МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова

Навчально-науковий педагогічний інститут імені В.О. Сухомлинського

Кафедра методики викладання фізики та математики

«Допущений до захисту»

Завідувач кафедри

_____Віктор Січко

«3» грудня 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти «магістр»

зі спеціальності 014.04 Середня освіта (Математика)

на тему:

ВИКОРИСТАННЯ UNITY ДЛЯ ВИВЧЕННЯ ОСНОВ ПРОГРАМУВАННЯ: ВІЗУАЛІЗАЦІЯ АЛГОРИТМІВ ТА КОНЦЕПЦІЙ C#

Виконав: студент 6811м групи

_____ ***Тицький Б.Ю.***

(підпис)

Керівник роботи:

доцент, к.ф.-м.н.

(посада, науковий ступень, вчене звання)

_____ ***Січко В.М.***

(підпис)

Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий педагогічний інститут імені В.О. Сухомлинського
Кафедра методики викладання фізики та математики
Спеціальність 014.04 Середня освіта (Математика)
Освітня програма Середня освіта (Математика, інформатика)

«ЗАТВЕРДЖУЮ»
Гарант освітньої програми
_____Літвінова М.Б.
(підпис)
«22» жовтня 2024 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти «магістр»
студента: Тицького Богдана Юрійовича

1. Тема роботи: « Використання Unity для вивчення основ програмування: візуалізація алгоритмів та концепцій C#»
2. Керівник роботи: Січко Віктор Михайлович, кандидат фізико-математичних наук, професор Кафедри методики викладання фізики та математики, затверджений наказом вищого навчального закладу від «26» листопада 2025 року № 1383-УЧ
3. Строк подання студентом роботи: 18.11.2025 року
4. Вихідні дані до роботи:

Наукові розробки вітчизняних і зарубіжних вчених, матеріали наукових конференцій. Державний стандарт базової і повної загальної середньої освіти, Міністерство освіти і науки України.

5. Зміст розрахунково-пояснювальної записки:

Розділ 1. Загально-теоретична характеристика інформаційно-комунікаційних технологій в сучасній освіті.
Розділ 2. Експериментальна робота з впровадження інформаційно-комунікаційних технологій, як інструменту розвитку критичного мислення.

6. Перелік графічного матеріалу:
7. Консультанти розділів роботи: к. ф.-м. н., доцент Мельник О. В.
8. Дата видачі завдання: 16.10.2024 р.

КАЛЕНДАРНИЙ ПЛАН на 2024-2025 навчальний рік

№ з/п	Назва етапів кваліфікаційну роботи	Строк виконання етапів роботи	Примітка
1.	Затвердження теми і керівника кваліфікаційної роботи	16.09.2024	
2.	Видача завдання	16.10.2024	
3.	Підбір літератури, узгодження структури робіт	28.10.2024	
4.	Вступ. Розділ 1. Загально-теоретична характеристика інформаційно-комунікаційних технологій в січасній освіти.	30.11.2024	
5.	Розділ 2. Експериментальна робота з впровадження інформаційно-комунікаційних технологій , як інструменту розвитку критичного мислення.	21.03.2025	
6.	Вступ, висновки, список використаних джерел	12.10.2025	
7.	Захист магістерської роботи: магістерська електронний варіант; довідка з бібліотеки (перевірка на антиплагіат) магістерська роздрукована робота (прошита); наукова стаття у фаховому збірнику (тези) у співавторстві із керівником; відгук від керівника; рецензія зовнішня не нижче кандидата наук; презентація /доповідь (захист)	22.12.2025	
8.	Подача роботи в деканат	05.12.2025	

Студент _____ Тицький Богдан Юрійович
(підпис)

Керівник роботи _____ Січко Віктор Михайлович
(підпис)

АНОТАЦІЯ

Тицький Б. Ю. Використання Unity для вивчення основ програмування: візуалізація алгоритмів та концепцій C#. Кваліфікаційна робота зі спеціальності 014.04 Середня освіта (Математика) освітньої програми «Середня освіта (Математика)». Миколаїв: НУК ім. адм. Макарова, 2025. 103 сторінки.

В кваліфікаційній роботі було здійснено науково-теоретичне обґрунтування та експериментальну перевірку ефективності використання Unity для візуалізації алгоритмів та концепцій C# у вивченні основ програмування.

