ...  
  shape(змінна);  
  popMatrix();  
}
```

Тобто в цілому це схоже на створення об'єкту із текстурою (див. лабораторна робота №5). Задається змінна, у змінну вкладається 3D-модель, завантажена зовні і потім цей об'єкт викликається, щоб бути у сцені. 3D-моделі можна задавати ті ж функції, що й іншим об'єктам, створеним у самому Processing. Так само, як і для інших об'єктів всі зміни (колір, розмір, текстура, матеріал, рух, положення, поворот) задаються перед рядком чи рядками зі створенням об'єкту (у даному випадку 3D-моделі) у void draw() (див. лабораторна робота №4-5).

2. Імпорт аудіоінформації

Для програвання аудіоінформації спочатку треба під'єднати спеціальну бібліотеку через наступний рядок

```
import ddf.minim.*;
```

Вона дозволить налаштовувати роботу з аудіоданими. Далі перед початком всього процесу після задання бібліотек потрібно задати дві змінні наступним чином

```
Minim minim;  
AudioPlayer music;
```

Перша змінна буде керувати можливостями бібліотеки – завантажувати музику, створювати аудіоплеєри, керувати роботою з мікрофоном, виконувати аналіз частот,

записувати звук тощо. Друга змінна буде програвати конкретний файл, який в неї на подальших етапах написання коду вкладеться.

Далі у **void setup()** після задання розмірів 3D-простору пишеться наступне

```
minim = new Minim(this);  
music = minim.loadFile("music.mp3");
```

Перший рядок дозволяє створити об'єкт-контролер можливостями бібліотеки для змінної, заданої раніше, класу **Minim** конкретно для цієї розроблюваної сцени.

Другий рядок означає, що за допомогою створеного у попередньому рядку контролера **minim** розробник створює аудіоплеєр для змінної **music**, що була задана раніше, який повинен буде програвати аудіофайл, що заданий у дужках. Цей аудіофайл, як і зображення чи тривимірні об'єкти, має збергатися в тій же папці, що й файл проєкту.

Один контролер **minim** може керувати декількома аудіоплеєрами, тобто можна створити декілька різних змінних для аудіоплеєрів, у кожний з них задати свій аудіотрек і потім певним чином відповідно до задуму запрограмувати та програвати.

Далі вже можна створювати рядки коду для програвання музики та інших її налаштувань. Їх можна вказати у **void setup ()**, тоді музика почне програватися одразу ж зі створенням сцени, або ж у межах опису функцій, що відповідатимуть за програвання при натисканні на кнопку миші чи клавішу на клавіатурі. Якщо їх вказати у **void draw ()**, то це спричинить постійне запускання музики (оскільки ця функція постійно оновлює кадр і формує те, що в ній прописано) і вона не програватиметься фактично. До музики можна застосовувати наступні дії:

- **music.play();** – аудіоплеєр **music** програватиме аудіотрек, що був у нього підвантажений у **void setup ()** один раз;
- **music.loop();** – аудіоплеєр **music** програватиме аудіотрек, що був у нього підвантажений у **void setup ()** безкінечну кількість раз;
- **music.rewind();** – якщо цей рядок написати **ПІСЛЯ music.play();**, то музика ще раз програється з початку, якщо написати **ДО**, то музика почнеться з початку;
- **music.skip();** – якщо цей рядок написати **ПІСЛЯ music.play();**, то музика промотається вперед чи назад на задану кількість мілісекунд (додатне значення – перемотка треку вперед, від'ємне – назад), наприклад якщо треба промотати на 5 секунд вперед, то треба в дужках написати 5000;
- **music.pause();** – якщо цей рядок написати **ПІСЛЯ music.play();**, то музика зупиниться;
- **music.setGain();** – якщо цей рядок написати **ДО music.play();**, то можна встановити гучність музики в дБ (від'ємне значення – музика буде звучати тихіше, 0 – нормальна гучність; додатне значення – гучніше), якщо написати **ПІСЛЯ**, то музика почне грати з нормальною гучністю, а потім змінить гучність до відповідного значення;
- **music.setBalance(-1);** – якщо цей рядок написати **ДО music.play();**, то можна зробити панорамування звуку (від'ємне значення – буде більш підсилений лівий канал, 0 – звук збалансований; додатне значення – буде більш підсилений правий канал), як що ж написати **ПІСЛЯ**, то так само, як і для гучності – спочатку буде збалансований звук, а потім вже в залежності від значення підсилить відповідний канал;
- **music.cue();** – якщо цей рядок написати **ДО music.play();**, то музика почне програватися з конкретної мілісекунди, записаної у дужках, якщо написати **ПІСЛЯ**, то вже після початку програвання музика перейде до відповідної мілісекунди;

- **music.setRate();** – якщо цей рядок написати **ДО music.play();**, то можна контролювати швидкість програвання музики (від 0 до 1 – уповільнення швидкості програвання, більше 1 – пришвидшення), якщо написати **ПІСЛЯ**, то вже після початку програвання музика вона почне грати швидше чи повільніше в залежності від значення.

*Як можна побачити всі дії починаються зі змінної **music**, але якщо змінна називається по-іншому при створенні плеєра, то тоді на початку цих дій пишеться відповідна змінна. Наприклад, якщо задається **AudioPlayer bell;**, то й у діях пишеться **bell.дія ()**, наприклад, **bell.play(); bell.loop(); bell.rewind();** тощо.*

3. Імпортування відеоінформації

У Processing можна під'єднувати і відеоінформацію у форматах .mp4 (зазвичай цей формат використовується), .mov та .avi. Під'єднується вона схожим чином до аудіоінформації. Спочатку підключається бібліотека для роботи з відео **import processing.video.*;**, потім прописується змінна **Movie movie;**, що дозволить працювати з відео, і далі вже у **void setup()** імпортується відеофайл та прив'язується до створеної змінної **movie = new Movie(this,"video.mp4");** і далі цю змінну з відео можна використовувати або як окремий об'єкт, або ж як фон. У **void draw ()** також треба задати місце, в межах якого програватиметься відео **image(movie,0,0,width,height);**, де **movie**, – змінна з підвантаженим відео, 0 – положення відео по осі X. 0 – положення відео по осі Y, **width** – ширина екрана для програвання (якщо написано таким чином, то дорівнює ширині AR-сцени), **height** – висота екрана для програвання (якщо написано таким чином, то дорівнює висоті AR-сцени).

Над відео можна виконувати наступні дії:

- **movie.play();** – програвання відео один раз від початку до кінця (якщо написати після паузи, то з того моменту, як відео було зупинено);
- **movie.loop();** – зациклення відео;
- **movie.pause();** – зупинення програвання відео на поточному кадрі;
- **movie.stop();** – повне зупинення програвання відео (після цього відео, якщо є **movie.play();** після цього рядка, то почне програватися з початку);
- **movie.jump();** – почати програвання відео з певної секунди;
- **movie.speed();** – пришвидшення / сповільнення відео (від 0 до 1 – швидкість відео зменшується, більше 1 – збільшується);
- **movie.volume();** – зміна гучності звуку у відео (змінюється від 0 до 1).

*Так само, як і для аудіоінформації, налаштування відео можна вказати у **void setup ()**, тоді відео почне програватися одразу ж зі створенням сцени, або ж у межах опису функції, що відповідатимуть за програвання при натисканні на кнопку миші чи клавішу на клавіатурі. Також в усіх цих рядках спочатку йде змінна **movie**, якщо вона була по-іншому названа при заданні змінної для відео (**Movie movie;**), то й при заданні дій треба буде вказувати саме ту назву – наприклад, у цьому рядку написано **Movie film;** то тоді буде **film.play(); film.loop(); film.pause();** тощо.*

Також відео може бути текстурою для об'єктів чи фону. Задається так само, як і для зображень-текстур (див. лабораторна робота №5), тільки замість зображення імпортується відео.

4. Керування діями за допомогою миші чи клавіатури

У Processing можна створювати керування діями об'єктів при натисканні на кнопку миші або ж при натисканні на клавішу клавіатури. Для цього можна використати наступний синтаксис:

1. Регулювання дій над об'єктами сцени при натисканні якої-небудь клавіші на клавіатурі

Далі наведений приклад для налаштування роботи аудіо. Для всіх інших об'єктів сенс подібний.

```
void draw()
{
  ...
}
void keyPressed()
{
  if(key == ' ') //дія виконуватиметься при натисканні на пробіл
  {
    //виконання дій з музикою, наприклад, нижче задається програвання музики з самого початку один раз
    music.rewind();
    music.play();
  }
}
```

Якщо необхідно, щоб система реагувала на натискання клавіші з літерою та щоб при цьому невраховувався регістр, то тоді треба написати у цьому рядку наступне

```
if(key == 'a' || key == 'A') //якщо важливий регістр, то тоді if(key == 'a') або if(key == 'A')
```

Якщо контролером дій є стрілки на клавіатурі, то треба прописати наступне у даному рядку

```
if(keyCode == LEFT) // стрілка вліво (RIGHT – стрілка вправо, UP – стрілка вгору, DOWN – стрілка вниз)
```

Якщо потрібно, щоб музика не програвалася заново при натисканні, наприклад, на пробіл, а навпаки, наприклад, дозволяла періодично зупиняти музику і програвати її з того ж місця, де відбулася пауза, то для цього треба прописати наступний синтаксис:

```
void keyPressed()
{
  if(key == ' ')
  {
    if(music.isPlaying()) // метод, що дозволяє програмі самостійно визначати, що музика грає
    {
      music.pause(); //якщо музика грає, то при натисканні на пробіл вона зупиниться
    }
    else
    {
      music.play(); //якщо ні, то почне грати із місця, де зупинилася
    }
  }
}
```

2. Регулювання дій з об'єктами при затисканні клавіші на клавіатурі

Якщо, наприклад, необхідно регулювати якийсь параметр при затисканні клавіші, наприклад, гучність, то потрібно прописати наступне

```
float volume = -30; //задати змінну для формування значення гучності
void setup()
{
  ...
}
```

```

music.setGain(volume); // задати початкове значення гучності через задану змінну та її значення
...
}
void draw()
{
...
if(keyPressed)
{
if(key == ' ') // коли затиснутий пробіл (keyPressed – метод, що відповідає за поведінку об'єкта, коли
якась клавіша затиснута)
{
volume += 0.1; // поки не буде відпущений пробіл, гучність зростатиме на 0,1 дБ
volume = constrain(volume,-30,5); // дозволяє зробити обмеження для змінної, щоб її значення було в
межах від -30 до 5 дБ, при виході за ці межі нічого
невідбуватиметься, можна обмеження не ставити, але чим
більша буде гучність, тим більше буде спотворення
}
}
}
}

```

4. Регулювання дій з об'єктами при використанні миші

У даному випадку використовується метод `mousePressed()`. Синтаксис наступний

```

void mousePressed() // коли натиснуто на ЛКМ
{
music.play(); // починає програватися музика
}

```

або

```

void mousePressed() // коли натиснуто на ЛКМ
{
if(music.isPlaying()) // система аналізує чи грає музику
{
music.pause(); // тоді музика зупиниться
}
else
{
music.play(); // якщо ж музика не грає, то вона почне грати з того місця, де зупинилася
}
}

```

або

```

float volume = -20; // задати змінну для формування значення початкової гучності
...
}
void mouseWheel (processing.event.MouseEvent event) // створення події при прокручуванні колесика на
миші
{
volume -= event.getCount(); // встановлюється напрямок руху колесика
music.setGain(volume); // якщо вгору, то звук ставатиме гучнішим, якщо вниз – тихішим
...
}

```

Можна так само задавати дії через затиснення ЛКМ. Працює так само як і затиснення клавіші клавіатури, єдине що замість змінної **keyPressed** використовується змінна **mousePressed**

```
float volume = -20; //задати змінну для формування значення початкової гучності
...
}

void draw()
{
  if(mousePressed)
  {
    volume += 0.2;
    music.setGain(volume);
  }
}
```

Всі вищеописані методи можна застосовувати і для контролювання анімації об'єктів, наприклад обертання об'єкта навколо осі, поки затиснута ЛКМ чи рух об'єкта в залежності від того, яка стрілочка натиснута тощо, і для програвання відеоінформації.

5. Використання тексту

У Processing можна використовувати шрифти. У загальному у **void draw ()** треба використати змінну **text ()**, наприклад