Здійснено аналіз науково-методичної літератури та мережі Інтернет з проблем методів навчання програмуванню, включаючи історичний розвиток підходів та порівняння традиційних та інтерактивних методів.

Проведено науково-теоретичне дослідження особливостей навчання програмуванню, зокрема історичного розвитку підходів до викладання, переваг та обмежень традиційних, проєктно-орієнтованих та ігрових методів, а також ролі візуалізації в освітньому процесі.

Розглянуто практичну реалізацію концепцій програмування в Unity: розробку базових проєктів для візуалізації алгоритмів, циклів, умовних операторів, функцій та об’єктно-орієнтованого підходу з використанням C#.

Обґрунтовано методи візуалізації в Unity, включаючи інтерактивні проєкти, гейміфікацію та інтеграцію теоретичних знань з практичними навичками для розвитку критичного мислення та мотивації студентів.

Розроблено методику використання Unity для інтерактивного навчання програмуванню з акцентом на візуалізацію алгоритмів та концепцій C#.

Експериментально перевірено ефективність розробленої методики візуалізації програмування в Unity, включаючи оцінку впливу на розуміння матеріалу, мотивацію та результати навчання.

Розроблено практичні рекомендації щодо впровадження Unity в навчальний процес для викладання програмування. Розроблені практичні рекомендації сприятимуть покращенню ефективності навчання програмуванню з використанням візуалізації в Unity.

Ключові слова: Unity, програмування, візуалізація, алгоритми, C#, навчання, методика.

ABSTRACT

Tytskyi B. Yu. Using Unity to study the basics of programming: visualization of algorithms and C# concepts. Qualification work in the specialty 014.04 Secondary education (Mathematics) of the educational program "Secondary education (Mathematics)". Mykolaiv: NUK named after adm. Makarov, 2025. 103 pages.

In the qualification work, a scientific and theoretical justification and experimental verification of the effectiveness of using Unity for visualizing algorithms and C# concepts in studying the basics of programming were carried out.

An analysis of scientific and methodological literature and the Internet on the problems of programming teaching methods, including the historical development of approaches and comparison of traditional and interactive methods, was carried out.

A scientific and theoretical study of the features of programming teaching was conducted, including the historical development of approaches to teaching, advantages and limitations of traditional, project-oriented and game methods, as well as the role of visualization in the educational process.

The practical implementation of programming concepts in Unity was considered: development of basic projects for visualizing algorithms, loops, conditional operators, functions and object-oriented approach using C#.

Methods of visualization in Unity were substantiated, including interactive projects, gamification and integration of theoretical knowledge with practical skills for the development of critical thinking and student motivation.

A methodology for using Unity for interactive programming teaching with an emphasis on visualizing algorithms and C# concepts has been developed.

The effectiveness of the developed methodology for visualizing programming in Unity has been experimentally tested, including an assessment of the impact on understanding the material, motivation and learning outcomes.

Practical recommendations for implementing Unity in the educational process for teaching programming have been developed. The developed practical recommendations will contribute to improving the effectiveness of programming teaching using visualization in Unity.

Keywords: Unity, programming, visualization, algorithms, C#, teaching, methodology.

ВСТУП

Актуальність нашого дослідження полягає в тому, що в умовах стрімкої цифровізації освіти та впровадження нової української школи програмування стало обов’язковим елементом шкільної інформатики, а також ключовою компетентністю XXI століття. Проте традиційні методи викладання (лекції, консольні програми, текстові підручники) часто призводять до втрати інтересу учнів, низької мотивації та поверхневого розуміння абстрактних алгоритмічних конструкцій. Unity як ігровий рушій із підтримкою C# дозволяє візуалізувати змінні, цикли, умови, функції та об’єктно-орієнтоване програмування в реальному часі, перетворюючи навчання на захопливу творчу діяльність.

Проаналізувавши сучасний стан викладання програмування в закладах загальної середньої та вищої освіти України, ми виявили низку ключових проблем:

– абстрактність та відірваність традиційного навчання програмуванню від реального візуального результату;

– недостатній рівень мотивації учнів до вивчення алгоритмізації та C#;

– низька ефективність засвоєння складних концепцій (цикли, функції, ООП) через відсутність миттєвого зворотного зв’язку;

– потреба в інноваційних педагогічних технологіях, які відповідають вимогам Нової української школи та концепції STEM/STEAM-освіти.

З огляду на ці аспекти, існує нагальна потреба в розробці та впровадженні інтерактивних візуальних методик навчання програмуванню, які б забезпечували глибоке розуміння матеріалу, підвищували мотивацію та формували стійкий інтерес до IT-сфери.

Останнім часом проблема використання Unity та інших візуальних середовищ для навчання програмуванню привернула увагу низки дослідників, а саме:

– Végh L. [23] досліджував використання Unity для створення 2D-ігор як засобу навчання програмування та візуалізації алгоритмічних конструкцій у старшій школі;

– Akcaoglu M., Dogan S., Hodges C. B. [24] розробили та оцінили середньошкільну програму з програмування на C# через Unity 3D, з акцентом на реальні ігрові проєкти для візуалізації концепцій;

– Adeniji I. et al. [25] проаналізували Unity як інструмент для наукової візуалізації алгоритмів у курсових дослідженнях студентів, включаючи 3D-моделі для кращого розуміння C#;

– Roane W. et al. [26] обґрунтували застосування Unity для анімації та інтерактивної візуалізації алгоритмів у вищій освіті, з прикладами на C# для студентів-педагогів;

Ці дослідження демонструють велику увагу науковців до актуальності проблеми використання Unity для візуалізації алгоритмів та концепцій C# у навчанні програмування. Але проблема системного впровадження Unity саме в шкільному курсі інформатики для старшокласників, з акцентом на C# та практичну апробацію, є недостатньо вивченою і потребує більш ґрунтовного дослідження, що і зумовило вибір теми даної кваліфікаційної роботи.

Мета дослідження – науково-теоретичне обґрунтування та експериментальна перевірка ефективності використання Unity для візуалізації алгоритмів та концепцій C# у процесі вивчення основ програмування.

Завдання дослідження:

1. Здійснити аналіз науково-методичної літератури та мережі Інтернет з проблем навчання програмуванню з використанням візуальних середовищ та Unity.

2. Розробити методику використання Unity для візуалізації базових алгоритмічних конструкцій та концепцій C#.

3. Експериментально перевірити ефективність розробленої методики у навчальному процесі.

4. Розробити практичні рекомендації щодо впровадження Unity у викладання програмування в закладах загальної середньої та вищої освіти.

Об'єкт дослідження – процес навчання основам програмування учнів старших класів.

Предмет дослідження – методика візуалізації алгоритмічних конструкцій та концепцій C# за допомогою ігрового рушія Unity.

У процесі виконання кваліфікаційної роботи було використано такі *методи дослідження*: аналіз науково-методичної літератури та джерел мережі Інтернет; педагогічне спостереження; анкетування; педагогічний експеримент; методи математичної статистики (критерій Стьюдента, коефіцієнт Коена).

Наукова новизна дослідження полягає в розробці та експериментальній перевірці методики використання Unity для візуалізації базових алгоритмічних конструкцій (змінні, умови, цикли, функції) та концепцій об'єктно-орієнтованого програмування на C# у шкільному та студентському навчанні.

Практична значущість. Отримані результати та розроблена методика можуть бути використані вчителями інформатики закладів загальної середньої освіти, викладачами ЗВО, а також у системі підвищення кваліфікації педагогічних кадрів для модернізації викладання програмування.

Апробація результатів та публікації. Основні положення кваліфікаційної роботи доповідалися на Всеукраїнських конференціях різного рівня: «Інформаційно-комунікаційні технології як інструмент розвитку критичного мислення під час уроків математики» (Всеукраїнської студентської науково-практичної конференції «Актуальні проблеми

соціально-гуманітарних наук: перспективи та виклики» м. Миколаї, 4 квітня2025 року); «Психологічні виклики та підтримка в освітньому процесі на уроках математики» (Збірник. Фізика, математика, інформатика: досвід та інновації: зб. матеріалів наук.-метод. семінару кафедри методики викладання фізики і математики, Миколаїв, 12 травня 2025 р. Вип. 30. / редкол. : В. Січко (голова), М. Літвінова (голов. ред.), О. Богатєнкова (тех. ред.) Миколаїв : НУК імені адмірала Макарова, 2025. 91 с.).

Структура та обсяг кваліфікаційної роботи. Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. Робота викладена на 92 сторінках тексту, містить 0 таблиць, 13 рисунків. У бібліографії подано 26 джерел, з них 18 зарубіжних.