```
text("Привіт!",100,100); // "Привіт!" – текст, який побачить користувач, 100 –
положення тексту по осі X, 100 – положення тексту по осі Y.
```

У текста можна налаштовувати різні параметри (вказуються до **text ()** у **void draw ()**):

- **textSize(40);** – кегль шрифту (у даному випадку 40);
- **fill(255,255,0, 120);** – колір шрифту (у даному випадку жовтий (255,255,0) і напівпрозорий (120));
- **textAlign(CENTER);** – вирівнювання тексту в межах сцени по ширині (можна і по висоті, наприклад, **textAlign(CENTER,CENTER);**);
- **stroke();** – контур тексту.

Також можна задавати власні гарнітури шрифту. Для цього спочатку треба задати змінну для гарнітури, наприклад, так

PFont font;

Потім у **void setup()** покласти у цю змінну значення, а саме гарнітуру та присвоїти гарнітуру тексту значення цієї змінної. Наприклад,

```
font = createFont("Arial",32);
textFont(font);
```

Якщо шрифт встановлений у систему комп'ютера, то можна прописати тільки його правильну назву, наприклад Arial, Helios тощо. Також можна скопіювати шрифт у папку з проєктом, тоді треба писати "Назва шрифту.розширення", щоб проєкт не був залежний від того чи є шрифт на комп'ютері, де він програватиметься. 32 – розмір шрифту у пікселях

Також можна створювати багаторядковий текст. Є два варіанти:

- **text("Рядок 1\nРядок 2\nРядок 3",100,100);** //самостійно розбивати текст на рядки
- **text("Довгий текст...",50,50,300,200);** //Система сама переноситиме рядки

Хід роботи

1. Описати задум, концепцію, колірно-шрифтове оформлення, цільову аудиторію та методи розповсюдження інтерактивної листівки з елементами доповненої реальності.

2. Описати елементи, якими буде наповнена сцена продукту. Листівка має містити:

- кілька 3Д-моделей (мінімум 3), що рухаються та мають текстуру;
- аудіо чи/та відеоінформацію;
- увімкнення/вимкнення програвання анімації чи характеристик аудіо/відео шляхом натиснення мишкою по екрану.

3. Створити зображення-маркер (особливості розроблення – див. лабораторна робота №4). Описати його основні елементи

4. Розробити код програми для розробленої листівки (структура коду програми – див. лабораторна робота №4). Описати що кожен рядок означає. Протестувати отриманий код, зробити скрін результату зчитування.

5. Створити протокол, де описати етапи роботи, елементи коду та занести скрін результату виконання. Зробити загальні висновки по роботі.

Контрольні запитання

1. Як можна завантажити 3D-об'єкт у систему та що з ним робити у середовищі програми?

2. Як можна завантажити аудіоінформацію у систему та які характеристики можна налаштовувати у середовищі програми?

3. Як можна завантажити відеоінформацію у систему та які характеристики можна налаштовувати у середовищі програми?

4. Як можна налаштувати текст у середовищі програми?

5. Як зробити регулювання параметрів об'єкта за допомогою колесика комп'ютерної миші?

СТВОРЕННЯ ЕЛЕМЕНТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ UNITY

Мета роботи: створення AR-елементів за допомогою програмного забезпечення Unity.

Теоретичні відомості

1. Загальні відомості про ресурс

Unity – багатоплатформовий інструмент для розробки відеоігор і застосунків, і рушій, на якому вони працюють. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

Програма-редактор Unity працює на Windows, macOS і Linux, а сам рушій може запускатися на 25 платформах, а саме iOS, Android, Tizen, Windows, Mac, Linux, PlayStation Xbox One, Apple ARKit та інші. Ігрова логіка пишеться за допомогою мови C#, раніше також була можливість використовувати Boo та JavaScript, але розробники відмовились від їх підтримки.

2. Функціональні можливості

Редактор Unity має інтерфейс, що складається з різних вікон, які можна розташувати на свій розсуд. Завдяки цьому можна проводити налагодження розроблюваного об'єкту прямо в редакторі. **Головні вікна** – це оглядач ресурсів проєкту **Project**, інспектор поточного об'єкта **Inspector**, вікно попереднього перегляду **Game**, оглядач сцени **Scene** та оглядач ієрархії ресурсів **Hierarchy**.

Проєкт в Unity поділяється на **сцени (рівні)** – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як об'єкти-моделі (ландшафт, персонажі, предмети довкілля тощо), так і порожні ігрові об'єкти – ті, що не мають моделі, проте задають поведінку інших об'єктів (тригери подій, точки збереження прогресу тощо). Їх дозволяється розташовувати, обертати, масштабувати, застосовувати до них скрипти. В них є назва, може бути тег (мітка) і шар, на якому він повинен відображатися. Різні моделі можуть об'єднуватися в набори (ассети) для швидкого доступу до них.

Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

Рендеринг зображення відбувається через віртуальну камеру огляду. В робочій області редактора ігрова сцена може розміщуватися як завгодно, а при рендерингу – так, як її видно з камери. В сцені може бути декілька камер, які рухаються за персонажем чи за вказаною траєкторією. Вигляд з камери подається в двовимірно чи тривимірно. Фон сцени, видимий через камеру, типово зображає небо, але може презентувати й інше довкілля.

Unity підтримує файли 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance.

3. Vuforia

Vuforia – це платформа доповненої реальності й інструментарій розробника програмного забезпечення доповненої реальності (Software Development Kit – SDK) для мобільних пристроїв, розроблені компанією Qualcomm. Vuforia використовує технології комп'ютерного зору, а також відстеження плоских зображень і простих об'ємних реальних об'єктів (наприклад, кубічних) в реальному часі. З версії 2.5 Vuforia розпізнає текст, а з 2.6 – має можливість розпізнавати циліндричні маркери.

Можливість візуалізації зображень дозволяє розробникам розташовувати і орієнтувати віртуальні об'єкти, такі, як 3D-моделі і медіаконтент, в поєднанні з реальною сценою при відображенні її на дисплеях мобільних пристроїв. Віртуальний об'єкт позиціонується на реальному образі так, щоб точка зору спостерігача співвідносилася з ними однаковим чином для досягнення відчуття, що віртуальний об'єкт є частиною реального світу.

Vuforia підтримує різні типи маркерів, в тому числі безмаркерні технології Image Target, тривимірні багатоцільові мішені Multi-Target та інші. Додаткові можливості дозволяють уникнути ефекту маскування об'єктів з використанням так званих «Віртуальних кнопок» («Virtual Buttons»), забезпечити селекцію об'єктів і можливість програмно створювати і реконфігурувати їх у рамках коду, що самостійно модифікується.

Vuforia надає інтерфейси програмування додатків на мовах C ++, Java, Objective-C, і .Net через інтеграцію з Unity. Додатки доповненої реальності, створені на платформі Vuforia, сумісні з широким спектром пристроїв, включаючи iPhone, iPad, смартфони та планшети на Android з версії 2.2 і процесором, починаючи з архітектур ARMv6 або 7 з можливістю проведення обчислень з плаваючою комою.

Особливості виконання

1. Зареєструватись на офіційному сайті Unity <https://unity.com/download> та завантажити програмне забезпечення **Unity Hub**.

2. Інсталювати завантажений файл за відповідними вказівками (краще, якщо буде можливість, встановлювати більш стару версію, оскільки вони є більш стабільними, наприклад, версію Unity 2022.3.62f1 або іншу 21 чи 22 року, обов'язкові компоненти – **Microsoft Visual Studio**, **Android Build Support** – для додатків для андроїду (Open JDK і Android SDK & NDK tools), **ios Build Support** – для айфонів (можна, але мало ймовірно, що вдасться зробити і завантажити додаток для ios системи через проблеми з авторськими правами). Якщо встановлення йде версії по замовчуванню, то варто потім встановити цю версію у **Unity Hub – Installs / Instal Editor / Обрати необхідну версію Unity (іншу видалити) / Обрати необхідні компоненти / Дочекатися встановлення**. Якщо якийсь компонент не встановлено – **Unity Hub / Installs та обрати біля встановленої версії шестерню / Add modules / Обрати необхідні / Install**.

3. Зареєструватися на офіційному сайті Vuforia <https://developer.vuforia.com/> та перейти у свій акаунт, де спочатку обрати меню **Plans & Licences** (раніше називалося **Licenses Manager**), що дозволить створити спеціальний **License Key** для роботи з **Vuforia**. У цьому меню необхідно створити нову ліцензію шляхом натискання на кнопку **Get Basic**. У вкладці, що з'явилася необхідно дати назву (англійською), поставити галочку та натиснути кнопку **Confirm**, погоджуючись з умовами. Після цього треба зайти у створений ключ та скопіювати ліцензію.

4. Наступним кроком необхідно створити маркер-зображення. Для цього спочатку необхідно скачати зображення, яке буде маркером. Далі треба перейти у меню **Target Manager** у акаунті та натиснути кнопку **Add Database**. У меню, що з'явилося, необхідно дати назву маркеру (англійською), обрати тип **Device** та натиснути кнопку **Create**. Потім обрати створений маркер та у новій вкладці натиснути кнопку **Add Target**, у

меню, що з'явилося, обрати тип **Single image**, обрати своє зображення у пункті **File**, встановити ширину **Width**, наприклад, **5** і натиснути кнопку **Add**. Після цього у характеристиці **Rating** зображення з'явиться певна кількість зірок – від 1 до 5, що відповідає рівню розпізнавання зображення. Найкращим варіантом є таке зображення, у якого рейтинг становить 5 зірок – воно легко та швидко розпізнається майже за будь-яких умов, найгіршим – 1. Краще підбирати зображення із рейтингом не менше 3 зірок. Наступним кроком необхідно клацнути на кнопку **Download Database (All)**, де обрати **Unity Editor** та натиснути **Download** та дочекатися завантаження, після чого відкрити завантажений файл та імпортувати створену базу до середовища **Unity**.

5. Відкрити встановлений **Unity Hub**, перейти у вкладку **Projects**, натиснути **New Project**, обрати шаблон **3D (built-in render pipeline)**, версію Юніті зверху, назвати проект та вказати шлях для збереження додатку (всі назви папок та файлів – **англійською!!!**).

6. Першочергово необхідно переформатувати обраний проект під систему **Android**. Для цього треба зайти в меню **File/Build Settings**. У новому вікні обрати **Android** та натиснути кнопку **Switch Platform**, а також виконати ряд дій з проектом для уникнення помилок, що були пояснено у відео.

7. У вкладці **Window** середовища **Unity** обрати пункт **Package Manager**, де знайти та встановити пакети **AR Foundation** та **Google AR Core XR Plugin**. Також треба оновити **XR-plugin Management** і **Text Mesh Pro**.

8. Завантажити (якщо пішла все-таки більш сучасна версія юніті) з <https://developer.vuforia.com/> з меню **Downloads** файл пункту для юніті, відкрити його та натиснути кнопку **Import** у меню, що з'явиться у середовищі **Unity** та дочекатися завершення операції. А для більш старої завантажити варіант з гугл-диску. Базу з маркером також завантажити в проект.

9. Наступним кроком необхідно створити AR-камеру. Для цього можна клацнути правою кнопкою миші у полі **Hierarchy** та вибрати вкладку **Vuforia Engine**, де обрати варіант **AR Camera** (або наступний шлях **GameObject / Vuforia Engine / AR Camera**).

*На кожному етапі у будь-який момент можна тестувати те, що виходить – треба натиснути на кнопку **Play** над робочим простором, де відображаються Ваші об'єкти і дочекатися, поки Юніті почне тестовий режим. Для виходу треба на неї обов'язково ще раз натиснути, бо потім якщо перейти до створення APK, то все злетить.*

10. Далі необхідно налаштувати роботу камери. Для цього необхідно обрати новостворену камеру у полі **Hierarchy**, потім перевести курсор у поле **Inspector** та обрати пункт **Open Vuforia Engine Configuration** та у новому меню, що відкрилося у полі вводу ліцензії ввести скопійований у 3 пункті ключ.

11. Далі необхідно зробити прив'язку до маркеру. Спочатку необхідно створити зображення-мітку в робочому полі **Scene**. Для цього слід клацнути правою кнопкою миші у полі **Hierarchy** та вибрати вкладку **Vuforia Engine**, де обрати варіант **Image Target**. Далі у полі **Inspector** створеного елементу **Image Target** обрати пункт **Image Target Behaviour**, потім у пункті **Type** обрати варіант **From Database**, після чого у пункті **Database** обрати назву імпортованої бази з маркером. Після виконання даних дій відповідний маркер-зображення з'явиться у полі **Scene** у початковій точці. Маркер як і будь-який інший об'єкт можна переміщувати (Обрати об'єкт у робочому просторі або у полі **Hierarchy / Inspector / Position** або відповідний курсор), перевертати (Обрати об'єкт у робочому просторі або у полі **Hierarchy / Inspector / Rotate** або відповідний курсор), масштабувати (Обрати об'єкт у робочому просторі або у полі **Hierarchy / Inspector / Scale** або відповідний курсор). Варто поставити його вертикально.

12. Наступним кроком необхідно наповнити маркер. Для даної роботи необхідно буде створити логічну завершену сцену з анімованою 3D-моделлю, 3D-декораціями та музичним наповненням. Для завантаження анімованої 3D-моделі можна завантажити звичайну фігуру в форматі .obj (бажано з витягнутими вбік руками, що нагадують літеру Т, розархівувати завантажений архів (буде містити 2 папки – **Source**, де безпосередньо зберігається файл 3D-моделі, та **Texture**, де зберігаються текстури, що формують зовнішній вигляд моделі) наприклад, з <https://sketchfab.com/>, а потім на сайті <https://www.mixamo.com> обрати необхідну анімацію і, натиснувши на кнопку **Upload Character** та виконавши усі вказівки, підставити свою 3D-модель та завантажити її. 3D-декорації можна також підвантажити та накласти на них текстуру або ж створити їх самостійно за допомогою можливостей Unity шляхом **Hierarchy/ПКМ/Create/3D-object** і потім обрати необхідний об'єкт та налаштувати його за допомогою поля **Inspector**.

13. Для легшого пошуку елементів та структурування поля **Project** можна створювати папки. Дане поле поділене на дві частини – менша частина показує усі папки та елементи проекту, права – склад папок. Усі основні елементи проекту містяться у папці **Assets**. Для створення папки або будь-якого іншого елементу треба клацнути правою кнопкою миші у правій частині поля **Project**, вибрати пункт **Create** та обрати **Folder** або будь-який інший необхідний елемент. Далі треба назвати дану папку для легкого пошуку елементів, зайти в неї та перетягнути в це поле завантажену анімовану модель та текстуру до неї. Далі необхідно витягнути цю фігуру на робочу область поля **Scene**, обрати її у полі **Hierarchy**, відредагувати її розмір та положення у полі **Inspector**, а також підв'язати її під AR-маркер шляхом перетягування елемента з назвою моделі на назву маркеру в меню **Hierarchy**. Далі у полі **Project** треба створити новий матеріал (ПКМ/Create/Material), обрати його, перетягнути обрану текстуру у поле **Inspector** даного матеріалу у пункт **Albedo**, а потім обрати необхідний елемент, на який необхідно накласти текстуру у полі **Hierarchy**, та перетягнути створений матеріал у поле **Materials/Elements** або просто на цей елемент. Таким чином обрана текстура накладеться на модель.

14. Далі необхідно налаштувати анімацію. Для цього необхідно обрати 3D-модель у поля **Project**, клацнути на ній ПКМ та обрати **Create/Animator Controller**. Далі обрати створений елемент та завантажити його. У новому полі клацнути ПКМ та створити новий стан **Empty**. Новоствореному стану дати назву, обрати його, в пункт **Motion** поля **Inspector** перетягнути анімацію моделі з поля **Project**, зациклити анімацію (завантажити створений стан та поставити галку у пункті **Loop**). Далі необхідно обрати 3D-модель у полі **Hierarchy**, у полі **Inspector** натиснути кнопку **Add Component** та обрати **Animator**, перетягнути створений **Animator Controller** у поле **Controller** створеного компоненту. Таким чином 3D-модель буде рухатися.

15. Для додавання музичного супроводу необхідно у папці **Assets** створити нову папку та перетягнути до неї будь-яку музику, що підходить під сюжет. Потім у полі **Hierarchy** створити новий порожній об'єкт (**Create Empty**), назвати його **Music**, обрати його та у полі **Inspector** натиснути кнопку **Add Component**, де додати компонент **Audio Source**, у поле **Audio Clip** якого перетягнути завантажену музику з поля **Project**. Музику також можна зациклити шляхом натискання галки у полі **Loop** компоненту **Audio Source**. Створений елемент **Music** у полі **Hierarchy** треба перетягнути у маркер у тому ж полі, аби музика підв'язалася під маркер.

16. Після створення усіх необхідних компонентів необхідно створити додаток. Для цього треба перейти **File/Build Settings** та натиснути кнопку **Build**. При виборі папки для збереження елемента необхідно слідкувати, аби в шляху знаходження файлу були лише латинські літери.

17. Для тестування додатку необхідно завантажити створений APK на телефон, встановити його, завантажити та навести на маркер на екрані або ж надрукований маркер. Перевірити також можна й у ПЗ – треба натиснути кнопочку **Play** у верхньому полі меню Юніті.

Хід роботи

1. Встановити Unity та всі необхідні компоненти.
2. Створити 3D-модель фону.
3. Створити анімовану 3D-модель, що підходить під сюжет.
4. Створити музичний супровід.
5. Елемент доповненої реальності повинен мати якусь логіку та ідею.
6. Створити звіт з виконаної роботи з супровідними скріншотами.

Контрольні запитання:

1. Unity. Операційні системи, під які можна розробляти додатки та мови програмування, які можна застосовувати.
2. Функціональні можливості Unity.
3. Vuforia та її особливості.
4. Як задати анімацію фігурі в Юніті?
5. Як створити музичний супровід додатку?

СТВОРЕННЯ ЕФЕКТІВ ЗА ДОПОМОГОЮ UNITY ТА ДОДАВАННЯ ЇХ ДО ЕЛЕМЕНТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ

Мета роботи: створення ефектів за допомогою програмного забезпечення Unity та впровадження їх у середовище елемента доповненої реальності.

Теоретичні відомості

Одним із застосовуваних ефектів Unity є **Particle System**, на базі якого будується різноманітна анімація в додатках – сніг, вогонь, феєрверк та інші. Компонент Particle System імітує рідкі субстанції на зразок різних рідин, хмар і чогось пов'язаного з вогнем шляхом генерації й анімації в сцені великої кількості невеликих 2D-елементів. Даний компонент має безліч налаштувань, що поділені на модулі. Біля заголовку кожного модуля розташовані чекбокси, які можна відзначати для включення або відключення тих чи інших налаштувань.

Крім модулів, в інспекторі даного компоненту є ще кілька інших елементів управління. Кнопка **Open Editor** відображає опції в окремому вікні редактора, яке також дозволяє редагувати одночасно кілька систем. А чекбокс **Resimulate** відповідає за те, будуть чи не будуть застосовуватися зміни властивостей до щойно створених системою часток у цей же момент. Кнопка **Wireframe** відображає контури меш-об'єктів, щоб показати частки, що знаходяться за ними. Існують такі модулі:

- **Particle System Module** – містить глобальні налаштування, які впливають на всю систему в цілому;
- **Emission Module** – властивості цього модуля впливають на частоту й синхронізацію генерації частинок;
- **Shape Module** – використовується для зміни форми Particle System, який генерується, і місця, звідки генеруються самі частинки;
- **Velocity Over Lifetime Module** – для зміни швидкості частинок протягом їх життєвого циклу;
- **Limit Velocity Over Lifetime Module** – управляє тим, як буде зменшуватися швидкість часток протягом їхнього життєвого циклу;
- **Force Over Lifetime** – можна призначити прискорення частинок за рахунок впливу зовнішніх сил;
- **Color Over Lifetime** – визначає як частки з плином часу змінюють колір і прозорість;
- **Color By Speed Module** – для зміни кольору частинок відповідно до їх швидкості за секунду;
- **Size Over Lifetime** – для зміни розміру частинок відповідно до кривої, яку можна налаштувати в цьому модулі;
- **Size by Speed** – для зміни розміру частинок відповідно їх швидкості в одиницю відстані в секунду;
- **Rotation Over Lifetime Module** – для організації обертання частинок в потрібному порядку під час їх руху;
- **Rotation by Speed** – для зміни обертання частинок відповідно їх швидкості в одиницю відстані в секунду;

- **External Forces Module** – модифікує ефект wind zones на частинках, що генеруються Particle System;
- **Sub Emitter** – дозволяє налаштовувати sub-emitters;
- **Texture Sheet Animation Module** – дозволяє використовувати свою текстуру як сітку невеликих зображень, які можна перемикаєти від кадру до кадру у вигляді анімації;
- **Rendering** – визначає як трансформується текстура частинок або меш.

У наступній таблиці 1 представлені параметри **Particle System**, та характеристики, які вони регулюють.

Таблиця 1 – Параметри Particle System

Назва параметру	Опис характеристики, що регулюється
Particle System Module	
Duration	Тривалість часу, протягом якого буде працювати система
Looped	Якщо підключена, то система почне свою роботу в місці завершення свого duration циклу і почне все заново.
Prewarm	Якщо підключена, то система буде ініціалізована так, ніби вона вже пройшла через весь цикл своєї роботи (працює виключно тоді, коли Looping також включено)
Start Delay	Затримка в секундах до старту системи генерації частинок після її включення
Start Lifetime	Початкова тривалість життя частинок
Start Speed	Початкова швидкість кожної частинки у відповідному їй напрямку
Start Size	Початковий розмір кожної частинки
Start Rotation	Початковий кут повороту кожної частинки
Start Color	Початковий колір кожної частинки
Gravity Multiplier	Зміна значення гравітації. Значення 0 призведе до відключення гравітації
Inherit Velocity	Характеризує наскільки самі частинки рухаються швидше чи повільніше за систему в цілому
Simulation Space	Визначає систему відліку для частинок
Play on Awake	Визначає автоматичність процесу роботи системи
Max Particles	Максимальна кількість частинок, що одночасно знаходяться у системі
Emission Module	
Rate	Кількість частинок, що генеруються за одиницю часу або пройденого шляху
Bursts	Можливість додаткових частинок генеруватися в потрібний момент часу
Shape Module	
Shape	Форма Particle System
Random Direction	При підключенні створює довільний початковий напрямок частинок
Radius	Радіус Particle System (тільки для сфери, півсфери і конуса)
Angle	Кут конуса Particle System (тільки для конуса)
Box X, Y, Z	Ширина, висота і глибина кубічної форми (тільки для кубів)
Mesh	Меш, що задає форму генератора (тільки для мешів)
Emit from Shell	Визначає, з якої сторони Particle System будуть генеруватися частинки – ззовні чи зсередини (тільки для сфер і півсфер)
Emit from	Визначає частину конуса, з якої буде відбуватися генерація частинок: (тільки для конусів)
Velocity Over Lifetime Module	
X, Y, Z	Швидкість по осях X, Y і Z.
Space	Визначає систему відліку для осей X,Y,Z
Limit Velocity Over Lifetime Module	
Speed	Ліміт швидкості
Space	Визначає відношення обмежень швидкості до локального або світового простору
Dampen	Характеризує значення зниження швидкості частинок в момент досягнення ними максимальної швидкості
Force Over Lifetime	
X, Y, Z	Сила, що застосовується до частинок по осям X, Y та Z
Space	Визначає простір застосування сили
Rotation by Speed	
Angular Velocity	Швидкість обертання в градусах в секунду
Speed Range	Нижня та верхня межі діапазону швидкості, на яку накладається крива розміру

Кінець таблиці 1

Назва параметру	Опис характеристики, що регулюється
Collision	
World/Planes popup	Визначення режиму World чи Planes
Planes	Розгортається список перетворень, який визначає площину зіткнень (тільки для режиму Planes)
Visualization	Визначення відображення Гизмо collision plane у видовому вікні сцени (тільки в режимі Planes)
Scale Plane	Розмір площини, що використовується для візуалізації (тільки в режимі Planes)
Dampen	Частина швидкості, що втрачається частинкою після зіткнення
Bounce	Частина швидкості частинки, переданої поверхні, від якої вони відбиваються
Lifetime Loss	Частина від загального часу життя частинки, яка втрачається при зіткненні.
Min Kill Speed	Мінімальна швидкість, нижче якої відбуватиметься видалення частинки після зіткнення
Particle Radius	Приблизний розмір часток, що використовується для уникнення відсікання з площинами зіткнення (тільки в режимі Planes).
Collides With	Частинки будуть стикатися тільки з об'єктами, розташованими на виділених шарах (тільки в режимі World).
Collision Quality	Визначає непроникність зіткнення (тільки в режимі World).
Send Collision Messages	Якщо підключено, то зіткнення частинки можуть бути зафіксовані через функцію скрипта OnParticleCollision.
Texture Sheet Animation Module	
Tiles	Кількість секцій, на які текстура буде розбита по осям X і Y
Animation	Режим анімації
Random Row	Навмання вибирається ряд зображень для відтворення анімації
Row	Вибирається конкретний ряд з сітки зображень для відтворення анімації
Frame Over Time	Крива, яка визначає те, як з плином часу змінюються кадри анімації
Cycles	Кількість повторень анімації протягом періоду життя частинки
Rendering	
Render Mode	Режими рендерингу
Camera Scale	Величина розтягування, що застосована пропорційно до руху камери (тільки в режимі Stretched Billboard).
Speed Scale	Величина розтягування, що застосована пропорційно до швидкості частинки (тільки в режимі Stretched Billboard).
Rendering	
Length Scale	Величина розтягування, що застосована пропорційно до довжини частинки (тільки в режимі Stretched Billboard).
Mesh	Кількість мешів (тільки для режиму Mesh).
Material	Матеріал, що формує текстуру частинки.
Sort Mode	Порядок розташування системи
Sorting Fudge	Зсув порядку сортування частинки
Cast Shadows	Наявність тіні від частинки
Receive Shadows	Визначає, чи повинні тіні відкидатися на частинки
Max Particle Size	Найбільший розмір частинки по відношенню до розміру вікна проекції.

Особливості виконання

Далі будуть приклади виконання деяких ефектів, вони можуть стати основою для створення подібних.

1. Ефект снігу, що падає

Даний ефект створюється наступним чином:

1. У полі **Hierarchy** створюється елемент **Particle System** шляхом ПКМ/Effect/Particle System.