РОЗДІЛ 1

СТАНОВЛЕННЯ ТА РОЗВИТОК МЕТОДИК ВИКЛАДАННЯ ПРОГРАМУВАННЯ В СУЧАСНОМУ ОСВІТНЬОМУ КОНТЕКСТІ

1.1 Історичний розвиток підходів до навчання програмування

Становлення підходів до навчання програмування відбувалося в тісному зв’язку з розвитком обчислювальної техніки, комп’ютерних наук та потребами індустрії. Кожен історичний етап – від епохи лампових комп’ютерів до ери персональних машин – вніс свої корективи до того, як, для кого та з якою метою викладалося програмування. Розгорнутий огляд історії цього процесу дає змогу краще зрозуміти витоки сучасних підходів, визначити сильні та слабкі сторони минулих практик та сформувати уявлення про майбутнє цієї дисципліни в освітньому середовищі.

1950–1960-ті роки: зародження програмування як освітньої дисципліни.

Період 1950–1960-х років вважається початковим етапом формування викладання програмування. У цей час програмування ще не розглядалося як окрема навчальна дисципліна, а було прикладною частиною наукових досліджень або інженерної роботи. Навчання програмування відбувалося переважно у форматі наставництва в наукових колективах, де кожен новий член поступово опановував роботу з машинною мовою, асемблером або першими високорівневими мовами [2].

Інструментальна база для програмування була надзвичайно обмеженою. Комп’ютери, такі як ENIAC, UNIVAC, IBM 701 або IBM 704, вимагали прямої взаємодії з апаратною частиною. Ранні програмісти працювали з перфокартами, тумблерами та шестигранними кодами. Відповідно, навчання було виключно практичним, індивідуальним і спрямованим на досягнення прикладних цілей.

У 1957 році компанія IBM розробила мову FORTRAN (FORmula TRANslating System), яка стала однією з перших високорівневих мов програмування, призначених для наукових обчислень. Її поява стала проривом, адже тепер не потрібно було писати сотні машинних команд – натомість використовувалися інструкції, ближчі до людської мови. FORTRAN, попри технічну орієнтацію, дозволив ширшому колу дослідників долучатися до програмування [3].

У 1959 році з'явилася мова COBOL (Common Business-Oriented Language), яка була спрямована на вирішення задач бізнесу й адміністрування. Вона запровадила описовий синтаксис, орієнтований на англомовну структуру речень, що полегшувало розуміння коду навіть нефахівцями. Це стало першим кроком до демократизації програмування – його спрощення й поширення серед нетехнічних фахівців[4].

ALGOL (1958) став основою для формування структурного підходу в програмуванні. Його вплив на академічне середовище був значним: ця мова широко використовувалася в університетах для викладання базових алгоритмічних конструкцій. ALGOL також стимулював розвиток систем формального опису мов (наприклад, нотація Бекуса-Наура), що мало фундаментальне значення для компіляторобудування і подальшої стандартизації навчальних курсів[5].

1960-ті роки: перші освітні програми та дидактичні експерименти. У 1960-х роках навчання програмуванню починає набувати ознак окремої педагогічної галузі. Провідні університети світу – MIT, Стенфорд, Карнегі-Меллон – впроваджують спеціалізовані курси, присвячені алгоритмічному мисленню, структурам даних, розробці програмного забезпечення.

Особливу роль у цей період відігравала діяльність Сеймура Пейперта, який у 1967 році створив мову LOGO. Вона була спрямована на дітей молодшого та середнього шкільного віку. Модель LOGO ґрунтувалася на ідеях Жана Піаже – зокрема конструктивізму – і передбачала активне залучення

дитини до створення «власного знання» через експериментування. Команди LOGO дозволяли керувати «черепашкою» на екрані, яка малювала лінії. У процесі побудови малюнків учні засвоювали принципи послідовності, повторення, умовного переходу тощо. LOGO став першим в історії прикладом інтеграції програмування та візуального мислення в навчанні[2].

У цей же період активно обговорювалися питання структури програм. Робота Едсгера Дейкстри та його послідовників популяризувала структурне програмування як противагу «спагетті-коду». Структурне програмування лягло в основу освітніх стандартів у більшості технічних навчальних закладів [6].

Крім LOGO, у навчанні застосовувалися також мови ALGOL W та PL/I, хоча вони були менш популярні через свою складність. У США з'являються перші шкільні комп'ютерні лабораторії, однак їхня кількість була обмеженою, а техніка – дорогою і громіздкою. У Європі популярності набувають радіоаматорські клуби, де підлітки знайомляться з базовим програмуванням.