2. У меню **Inspector** цього елемента у полі **Shape/Shape** обирається варіант **Box**.

3. У полі **Transform** задається розмір та положення площі, яку займатиме елемент (розмір можна проставити 10,10, 1 (поле Z не чіпати)).

4. Далі необхідно провести попереднє налаштування самого елемента. Для цього проставити наступні параметри:

- **Prewarm** – поставити галку;
- **Start Lifetime** – 10;
- **Start Speed** – 0 (аби частинки не рухалися вгору);
- **Start Size** – встановити Random Between Two Constants і проставити невеликі значення (наприклад, від 0,01 до 0,03);
- **Velocity over Lifetime** – підключити та поставити значення X (від -2 до 2), Y (від -2 до 2), Z (від -1 до -2) – для того, щоб сніг йшов вниз та нерівномірно, а з певним відхиленням від прямого напрямку;
- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%.

5. Далі необхідно налаштувати зовнішній вигляд снігу. Для цього спочатку створити новий документ у Photoshop розміром 100x100 пікселів у форматі RGB 72dpi. Далі залити фон чорним, розділити навпіл по горизонталі та вертикалі за допомогою направляючих та створити новий шар. На цьому шарі створити 4 елементи (кожний у своїй четверті) за допомогою пензлика із розміром Opacity 35% та розміром 6 пікселів. Має вийти щось схоже на приклад, наведений на рисунку 3. Зберегти його у форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

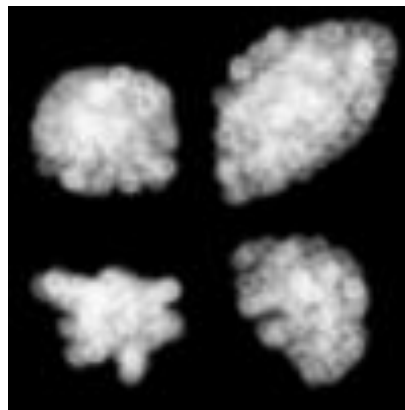


Рисунок 3 – Приклад текстури

6. Створити у цій же папці **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у попередніх пунктах;

7. Далі треба налаштувати зовнішній вигляд сніжинок, аби це не був просто квадрат з 4 елементами. Для цього треба налаштувати:

- **Texture Sheet Animation** – підключити і встановити значення Tiles X=2, Y=2;
- **Frame over Time** – від 0 до 3,9996 (таким чином обирається наближений зовнішній вигляд сніжинки відповідно до створеного у 5 пункті елементу);
- **Emission** – обрати даний пункт та поставити значення **Rate Over Time** ± 70 .

Загалом можна ще позмінювати попередні параметри. У результаті буде отриманий ефект снігу, який можна підв'язати до маркеру шляхом перетягування **Particle System** на маркер у полі **Hierarchy**

2. Ефект світлячків

Цей ефект чимось схожий на попередній. Етапи створення:

1. У полі **Hierarchy** створюється елемент **Particle System** шляхом ПКМ/Effect/Particle System.
2. У меню **Inspector** цього елемента у полі **Shape/Shape** обирається варіант **Sphere** та радіус 2.
3. У полі **Transform** задається розмір та положення площі, яку займатиме елемент.
4. Далі необхідно провести попереднє налаштування самого елемента. Для цього проставити наступні параметри:

- **Start Speed** – -5 (аби частинки рухалися до центру);
- **Start Size** – встановити **Random Between Two Constants** і проставити невеликі значення (наприклад, від 0,1 до 0,15);
- **Limit Velocity over Lifetime** – підключити та поставити значення **Speed** 0.5 **Dampen** 0.2 для того, щоб частинки рухалися більш плавно;
- **Size over lifetime** – обрати цей пункт, клацнути на полі **Size** та сформувати криву, схожу на представлену на рисунку 4. Це дозволить зробити зміну розміру більш плавною та реалістичною із певним затуханням на кінці;
- **Noise** – обрати даний пункт, встановити значення параметрів **Strength** 1, **Frequency** 2 та **Scroll Speed** 0,5. Це дозволить зробити рух світляків більш хаотичним.

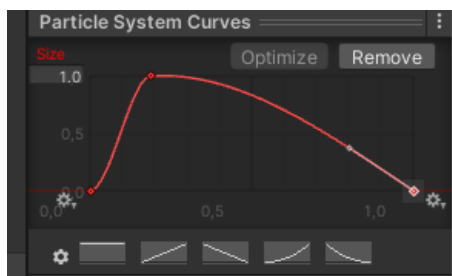


Рисунок 4 – Крива зміни розміру

5. Для створення шляху світляків необхідно обрати поле **Trail**, створити новий **Material** у новоствореній папці у папці **Assets**, у полі **Inspector** у полі **Shader** обрати **Particles/Standard Units** та у полі **Render Mode** режим **Additive** та перетягнути даний матеріал у поле **Renderer/Trail Materials** поля **Inspector** створеного **Particle System**.

6. Для налаштування даного сліду треба у полі **Trails** налаштувати наступні параметри – **LifeTime** 0.05, **Minimum Vertex Distant** поставити 0 та **Width Over Trail** обрати **Curve** та налаштувати криву, схожу на рисунок 4, та обрати значення початку відліку в 0,3 у верхньому лівому куті поля для редагування кривої.

7. Також треба налаштувати колір сліду. Для цього у полі **Start Color** треба обрати будь-який бажаний колір.

8. Наступним кроком необхідно створити самого світлячка. Для цього спочатку треба створити новий документ у Photoshop розміром 256x128 пікселів у форматі RGB 72dpi. Далі залити фон чорним, розділити навпіл по горизонталі та на 4 рівні частини по вертикалі за допомогою направляючих, а також розставити направляючі по межах області. В одній половині створити світлячка за допомогою еліпса, схожого на приклад на рисунку 5 зліва – спочатку голову, потім витягнуте тіло, копіюючи попередній шар, потім крила, роблячи копію з шару з тілом. Далі зробити групу з шарів з крилами, надати цій групі значення **Opacity** 50% та додати ефект тіні, де виставити параметри, наведені на рисунку 6а. Далі скопіювати шар з головою та створити у кінці тільки трохи витягнуту частину, яка буде світити. Для цього шару додати ефект зовнішнього світіння та задати наступні параметри, наведені на рисунку 6б. Далі створити групу з попередніх шарів, створити її дуплікат, перетягнути її в праву половину та трохи видозмінити положення крил відповідно до прикладу, представленому на рисунку 5. Потім зберегти файл у форматі .jpeg та втягнути

цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

9. Створити у цій же папці **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у попередніх пунктах;

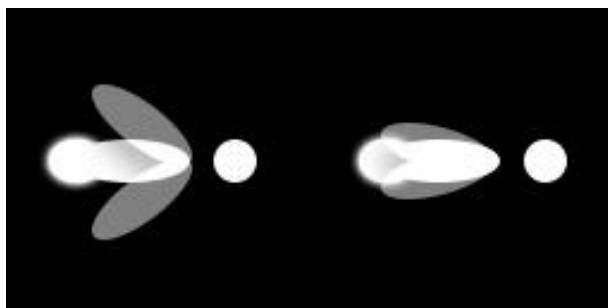
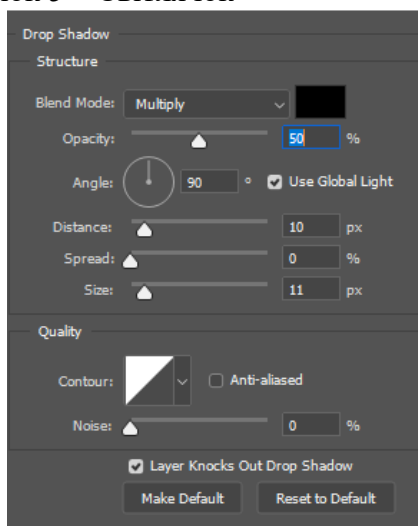
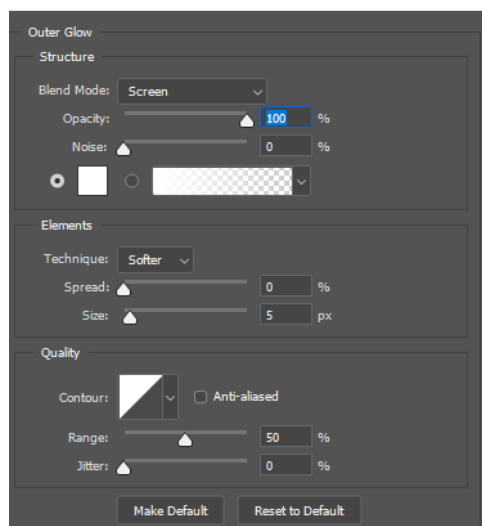


Рисунок 5 – Світлячок



а)



б)

Рисунок 6 – Параметри налаштування ефектів

10. Далі треба налаштувати зовнішній вигляд світлячків, аби це не був просто квадрат з 2 елементами. Для цього треба налаштувати:

- **Texture Sheet Animation** – підключити і встановити значення **Tiles X=2, Y=1** та **Cycles – 30**;
- **Renderer/Renderer Mode** – Stretched Billboard
- **Renderer/Length Scale** – встановити значення -1 (таким чином світлячки трохи крутяться при русі);

Загалом можна ще позмінювати попередні параметри. У результаті буде отриманий ефект світлячків з анімацією крил, який можна підв'язати до маркера шляхом перетягування **Particle System** на маркер у полі **Hierarchy**

3. Ефект вогню

Даний ефект поєднує декілька елементів **Particle System**. Для його створення треба:

1. У полі **Hierarchy** створюється елемент **Particle System** шляхом ПКМ/Effect/Particle System.

2. У полі **Transform** задається розмір та положення площі, яку займатиме елемент.

3. Далі необхідно провести попереднє налаштування самого елемента. Для цього проставити наступні параметри:

- **Start Lifetime** – 3;
- **Start Speed** – 4;
- **Start Color** – встановити сіруватий колір;
- **Start Size** – 2;
- **Shape** – встановити режим Cone та Angle = 15;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time ± 160 .

4. Наступним кроком необхідно створити текстуру для основи вогню. Для цього спочатку треба створити новий документ у Photoshop розміром 512x512 пікселів у форматі RGB 72dpi. Далі залити фон чорним, розділити навпіл по горизонталі та вертикалі за допомогою направляючих та створити новий шар. На цьому шарі створити 4 елементи (кожний у своїй четверті) за допомогою пензлика із розміром Opacity 100% та розміром 30 пікселів. Має вийти щось схоже на приклад, наведений на рисунку 7а. Потім необхідно взяти гумку з розміром 40 пікселів та дещо підтерти ці елементи, щоб вони були схожі на частинки полум'я. Має вийти щось схоже на приклад на рисунку 7б. Далі треба застосувати ефект Motion Blur (Розмиття у русі) із значенням кута 90. Зберегти отриманий файл у форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

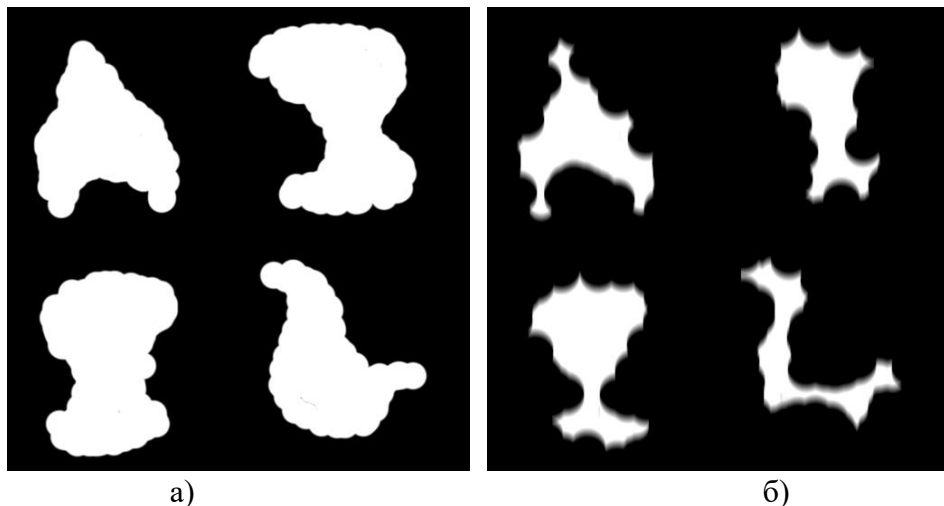


Рисунок 7 – Приклад текстури

5. Створити у цій же папці **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле Albedo та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у попередніх пунктах;

6. Далі треба налаштувати зовнішній вигляд частинок вогню, аби це не був просто квадрат з 4 елементами. Для цього треба налаштувати:

- **Texture Sheet Animation** – підключити і встановити значення Tiles X=2, Y=2;
- **Frame over Time** – від 0 до 3,9996 (таким чином обирається наближений зовнішній вигляд іскорки відповідно до створеного у 5 пункті елементу);

7. Далі слід налаштувати параметр **Size over lifetime** – обрати цей пункт, клацнути на полі **Size** та сформувати криву, схожу на представлену на рисунку 8, а також поставити

початкове значення 2 у полі редагування кривої у лівому верхньому куті. Це дозволить зробити зміну розміру більш плавною та реалістичною;

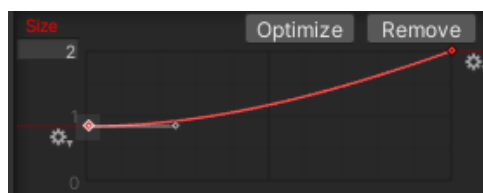


Рисунок 8 – Крива зміни розміру

8. **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%. Тоді слід налаштувати колір. Має бути 3 бігунки у нижній частині діапазону – жовтий, рудий та червоний. Зайві можна видалити.

9. Обрати параметр **Start Rotation** та встановити **Random Between Two Constants** і значення від -15 до 50 (це дозволить уникнути повторів елементів один за одним).

10. Далі треба створити джерело вогню. Натискаємо ПКМ на **Particle System** у полі **Hierarchy** та знову обираємо **Effect/Particle System**.

11. Необхідно провести попереднє налаштування самого елементу. Для цього проставити наступні параметри:

- **Start Lifetime** – 1,2;
- **Start Speed** – 2,5;
- **Start Color** – встановити темно-сірий колір;
- **Start Size** – 2;
- **Shape** – встановити режим Cone та Angle = 25, Radius = 0.2;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time ± 20 ;
- Скопіювати колір з основного **Particle System** і вставити його до того ж самого поля цього дочірнього.

12. Наступним кроком необхідно створити текстуру для світіння джерела. Для цього спочатку треба створити новий документ у Photoshop розміром 100x100 пікселів у форматі RGB 72dpi (див. рис. 9). Далі залити фон чорним, розділити навпіл по горизонталі та вертикалі за допомогою направляючих та створити новий шар. На цьому шарі створити за допомогою інструменту градієнт (обрати радіальний та у виборі кольору зробити його білим з прозорістю підкладкою), провівши ним від центру прямо праворуч до межі зображення. Задати Opacity шару 50%. Потім створити новий шар, виділити область за допомогою круглого інструменту для виділення та за допомогою інструменту градієнт з тими ж параметри створити радіальний градієнт, провівши їм від центру прямо праворуч. Далі створити новий шар та створити в центрі коло за допомогою еліпса із рамкою та заливкою білого кольору. Зберегти отриманий файл у форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;



Рисунок 9 – Текстура для джерела вогню

13. Створити у цій же папці **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у пункті 10.

14. Трошки посунути елемент нижче, аби здавалося, що це основа вогню.

15. Далі слід налаштувати параметр **Size over lifetime** – обрати цей пункт, клацнути на полі **Size** та сформувати криву, схожу на представлену на рисунку 8, а також поставити початкове значення 2 у полі редагування кривої у лівому верхньому куті.

16. Створити дим. Для цього скопіювати попередній **Particle System**, що для основного вогню. Потім змінити деякі параметри, а саме:

- **Start Lifetime** – 3;
- **Start Speed** – 4;
- **Start Color** – встановити темно-сірий колір (майже чорний) і прибрати **Color over lifetime**;
- **Start Size** – 2;
- **Shape** – встановити режим **Cone** та **Angle** = 30, **Radius** = 1;
- **Order in Layer (Renderer)** – встановити значення -1, а у основного **Particle System** – встановити значення 1;
- У основного **Particle System** встановити режим **By Distance** у полі **Sort Mode (Renderer)**.

17. Остання частина – створення іскор. Для цього треба спочатку натиснути ПКМ на **Particle System** у полі **Hierarchy** та знову обрати **Effect/Particle System**. Потім проставити наступні параметри:

- **Start Lifetime** – 5;
- **Start Speed** – 2;
- **Start Color** – встановити колір рудий основного елементу зі значенням альфа-каналу 80;
- **Start Size** – від 0,1 до 0,2;
- **Shape** – встановити режим **Cone** та **Angle** = 33, **Radius** = 0.3;
- **Emission** – обрати даний пункт та поставити значення **Rate Over Time** ±30.
- Підключити параметр **Noise**.

18. Перетягнути створений у пунктах 12-13 матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у пункті 17.

Загалом можна ще позмінювати попередні параметри. Також для легкості переміщення можна зробити системи частинок та диму дочірніми елементами системи вогню, перетягнувши їх на неї у полі **Hierarchy**. У результаті буде отриманий ефект вогню, який можна підв'язати до маркера шляхом перетягування **Particle System** на маркер у полі **Hierarchy**.

4. Ефект входження у потік Wrap Mode

1. У полі **Hierarchy** створюється елемент **Particle System** шляхом ПКМ/Effect/Particle System.

2. У полі **Transform** задається розмір та положення площі, яку займатиме елемент.

3. Далі необхідно провести попереднє налаштування самого елемента. Для цього проставити наступні параметри:

- **Start Lifetime** – 2;
- **Start Speed** – 0;
- **Start Color** – встановити Gradient та проставити різні кольори;
- **Start Size** – від 0,15 до 0,25;
- **Shape** – встановити режим Cone, Angle = 60, Radius = 4, Radius Thickness = 0 (аби частинки генерувалися по краях площі системи), Emit From = Volume, Length = 3, Rotation X = -90 (аби частинки летіли на спостерігача у камеру);
- **Emission** – обрати даний пункт та поставити значення Rate Over Time ± 100 ;
- **Velocity over Lifetime** – для швидкості по осі Y поставити значення ± 90 .

4. Наступним кроком необхідно створити текстуру для основи елементу потоку. Для цього спочатку треба створити новий документ у Photoshop розміром 100x100 пікселів у форматі RGB 72dpi. Далі залити фон чорним, розділити навпіл по горизонталі за допомогою направляючих. Створити коло за допомогою еліпса та відкласти від нього 2 прямі, що закінчуються в одній точці. Зафарбувати простір між колом та прямими, растровизувавши одну з них. Має вийти щось схоже на приклад на рисунку 10. Зберегти отриманий файл у форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

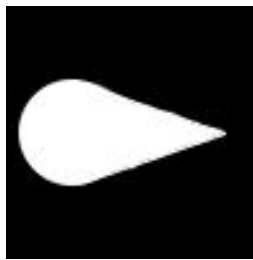


Рисунок 10 – Приклад текстури

5. Створити у цій же папці **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у попередніх пунктах.

6. Провести певне налаштування частинок потоку, а саме:

- **Renderer/Renderer Mode** – Stretched Billboard;
- **Renderer/Length Scale** – встановити значення 25.

Загалом можна ще позмінювати попередні параметри. У результаті буде отриманий ефект входження у потік Wrap Mode, який можна підв'язати до маркера шляхом перетягування **Particle System** на маркер у полі **Hierarchy**

5. Ефект магічної руни

1. Завантажити у середовище ПЗ Photoshop зображення руни, схожій на приклад на рисунку 11, та зробити його інверсію Image/Adjustments/Invert. Зберегти отриманий файл у

форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**.



Рисунок 11 – Приклад зображення для руни

2. У полі **Hierarchy** створюється елемент **Particle System** шляхом ПКМ/Effect/Particle System.

3. Створити у папці з пункту 1 **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Units** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у попередньому пункті.

4. У полі **Transform** задається розмір та положення площі, яку займатиме елемент.

5. Провести наступне налаштування:

- **Start Lifetime** – 3;
- **Start Speed** – 0;
- **Start Color** – встановити будь-який колір;
- **Start Size** – 3;
- **Shape** – прибрати;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time 1
- **Max Particles** – 1;
- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%;
- Підрегулювати **Size over Lifetime** для більшої плавності анімації;
- **Rotation by Speed** – обрати та поставити значення ± 60 ;
- **Renderer Mode** – Horizontal billboard;
- **Rendering/Max Particle size** – 3;

6. Далі треба створити пульсацію від руни. Для цього спочатку треба скопіювати попередній елемент в полі **Hierarchy** та зробити новостворений його дочірнім елементом, перетягнувши його на назву основного.

7. Далі треба створити анімацію пульсації для руни. Для цього спочатку треба створити новий документ у Photoshop розміром 2000x2000 пікселів у форматі RGB 72dpi. Далі залити фон чорним, розділити навпіл по горизонталі та вертикалі за допомогою направляючих. Створити коло за допомогою еліпса без заливки з білим контуром товщиною 15 пікселів та додати до нього ефект зовнішнього світіння, параметри якого представлені на рисунку 12. Зберегти отриманий файл у форматі .jpeg та втягнути цей файл

у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

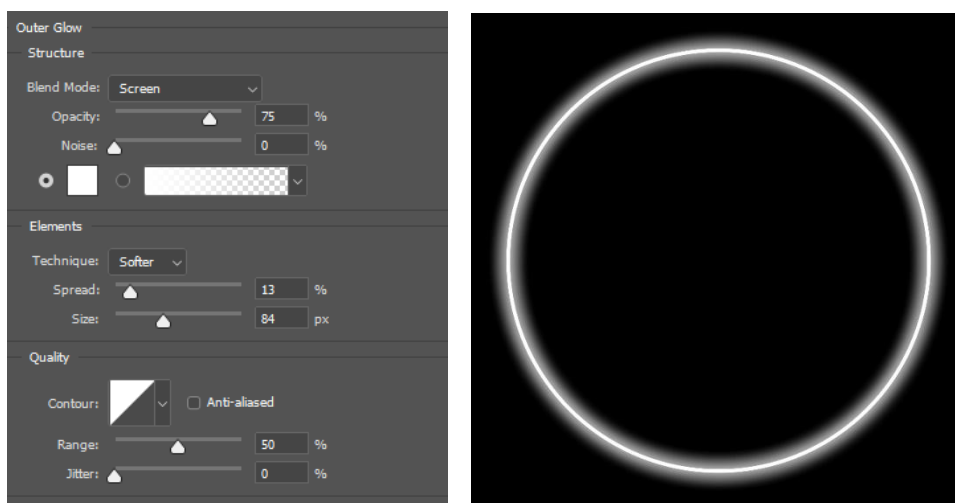


Рисунок 12 – Параметри зовнішнього світіння для текстури та її зовнішній вигляд

8. Створити у папці з пункту 7 **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створене у попередньому пункті зображення на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** елементу **Particle System**, створеного у пункті 6.

9. Налаштувати певні параметри:

- **Start Lifetime** – 3;
- **Start Speed** – 0;
- **Start Size** – 5;
- **Shape** – прибрати;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time 10
- **Max Particles** – 10;
- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%, а також можна створити певний градієнт для зафарбовування кілець;
- Підрегулювати **Size over Lifetime** для більшої плавності анімації;
- **Rotation by Speed** – прибрати;
- **Renderer Mode** – Horizontal billboard

10. Далі можна створити анімацію кілець вгору від руни. Для цього слід скопіювати попередній створений елемент, та провести певні налаштування:

- **Start Lifetime** – 2;
- **Start Speed** – 1;
- **Start Size** – 3;
- **Simulation Speed** – 0.8;
- **Shape** – прибрати;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time 4;
- **Max Particles** – 25;
- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%, а також можна створити певний градієнт для зафарбовування кілець;

11. Також можна створити дим над руною. Для цього створити новий елемент **Particle System** та зробити його дочірнім основного.

12. Для цього спочатку створити новий документ у Photoshop розміром 128x128 пікселів у форматі RGB 72dpi. Далі залити фон чорним, взяти пензлик з Opacity 30% та розміром 10 пікселів та намалювати щось схоже на приклад на рисунку 13. Зберегти отриманий файл у форматі .jpeg та втягнути цей файл у поле **Project** у новостворену папку у папці **Asset**, вибрати його та поставити режим у полі **Texture Type** «**Sprite 2D and UI**», натиснути **Apply**;

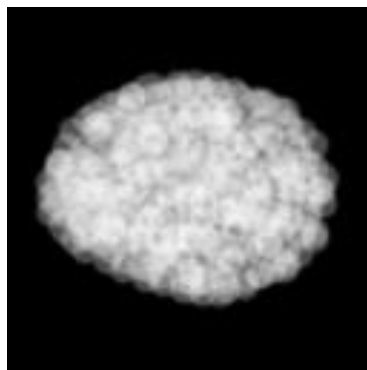


Рисунок 13 – Приклад текстури

Створити у відповідній папці з матеріалами **Material**, поставити у полі **Inspector** у полі **Shader** обрати **Particles/Standard Unlit** та у полі **Render Mode** режим **Additive**, перетягнути створену текстуру у попередньому пункті на поле **Albedo** та перетягнути даний матеріал у поле **Renderer/Material** поля **Inspector** цього елементу диму **Particle System**.

13. Провести певні налаштування:

- **Start Lifetime** – 5;
- **Start Speed** – 0,25;
- **Start Color** – зробити альфа такою, аби дим був прозорий;
- **Start Size** – 0,5;
- **Start Rotation** – від 0 до 180;
- **Simulation Speed** – 0.8;
- **Shape** – обрати режим Cone та Angle=70;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time ± 100 ;
- **Max Particles** – 1000;
- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%, а також можна створити певний градієнт для зафарбовування диму.

14. Також можна створити частинки, що вилітають з руни. Для цього знову ж таки створюється новий дочірній елемент **Particle System**. Потім підвантажується текстура (див. п.12 попереднього ефекту) та налаштовуються певні параметри, а саме:

- **Start Lifetime** – 5;
- **Start Speed** – 1;
- **Start Color** – зробити альфа такою, аби частинки були прозорі;
- **Start Size** – 0,05;
- **Shape** – обрати режим Cone та Angle=0;
- **Emission** – обрати даний пункт та поставити значення Rate Over Time ± 10 ;
- **Max Particles** – 1000;

- **Color over lifetime** – обрати цей пункт, клацнути на білому полі Color, встановити мітки 10% та 90% на верхній частині діапазону та встановити значення Alpha 0 для 0% та 100%, а також можна створити певний градієнт для зафарбовування частинок;
- **Rendering/Max Particle Size** – 0.5;
- **Renderer Mode** – Billboard

Загалом можна ще позмінювати попередні параметри. Також для легкості переміщення можна зробити системи кілець, частинок та диму дочірніми елементами системи пентаграми, перетягнувши їх на неї у полі **Hierarchy**. У результаті буде отриманий ефект магічної руни, який можна підв'язати до маркеру шляхом перетягування **Particle System** на маркер у полі **Hierarchy**

Хід роботи

1. Створити ефект, який сюжетно підходить під створений додаток комп'ютерного практикуму №2 (можна з представлених у прикладі, а можна й свій).
2. Підв'язати його під маркер та створити новий додаток.
3. Елемент доповненої реальності повинен мати якусь логіку та ідею.
4. Створити звіт з виконаної роботи з супровідними скріншотами.

Контрольні запитання:

1. Особливості ефекту Particle System.
2. Параметри та модулі ефекту Particle System.
3. Як налаштувати анімацію текстури для Particle System?
4. Як налаштувати зміну кольору та прозорості частинок?
5. Як накласти матеріал на частинки?

СТВОРЕННЯ ДОДАТКУ З ЕЛЕМЕНТАМИ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ЗА ДОПОМОГОЮ UNITY

Мета роботи: вивчення особливостей створення сцен з різним наповненням, кнопки переходу між ними, кнопки управління персонажем, а також різні варіанти меню

Особливості виконання

1. Створення сцен та налаштування їх порядку

Спочатку треба у папці **Scenes** створити новий файл – нову сцену (**Project/Scenes/ПКМ/Create/Scene**). Далі треба створити наповнення кожної сцени. Сам простір сцени задається за допомогою **Canvas (Hierarchy/ПКМ/Create/UI/Canvas)**. Далі треба задати його розмір – вкладка **Game** та обрати розмір у третій вкладці цього нового меню – можна один з існуючих або задати свій власний, обравши **+**. **Цей розмір буде однаковим для всіх сцен** – вони автоматично будуть змінювати свій розмір при його налаштуванні. Далі треба створити наповнення сцени – створити та налаштувати положення, розмір та вигляд текстових елементів (**Hierarchy/ПКМ на Canvas /Create/UI/Legacy/Text**), зображень (**Hierarchy/ПКМ на Canvas /Create/UI/Image(Panel)**), якщо потрібно, та кнопок (**Hierarchy/ПКМ на Canvas /Create/UI/Legacy/Button**), а також фону (**Hierarchy/ПКМ на Canvas /Create/UI/Legacy/Panel**).

Шрифти у тексті змінюються наступним чином – для цього треба підвантажувати відповідний шрифт у **Assets** та перетягнути його у відповідне поле налаштування гарнітури шрифту у полі **Inspector**. Для зображення треба задавати параметр **Texture Type Sprite (2D and UI)** – і те, яке в подальшому стане кнопкою, і просто зображення. Для цього його треба підвантажити в **Assets** та двічі клацнути на ньому, а далі вже обрати **Texture Type Sprite (2D and UI)**. Потім вже його можна підвантажувати у відповідне меню об'єкта **Image** у полі ієрархії. Підготувати таким чином декілька сцен.

Далі треба перетягнути усі сцени до **Hierarchy**. Розставити їх відповідним чином. Потім перейти **File/Build Settings** та натиснути кнопку **Add open scenes**. За потреби можна й у цьому меню розставити сцени так, як потрібно, обравши відповідну сцену та перетягнувши її над чи під інші, та запам'ятати їх порядковий номер (його можна у будь-який момент переглянути, повернувшись до цього меню). **Сцена з номером 0** буде першою, яку Ви побачите при завантаженні додатку на телефоні.

2. Різні варіанти меню

а) меню як сцена – створити меню відповідно до пункту 1, створивши нову сцену та використовуючи **UI/Canvas** з подальшим його наповненням

б) Створення вікна завантаження

Створити нову сцену у папці **Scenes (ПКМ/Create/Scene)**. Створити перший **Canvas** у полі ієрархії та його наповнення (**Panel, Image, Text, Button** тощо) – це буде та сцена, що зустрічатиме користувача перед завантаженням меню завантаження іншої сцени. Далі створити ще один **Canvas** необхідного розміру та з таким ж характеристиками, що й перший та наповнити його – це буде меню завантаження. Створити для нього або **Image** або **Panel** для створення кольору фону. Додати текст та задати його положення, параметри та написати, наприклад, **Loading...** та

елемент Slider через UI (задати його параметри). Також можна створити та додати анімаційне зображення, яке буде відображати та доповнювати процес завантаження. Для цього розробляємо декілька положень одного й того ж об'єкту, наприклад, як на рисунку 1 та підвантажуюмо їх у юніті. Потім перетаскуємо початковий варіант на сцену завантаження на другий Canvas. Далі треба перейти в анімацію (Window/Animation/Animation). Далі у новому вікні натиснути Create (у правій частині, якщо не видно, то треба вікно трошки роздвинути), зберегти аніматор (нічого не змінювати, як дає за замовчуванням, так і зберігати), і натиснути на кнопку для запису (червоний кружечок як при зйомці відео) та перетягнути у поле підвантажені зображення на певні проміжки часу, налаштувавши їх за потреби. Для створення будь-якої іншої анімації необхідно ввімкнути запис, перетягнути бігунок у правій частині цього меню у відповідне місце на часовій шкалі і зробити якусь зміну, наприклад, збільшити об'єкт, потім аналогічним чином за потреби додати ще точок, які можна рухати в подальшому, затиснувши ПКМ та перемістивши у відповідне місце часової шкали. Для видалення необхідно обрати ключову точку та натиснути Delete на клавіатурі. Так, до речі, можна створити й анімацію для кнопок чи інших об'єктів.



Рисунок 1 – Приклад зображення для завантаження

Створити скрипт LoadingScene та прописати у ньому наступне:

1. Якщо у Player Settings / Other Settings / Active Input Handling стоїть Input System

Package