1970-ті роки: поширення доступних мов та спроби стандартизації освіти. У 1970-х роках у світі починає активно формуватись середовище, сприятливе для масового викладання програмування. Однією з ключових подій стало створення мови Pascal у 1970 році Ніклаусом Віртом. Мова була спеціально розроблена як дидактичний інструмент для викладання структурованого програмування. Її чітка структура, строгі правила типізації та можливість модульної побудови програм ідеально відповідали вимогам навчального середовища.

Водночас велике значення мала поява мови BASIC (1964), що набула особливої популярності саме в 1970-х – завдяки поширенню мікрокомп'ютерів, таких як Altair 8800. BASIC став фактично «мовою першого контакту» для тисяч учнів і студентів. Завдяки простому синтаксису та доступності літератури навчання програмуванню стало масовим явищем у багатьох розвинених країнах. Підручники, журнали, телевізійні програми –

усе це сприяло популяризації програмування як хобі та майбутньої професії [8].

У 1972 році створюється мова С, яка, попри свою складність, стала основою для системного програмування. У навчальному середовищі її використовували переважно у технічних університетах. Проте саме на цей період припадає початок усвідомлення важливості формування алгоритмічного мислення навіть у непрофільних фахівців.

Окремої уваги заслуговують перші спроби стандартизації шкільного курсу інформатики. У США та Великобританії розпочинаються експерименти зі створення навчальних планів, які включають базове знайомство з алгоритмами, логікою, та основами програмування. Це сприяло поступовому переходу від факультативного до системного навчання в школі.

1980–1990-ті роки: інституціоналізація та демократизація програмування в освіті. Період 1980–1990-х років знаменував собою важливий етап у розвитку підходів до навчання програмування, що характеризувався інтенсивною інституціоналізацією цієї освітньої галузі, а також дедалі глибшим проникненням інформаційних технологій у загальноосвітній простір. Саме в цей час відбувається системне переосмислення ролі програмування в навчальних планах різного рівня – від початкової школи до вищої технічної освіти.

Поширення персональних комп’ютерів, зокрема таких моделей, як Commodore 64, Apple II, ZX Spectrum, IBM PC, сприяло істотному розширенню доступу до засобів обчислювальної техніки. Це, у свою чергу, створило сприятливе підґрунтя для масового впровадження комп’ютерних технологій у шкільну й позашкільну освіту. І хоча технічна база залишалась нерівномірною між країнами та регіонами, загальна тенденція до популяризації обчислювальної грамотності була незворотною.

У 1980-х роках у багатьох країнах Європи й Північної Америки активно розробляються державні стратегії інформатизації освіти, в межах яких

програмування визнається не лише прикладною навичкою, а й інструментом розвитку логіко-алгоритмічного мислення. У Великій Британії, наприклад, державна ініціатива «Micros in Schools» передбачала постачання комп'ютерів BBC Micro до освітніх закладів та супроводжувалася створенням спеціальних освітніх телепрограм, підручників і дидактичних матеріалів. Аналогічні процеси мали місце в США, Канаді, Франції, Німеччині та Японії.[9]

На дидактичному рівні спостерігається поглиблення інтеграції між методиками викладання програмування та педагогічною психологією. Теорії Жана Піаже, Джерома Брунера, Сеймура Пейперта набувають популярності серед розробників освітніх курсів, оскільки підкреслюють роль активного, конструктивного навчання, яке передбачає творчу діяльність учня у процесі формування алгоритмічного досвіду. Ідеї Пейперта, зокрема його концепція «математичного мислення через програмування», отримують подальший розвиток у проектах на основі мови LOGO, включаючи створення середовищ TurtleArt та LEGO Mindstorms.

У вищій освіті програмування поступово стає фундаментальною частиною навчальних планів не лише інженерних і математичних спеціальностей, але й таких галузей, як економіка, біологія, соціологія. Відбувається активне впровадження модульного принципу викладання, при якому навчальні курси розділяються на окремі теми: алгоритми, структури даних, об'єктно-орієнтоване програмування, бази даних, теорія обчислень тощо.

Окремо варто згадати про стрімке поширення мови програмування C++ (вперше представлена у 1983 році), яка відкрила новий етап в еволюції викладання програмування, пов'язаний із переходом до об'єктно-орієнтованої парадигми. Цей перехід вимагав адаптації як дидактичних підходів, так і зміни освітніх цілей – від суто алгоритмічного мислення до розуміння концепцій інкапсуляції, спадкування, поліморфізму.