```
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using UnityEngine.InputSystem;
public class LoadingScene : MonoBehaviour
{
    public GameObject loadingScreen;
    public Slider scale;
    public float fillSpeed = 0.5f; // швидкість анімації заповнення
    public void Load()
    {
        loadingScreen.SetActive(true);
        StartCoroutine(DelayedLoad());
    }
    IEnumerator DelayedLoad()
    {
        yield return null;
        StartCoroutine(LoadAsync());
    }
    IEnumerator LoadAsync()
    {
        AsyncOperation loadAsync = SceneManager.LoadSceneAsync("menu");
        loadAsync.allowSceneActivation = false;
        float displayedProgress = 0f; // поточне значення для плавного оновлення
        while (!loadAsync.isDone)
        {
            // Обчислюємо фактичний прогрес (0 → 1)
            float targetProgress = Mathf.Clamp01(loadAsync.progress / 0.9f);
            // Плавно наближаємо значення слайдера до фактичного
            displayedProgress = Mathf.MoveTowards(displayedProgress, targetProgress, fillSpeed * Time.deltaTime);
            scale.value = displayedProgress;
            // Коли сцена завантажена (90%)
            if (loadAsync.progress >= 0.9f && displayedProgress >= 1f)
            {

```

```

        bool touchPressed = Touchscreen.current != null && Touchscreen.current.primaryTouch.press.isPressed;
        bool mousePressed = Mouse.current != null && Mouse.current.leftButton.wasPressedThisFrame;
        if (touchPressed || mousePressed)
        {
            yield return new WaitForSeconds(1.0f);
            loadAsync.allowSceneActivation = true;
        }
    }
    yield return null;
}
}
}

```

2. Якщо у Player Settings / Other Settings / Active Input Handling стоїть Input System

Package

```

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
public class LoadingScene : MonoBehaviour
{
    public GameObject loadingScreen;
    public Slider scale;
    public float fillSpeed = 0.5f; // швидкість анімації заповнення
    public void Load()
    {
        loadingScreen.SetActive(true);
        StartCoroutine(DelayedLoad());
    }
    IEnumerator DelayedLoad()
    {
        yield return null; // дочекайся одного кадру, щоб UI оновився
        StartCoroutine(LoadAsync());
    }
    IEnumerator LoadAsync()
    {
        // Завантажуємо сцену асинхронно (заміни "NextScene" на свою)
        AsyncOperation loadAsync = SceneManager.LoadSceneAsync("NextScene");
        loadAsync.allowSceneActivation = false;
        float displayedProgress = 0f;
        while (!loadAsync.isDone)
        {
            // Отримуємо фактичний прогрес завантаження (0 → 0.9)
            float targetProgress = Mathf.Clamp01(loadAsync.progress / 0.9f);
            // Плавню рухаємо слайдер до потрібного значення
            displayedProgress = Mathf.MoveTowards(displayedProgress, targetProgress, fillSpeed * Time.deltaTime);
            scale.value = displayedProgress;
            // ♦ Коли сцена завантажена і слайдер заповнений
            if (loadAsync.progress >= 0.9f && displayedProgress >= 1f)
            {
                // ♦ Старий Input System
                if (Input.touchCount > 0 || Input.GetMouseButtonDown(0))
                {
                    yield return new WaitForSeconds(1.0f);
                    loadAsync.allowSceneActivation = true;
                }
            }
            yield return null;
        }
    }
}

```

Перетягнути до першого канваса (**вікно, яке першим побачить користувач**) цієї сцени створений скрипт до інспектора. Він з'явиться як компонент цього об'єкту. У **LoadingScreen** цього компоненту перетягнути другий канвас з **елементами загрузки**, у **Scale** – Ваш слайдер. Ще можна задати швидкість прокрутки слайдеру. Відключити видимість другого канвасу з елементами загрузки (у полі **Inspector** біля назви цього елемента зліва треба зняти галочку).

Створити кнопку у першому канвасі – **Hierarchy / Обрати канвас / ПКМ / UI / Legacy / Button**, створити новий стан у **On Click**, перетягти у ліве поле перший канвас зі скриптом з попереднього пункту та обрати метод **LoadScene/Load()**.

Перейти **File/Build Settings** та натиснути кнопку **Add open scenes**.

в) Створення меню, що викликається при натисканні на кнопку

Створити кнопку для виклику випадаючого меню (наприклад на сцені з AR-елементами). Створити візуал для кнопки за потреби (задати розміри, встановити **Texture Type Sprite (2D and UI)** якщо підвантажуватиметься зображення); створити **Canvas**, якщо ще не був у сцені створений, створити **Panel** у **Canvas** (налаштувати її параметри та розміщення) та створити у ньому необхідні дочірні елементи за задумкою.

Створити кнопку для меню, при натисканні на яку воно закриватиметься.

Відключити видимість панелі, так як початково користувач не має її бачити (у полі **Inspector** біля назви цього елемента зліва треба зняти галочку) і у кнопці меню створити новий стан у полі **On Click**, куди перетягуємо у нижнє ліве поле нашу панель, а у правій частині обираємо метод **GameObject/SetActive(bool)**, і ставимо галочку у чекбоксі, що з'явився. Таким чином дана панель викликатиметься, якщо натиснути на цю кнопку.

Для кнопки виходу з меню проробити ті ж самі дії, що описані вище, але галочку не ставити

3. Створення кнопок

Середовище Unity дозволяє створювати кнопки, які дозволятимуть здійснювати різноманітні запрограмовані дії. Наприклад, перехід на наступну сцену, повернення до попередньої, вихід з гри тощо. Для створення кнопки необхідно спочатку створити новий об'єкт **UI/Button** у полі **Hierarchy**. Далі за допомогою параметрів (**Rect Transform, Source Image** тощо) поля **Inspector** можна налаштувати положення, розмір, вирівнювання кнопок, їх відступи від країв, підписи, зовнішній вигляд тощо. Для видозміни зовнішнього вигляду кнопок слід підвантажити необхідні зображення та задати **Texture Type Sprite (2D and UI)** та закинути відповідне зображення у поле **Source Image** кнопки. Процес програмування кнопок наступний: можна створити новий скрипт у полі **Assets**, прописати відповідний код з таблиці 1 відповідно до функцій кнопки, створити порожній об'єкт у **Hierarchy**, перетягнути на нього файл коду та виконати певне налаштування кнопки – у полі **OnClick** кнопки створити новий стан, що відповідатиме за те що відбуватиметься, коли на кнопку буде натиснуто, далі в нього у відповідне порожнє поле закинути створений порожній об'єкт із скриптом, і потім у полі де потрібно обрати якусь функцію у цьому ж меню обрати створений клас (**public class «НазваКласу» : MonoBehaviour**) та знайти створений метод, наприклад, **ChangeScenes(int _number)**. Після цього в залежності від методу може бути ще додаткові налаштування – задання номеру сцени, прописання **url** сторінки і тд. Можна навпаки створити порожній об'єкт у **Hierarchy**, обрати його, натиснути на кнопку **Add Component** у полі **Inspector** та обрати **New Script**, натиснути на ньому, знайти його у ассетах, зайти у нього та написати код, представлений у таблиці 1 відповідно до функцій кнопки, а потім зробити таке ж налаштування кнопки.

Таблиця 1 – Приклад кодів для управління кнопками

Рядок коду	Характеристика, за яку відповідає
Перехід між сценами	
<code>using System.Collections;</code>	Підключення бібліотек
<code>using System.Collections.Generic;</code>	
<code>using UnityEngine;</code>	
<code>using UnityEngine.SceneManagement;</code>	
<code>public class LoadScenes : MonoBehaviour</code>	Як тут названий клас, так має називатися і код
<code>{</code>	
<code> public void ChangeScenes(int number)</code>	Задання методу ChangeScenes
<code>{</code>	
<code> SceneManager.LoadScene(number);</code>	Перехід на сцену з певним номером
<code>}</code>	
<code>}</code>	
Вихід з додатку	
<code>using System.Collections;</code>	Підключення бібліотек
<code>using System.Collections.Generic;</code>	
<code>using UnityEngine;</code>	
<code>using UnityEngine.SceneManagement;</code>	
<code>public class exitgame : MonoBehaviour</code>	Як тут названий клас, так має називатися і код
<code>{</code>	
<code> public void ExitPressed()</code>	Задання методу ExitPressed
<code>{</code>	
<code> Application.Quit();</code>	Вихід з системи
<code> Debug.Log("Exit pressed!");</code>	Необов'язковий рядок, але він дасть змогу перевірити правильність коду в меню Unity
<code>}</code>	
<code>}</code>	
Оновити сцену	
<code>using System.Collections;</code>	Підключення бібліотек
<code>using System.Collections.Generic;</code>	
<code>using UnityEngine;</code>	
<code>using UnityEngine.SceneManagement;</code>	
<code>public class restart : MonoBehaviour</code>	Як тут названий клас, так має називатися і код
<code>{</code>	
<code> public void RestartLevel()</code>	Задання методу для виконання процесу перезавантаження сторінки
<code>{</code>	
<code> SceneManager.LoadScene("lab_3");</code>	Перезавантаження сцени з AR (у даному випадку вона називається lab 3)
<code>}</code>	
<code>}</code>	
Перехід на сторінку в Інтернеті	
<code>using System.Collections;</code>	Підключення бібліотек
<code>using System.Collections.Generic;</code>	
<code>using UnityEngine;</code>	
<code>public class OpenWeb : MonoBehaviour</code>	Як тут названий клас, так має називатися і код
<code>{</code>	
<code> public void UrlOpen(string url)</code>	Задання методу UrlOpen
<code>{</code>	
<code> Application.OpenURL(url);</code>	Завантаження сторінки по заданому посиланню при натисканні на створену кнопку
<code>}</code>	
<code>}</code>	

4. Створення контролеру руху моделі

У сцені, де є персонаж, треба створити контролер. Для цього спочатку треба завантажити джойстик. Для цього переходимо у вкладку **Window/Asset Store**. У вікні, що

з'явиться обираємо **Search Online** і у вікні браузера обираємо джойстик, наприклад **Joystick Pack**. Імпортуємо його у юніті. Далі підвантажуємо його з папки, що з'явилася в меню **Assets Fixed Joystick/Prefabs/Fixed Joystick** під створений **Canvas** та регулюємо його позицію та розмір за потреби за допомогою меню **Inspector**. Також треба буде підвантажити анімацію руху персонажу, коли він іде чи біжить;

Далі створюємо новий **C#Script** у папці зі скриптами поля **Assets**. У ньому прописуємо наступний код:

<code>using System.Collections;</code>	Підключення бібліотек
<code>using System.Collections.Generic;</code>	
<code>using UnityEngine;</code>	
<code>[RequireComponent (typeof(Rigidbody),</code>	Фіксування важливих компонентів Rigidbody та
<code>typeof(BoxCollider))]</code>	BoxCollider, щоб вони не зникли у Вашого
	персонажу
<code>public class gameController1 : MonoBehaviour</code>	У даному випадку задається поведінка скрипта
	<code>gameController1</code> , проте ця назва буде такою, як
	названий Ваш C#Script
<code>{</code>	
<code>[SerializeField] private Rigidbody _rigidbody;</code>	Створення полів-посилань на змінні, що будуть
<code>[SerializeField] private FixedJoystick _joystick;</code>	зберігати інформацію про такі компоненти, як
<code>[SerializeField] private Animator _animator;</code>	Rigidbody, joystick, animator, moveSpeed
<code>[SerializeField] private float _moveSpeed;</code>	(швидкість руху)
<code>private void FixedUpdate()</code>	Задання методу для врегулювання фізичної
	поведінки фігури
<code>{</code>	
<code>_rigidbody.velocity = new Vector3(_joystick.Horizontal *</code>	
<code>_moveSpeed, _rigidbody.velocity.y, _joystick.Vertical *</code>	
<code>_moveSpeed);</code>	Задання руху об'єкта
<code>}</code>	
<code>}</code>	

Зберігаємо результат, повертаємось до Unity і перетягуємо створений код на фігуру. У полі **Inspector** Вашої фігури у підпункті зі скриптом має з'явитися підпункти **Rigidbody**, **joystick**, **animator** та **moveSpeed**, проте якщо вони не з'явилися, то треба у полі **Inspector** підвантажити пункти **Rigidbody** і **BoxCollider**. Потім у поле **Rigidbody** треба перетягнути Вашу фігуру з поля **Hierarchy**, **joystick** – Важ джойстик, **animator** – знову ж таки Вашу фігуру, а також задати певну швидкість руху у полі **moveSpeed**;

Далі здійснюємо ряд налаштувань поля **Rigidbody** – знімаємо галку з **Use Gravity**, щоб персонаж не зникав, **Interpolate** – **Interpolate** (для більш чіткого руху), **Constraints/Freeze Rotation** – для заборони зайвих поворотів;

Повертаємось до коду та пишемо наступне після поля із заданням руху об'єкта:

<code>if (_joystick.Horizontal != 0 _joystick.Vertical != 0)</code>	У випадку якщо джойстик кудись рухається
<code>{</code>	виконується наступне
<code>transform.rotation =</code>	
<code>Quaternion.LookRotation(_rigidbody.velocity);</code>	Фігура повертається у напрямку руху джойстика
<code>_animator.SetBool("isRunning", true);</code>	Виконується анімація руху
<code>}</code>	
<code>else</code>	У іншому випадку (коли джойстик нерухомий)
<code>_animator.SetBool("isRunning", false);</code>	Відбувається анімація спокою

Далі переходимо до аніматора, який контролює анімацію руху об'єкта, натискаючи на нього двічі у полі **Assets**. Меню аніматора поділено на 2 частини: у лівій є дві вкладки – **Layers** та **Parameters**, у правій можна побачити різні стани, які контролює аніматор та взаємозв'язки між ними. Спочатку необхідно перейти у вкладку **Parameters** та створити новий параметр **isRunning** шляхом натискання на «+» та вибору типу параметру як **Bool**.

Далі створюємо новий стан у правому полі, називаємо його, наприклад, **Running**, задаємо для нього анімацію руху персонажа, коли він іде чи біжить, а також задаємо до нього шлях від Вашого основного стану з початковою анімацією (забарвлений оранжевим кольором), натиснувши на початковому стані правою клавішею миші та вибравши пункт **make transition**. Далі обираємо створену стрілку-зв'язок, прибираємо галку з пункту **Has Exit Time** та у полі **Conditions** створюємо новий елемент і одразу з'являється створений попередньо параметр **isRunning** зі значенням **True**. Аналогічно створюємо зв'язок від новоствореного стану **Running** до початкового (забарвлений оранжевим кольором), задаємо ті ж самі параметри, тільки для **isRunning** ставимо значення **False** для повернення до початкової анімації;

Готовий код має наступний вигляд:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
[RequireComponent(typeof(Rigidbody), typeof(BoxCollider))]
public class gameController1 : MonoBehaviour
{
    [SerializeField] private Rigidbody _rigidbody;
    [SerializeField] private FixedJoystick _joystick;
    [SerializeField] private Animator _animator;
    [SerializeField] private float _moveSpeed;
    // Start is called before the first frame update
    private void FixedUpdate()
    {
        _rigidbody.velocity = new Vector3(_joystick.Horizontal * _moveSpeed, _rigidbody.velocity.y,
        _joystick.Vertical * _moveSpeed);
        if (_joystick.Horizontal != 0 || _joystick.Vertical != 0)
        {
            transform.rotation = Quaternion.LookRotation(_rigidbody.velocity);
            _animator.SetBool("isRunning", true);
        }
        else
            _animator.SetBool("isRunning", false);
    }
}
```

Головні моменти, які варто не забувати

1. Кожен рядок коду, окрім **private void «Назва методу»()**, **public class «Назва коду»: MonoBehaviour**, **[RequireComponent(typeof(Rigidbody), typeof(BoxCollider))]** має закінчуватися крапкою з комою «;», бо програма вважатиме, що дія ще не завершилася.
2. Дужки мають обов'язково закриватися
3. У даному рядку **public class «Назва коду» : MonoBehaviour** назва має точно відповідати назві коду. Наприклад, якщо код називається **gameController1**, то і даний рядок має мати наступний вигляд **public class gameController1: MonoBehaviour**.

Хід роботи

1. Створити додаток з декількома сценами, одна з яких меню завантаження сцени, одна меню самого додатку, одна – сцена з AR і інші за задумом. Має бути мінімально 4-5 сцен.

2. Створити кнопки переходу між сценами, а також додаткові (мінімум 2) за потреби – кнопка оновлення, виходу з додатку взагалі, викликання меню, переходу по посиланню або будь-якого іншого ефекту.

3. Створити контролер для управління рухом моделі.

4. Створити звіт з виконаної роботи з супровідними скріншотами.

Контрольні запитання:

1. Алгоритм створення переходу між сценами.

2. Алгоритм створення кнопок.

3. Алгоритм створення контролеру для управління персонажами.

ВИЗНАЧЕННЯ ВПЛИВУ ХАРАКТЕРИСТИК ЗАДРУКОВУВАНОВОГО МАТЕРІАЛУ НА ПОЛІГРАФІЧНЕ ВИКОНАННЯ AR-МАРКЕРІВ ТА ВІДТВОРЕННЯ НА ЕКРАНІ ЦИФРОВОГО КОНТЕНТУ

Мета роботи: визначення впливу характеристик матеріалу на поліграфічне виконання маркерів доповненої реальності.

Хід роботи

1. Створити тест-об'єкт, що міститиме прості маркери доповненої реальності (до 4 штук) зі зміною розміру для оцінювання чіткості їх відтворення, а також штрихове поле у горизонтальному напрямку зі зміною товщини ліній для оцінювання якості відтворення тонких ліній, їх спотворення на відбитках та якості країв.

2. Роздрукувати тест-об'єкт на декількох матеріалах з різними характеристиками.

3. Перевірити як характеристики матеріалу (колір, пористість, додаткове покриття, білість, глянець, непрозорість) впливатимуть на поліграфічне відтворення маркерів – товщину ліній, чорний колір та відповідність та якість відтворення тонких елементів та рисунку маркеру – за допомогою електронного мікроскопу та спектрофотометру.

4. Порівняти отримані значення з оригінальними, оцінити отримані відбитки візуально на наявність недоліків та зробити висновки стосовно впливу характеристик маркерів на їх поліграфічне виконання.

5. Підв'язати маркери під простий елемент доповненої реальності та перевірити як характеристики матеріалів впливають на відтворення маркерів на пристрої.

6. Створити протокол з результатами проробленої роботи.

Контрольні запитання:

1. Маркер доповненої реальності.
2. Види маркерів та їх характеристики.
3. Класифікаційні характеристики маркерів.

**ВИЗНАЧЕННЯ ВПЛИВУ ОСВІТЛЕННЯ НА БЕЗВІДМОВНІСТЬ
МАРКЕРІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ ПРОТЯГОМ
ВІДПОВІДНОГО ТЕРМІНУ ЕКСПЛУАТАЦІЇ ПРОДУКЦІЇ**

Мета роботи: визначення впливу характеристик освітлення на відтворення маркерів доповненої реальності.

Хід роботи

1. Створити тест-об'єкт, що міститиме три прості маркери доповненої реальності з різним ступенем деталізації рисунку маркеру (різний ступінь заповненості внутрішньорамкового простору) – 20%, 50%, 80%. Кожен маркер має мати три розміри – 100 x 100, 150 x 150, 200 x 200. Роздрукувати тест-об'єкти.

2. Протестити швидкість сканування зразків при нормальних умовах – при стандартному освітленні кімнати та зафіксувати час, який необхідний на відтворення кожного зразка (від початку сканування до моменту відтворення). Також виміряти початкові показники колірних координат маркерів за допомогою спектрофотометру.

3. Прикріпити зразки за склом на балконі або іншому місці з інтенсивними умовами освітлення. Зафіксувати початковий час на їх сканування та відтворення. Провести тестування зразків протягом двох тижнів за різних умов освітлення – вранці, вдень, ввечері та майже в сутінках.

4. Порівняти отримані дані з отриманими за нормальних умов використання маркерів. Оцінити різницю, враховуючи розмір маркерів та ступінь їх деталізації, побудувати графіки зміни показників безвідмовності маркеру, зробити висновок щодо впливу освітлення на вибір показників маркерів.

5. Провести вимірювання колірних координат маркерів у кінці дослідження, порівняти з початковими, визначити колірну різницю.

6. Сформулювати протокол з результатами проробленої роботи.

Контрольні запитання:

1. Які існують технології доповненої реальності?
2. Що впливає на відтворення маркерів доповненої реальності?
3. Які є основи для відтворення маркерів доповненої реальності?
4. Що не є доповненою реальністю?
5. Які особливості простого маркеру?

ВИЗНАЧЕННЯ ВПЛИВУ ОСОБЛИВОСТЕЙ ЗОБРАЖЕННЯ НА МОЖЛИВІСТЬ ЇХ ВИКОРИСТАННЯ ЯК AR-МАРКЕРІВ

Мета роботи: визначення впливу особливостей зображення на можливість їх використання у якості маркерів доповненої реальності.

Хід роботи

1. Отримати від викладача 10 зображень з різними характеристиками – логотип, картина, чорно-біле фото чи зображення у сепії, скульптура, фото у поганій якості тощо.

2. Оцінити візуально на можливість їх використання у якості маркеру доповненої реальності. Пояснити свій висновок

3. Завантажити зображення у базу Vuforia та отримати рейтинг кожного з них. Порівняти отриманий рейтинг із висунутими на попередньому етапі результатами оцінювання.

4. Протестувати отримані зображення – створити за допомогою Vuforia просту сцену, наповнити її інтерактивними елементами (можна використати проєкт з лабораторної 1-3) – та визначити час на сканування та стабільність трекінгу (зображення не мерехтить, «не злітає», не переміщується без руху пристрою).

4. Порівняти результати експериментального тестування з рейтингом, отриманим у Vuforia, та оцінити ступінь їх відповідності. Сформулювати рекомендації щодо вибору зображень для використання як зображення-маркер доповненої реальності.

5. Сформувати протокол з результатами проведеної роботи.

Контрольні запитання:

1. Які існують види доповненої реальності за ступенем взаємодії з користувачем?
2. Які існують види доповненої реальності за особливостями об'єкту розпізнавання?
3. Чим віртуальна реальність відрізняється від доповненої?
4. Які існують реальності?
5. Що потрібно для створення доповненої реальності?

РЕКОМЕНДОВАНИЙ ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Створення інтерактивних медіа : навчальний посібник для студентів спеціальності 8.05150102 "Технології електронних мультимедійних видань" / О. С. Євсєєв. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 136 с. (Укр. мов.) .ISBN 978-966-676-608-6.
2. Мультимедійне видавництво: навчальний посібник для студентів спеціальності "Технології електронних мультимедійних видань" / О. І. Пушкар, О. С. Завгородня. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 204 с. (Укр. мов.) ISBN 978-966-676-619-2
3. Azuma, R. T. (1995). Predictive Tracking for Augmented Reality. [Doctoral thesis, University of North Carolina]. UNC Chapel Hill Department of Computer Science. Retrieved April 28, 2021, from <http://www.cs.unc.edu/~azuma/ARpresence.pdf>
4. R. Azuma, A Survey of Augmented Reality Presence: Teleoperators and Virtual Environments, pp. 355—385, August 1997.
5. P. Milgram and A. F. Kishino, Taxonomy of Mixed Reality Visual Displays Архивировано 3 ноября 2009 года. IEICE Transactions on Information and Systems, E77-D(12), pp. 1321—1329, 1994.
6. Slyusar, Vadym Augmented reality in the interests of ESMRM and munitions safety. Coordination problems of military technical and deensive industrial policy in Ukraine. Weapons and military equipment development perspectives/ VII International Scientific and Practical Conference. Abstracts of reports. - October 8–10, 2019. - Kyiv. - Pp. 193 - 194. (2019).
7. Yavuza M., Çorbacioğlu E., Başoğlu N. A., Daim T. U., Shayganc A. (2021). Augmented reality technology adoption: Case of a mobile application in Turkey. Technology in Society, 66, <https://doi.org/10.1016/j.techsoc.2021.101598>
8. Verkhova, G. V., Akimov S. V., Kotelnikov M. M. (2019). Markerless augmented reality technology in modern education. F JPIT, 20 (2), 29–35; <https://doi.org/10.25045/jpit.v10.i2.05>
9. Theodoropoulos A., Lepouras G. (2021) Augmented Reality and programming education: A systematic review. International Journal of Child-Computer Interaction, 30, <https://doi.org/10.1016/j.ijcci.2021.100335>
10. Волинець В. О. Віртуальна, доповнена і змішана реальність: сутність понять та специфіка відповідних комп'ютерних систем [Текст] / В. О. Волинець // Питання культурології. Мистецтво та культура, навчальні практики. – 2021. – doi: <https://doi.org/10.31866/2410-1311.37.2021.23732>
11. Литвинова С. Г. Концептуальні підходи до використання засобів доповненої реальності в освітньому процесі [Текст] / С. Г. Литвинова, О. Ю. Буров, С. О. Семеріков // Сучасні інформаційні технології та інноваційні методики навчання в підготовці фахівців: методологія, теорія, досвід, проблеми. Проблеми використання інформаційних технологій у сучасних закладах освіти. – 2020. - DOI: 10.31652/2412-1142-2020-55-46-62.
12. Щегельська Ю. П. Особливості застосування технологій доповненої реальності як інструмента перетворення друкованої продукції на тривимірну в практиці промоційних комунікацій [Текст] / Ю. П. Щегельська // Поліграфія і видавнича справа. Соціальні комунікації. – 2019. – doi: 10.32403/0554-4866-2019-1-77-101-110
13. Бізюк А. В. Вибір оптимального алгоритма розпізнавання маркера для доповненої реальності [Текст] / А. В. Бізюк, М. О. Мажуга // Системи обробки інформації. Інфокомунікаційні системи. – 2017. – DOI:10.30748/soi.2017.150.
14. Що таке доповнена реальність [Електронний ресурс] – Режим доступу: <https://teach-hub.com/scho-take-dopovnena-realnist/>, вільний. – Назва з екрану – Мова укр.
15. Що таке AR? Поняття доповненої реальності [Електронний ресурс] – Режим доступу: <https://www.adobe.com/ua/products/substance3d/discover/what-is-ar.html>, вільний. – Назва з екрану – Мова укр.
16. Augmented Reality, AR [Електронний ресурс] – Режим доступу: <https://www.it.ua/knowledge-base/technology-innovation/dopolnennaja-realnost-ar>, вільний. – Назва з екрану – Мова укр.

17. Що таке доповнена реальність (AR) і Як це працює [Електронний ресурс] – Режим доступу: <https://ulab.sumdu.edu.ua/uk/shho-take-dopovnena-realnist-ar-i-yak-ce-pracjuie>, вільний. – Назва з екрану – Мова укр.
18. Експеримент із технологією доповненої реальності у вебi (front-end only) [Електронний ресурс] – Режим доступу: <https://dou.ua/lenta/articles/web-ar/>, вільний. – Назва з екрану – Мова укр.
19. Історія та еволюція технології доповненої реальності [Електронний ресурс] – Режим доступу: <https://ts2.space/uk/історія-та-еволюція-технології-допов/>, вільний. – Назва з екрану – Мова укр.
20. Коротка історія VR та AR: від Virtual Boy до Reality Pro [Електронний ресурс] – Режим доступу: <https://ilounge.ua/ua/review/kortka-istoriya-vr-ta-ar/>, вільний. – Назва з екрану – Мова укр.
21. Jung, T., & Claudia tom Dieck, M. (2018). Augmented reality and virtual reality: Empowering human, place and business (Progress in IS). Cham, Switzerland: Springer. Retrieved April 26, 2021, from <https://www.worldcat.org/title/augmented-reality-and-virtual-reality-empowering-human-place-and-business/oclc/1008871983>
22. Prasad B., Goswamib M. L. (2021). Advancements in augmented reality. Materials Today: Proceedings, <https://doi.org/10.1016/j.matpr.2021.03.696>.
23. Catalán, A., Gidlöf F. (2018). Exploring the Use of Augmented Reality in the Experience Industry [Doctoral thesis, Upsala University]. Faculty of Science and Technology. April 28, 2021 <http://www.diva-portal.org/smash/get/diva2:1223688/FULLTEXT01.pdf>.
24. Busel, M. (n.d.). The 6 Biggest Challenges Facing Augmented Reality. November 17, 2020, from <https://medium.com/the-mission/the-6-biggest-challenges-facing-augmented-reality-8d48c470286d>
25. Синтаксис середовища Processing. Режим доступу: <https://processing.org/reference/>
26. Відеоуроки по середовищу Processing. Режим доступу: <https://processing.org/tutorials/>
27. Приклади по середовищу Processing. Режим доступу: <https://processing.org/examples/>
28. Нормативні документи
29. Періодичні фахові та професійні видання.

Інформаційні ресурси

1. Науково-технічна бібліотека ім. Г.І. Денисенка КПІ ім. Ігоря Сікорського <http://www.library.kpi.ua>
2. Електронний архів наукових та освітніх матеріалів КПІ ім. Ігоря Сікорського <http://www.ela.kpi.ua>

ДОДАТОК А

Приклад оформлення титульного аркушу

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий видавничо-поліграфічний інститут
Кафедра репрографії/ТПВ

Лабораторна робота №_

з дисципліни: «Технології доповненої реальності»

Виконав/ла:

Студент/ка _____-го курсу, групи _____

Прізвище та ініціали

Перевірив/ла:

посада, Прізвище та ініціали викладача

Київ – 202_