У 1990-х роках дедалі більшого значення набуває поняття «комп’ютерної грамотності» в широкому сенсі, яке включає як базові навички користування програмним забезпеченням, так і елементи логічного, структурного та креативного мислення, притаманні програмістській діяльності. У цей період зростає інтерес до середовищ візуального програмування, таких як HyperCard, Visual Basic, а пізніше – Delphi, що, з одного боку, спрощувало вхід у програмування для новачків, а з іншого – викликало дискусії щодо глибини засвоєних знань та потреби збереження «чистої» алгоритмічної бази[10].

Також варто зазначити вплив глобалізації освітнього процесу, що проявлявся через обмін досвідом між країнами, переклад та адаптацію найкращих освітніх практик, а також розвиток міжнародних стандартів, зокрема ACM/IEEE Computing Curricula (1991), які встановлювали рамки для формування базових навчальних програм у галузі комп’ютерних наук і програмування.

Таким чином, 1980–1990-ті роки заклали основи для сучасної інформатичної освіти, в якій програмування поступово перетворюється з елітарної технічної навички на універсальний інструмент пізнання, вираження та професійної самореалізації в епоху цифрової трансформації.

2000–2020-ті роки: інституціоналізація цифрової освіти та глобалізація підходів до навчання програмування. На початку XXI століття програмування остаточно утверджується як ключовий компонент не лише вищої технічної освіти, а й шкільної, позашкільної та неформальної освітньої сфери. Це десятиліття позначене стрімкою експансією інформаційних технологій в усі сфери життя – від науки та промисловості до культури, побуту й освіти. На тлі цифрової трансформації змінюється сама філософія викладання програмування: від акценту на синтаксис і мови – до орієнтації на розвиток алгоритмічного, обчислювального та критичного мислення [11].

Початок 2000-х років характеризується активним поширенням персональних комп'ютерів та інтернету в навчальних закладах, що дозволяє впроваджувати нові форми взаємодії – зокрема електронне навчання (e-learning), дистанційні курси та віртуальні середовища для програмування. З'являються інтерактивні платформи на кшталт Codecademy, Udacity, Coursera та Khan Academy, які надають відкритий доступ до навчальних матеріалів з інформатики та програмування незалежно від місця проживання та соціального статусу учнів. Це знаменує перехід до епохи масового відкритого онлайн-навчання (Massive Open Online Courses, MOOC), що розмиває традиційні освітні кордони.

У середній освіті одним із найбільш резонансних кроків стало створення в 2006 році візуального середовища Scratch Массачусетським технологічним інститутом. Scratch, орієнтований на дітей 8–16 років, втілив ідеї конструктивістського навчання через творчість, гру та інтерактивне моделювання. Блочна система програмування дозволила виключити синтаксичні помилки, зосередившись на логіці й структурі алгоритмів. Scratch поступово став світовим стандартом початкового навчання програмуванню й відкрив двері до цифрової грамотності для мільйонів учнів [12].

Паралельно з цим, міжнародні освітні ініціативи, зокрема *Computer Science for All* у США (2016), *Informatik in der Schule* у Німеччині та *Цифрова школа* в Україні, стимулювали інтеграцію програмування до базових шкільних програм. В Європі та Північній Америці розпочалася системна підготовка вчителів нової генерації, здатних не лише викладати синтаксис мов, але й модерувати дослідницьку, проєктну та міждисциплінарну діяльність із використанням засобів програмування. Такий підхід наблизив викладання інформатики до гуманістичної освітньої парадигми, де комп'ютер виступає не лише інструментом, а середовищем розвитку особистості [11].

У вищій освіті відбувається поступова переорієнтація від традиційних курсів «Програмування на C++» до міждисциплінарних модулів з аналізу

даних, штучного інтелекту, хмарних обчислень, інженерії програмного забезпечення. Впроваджуються принципи STEM- та STEAM-освіти, які інтегрують інформатику в ширший контекст технічних, природничих, мистецьких і гуманітарних дисциплін. Значного поширення набувають хакатони, дослідницькі марафони, стартап-школи, в яких програмування розглядається не лише як навичка, а як форма творчої й підприємницької активності [11].

Також у 2010-х роках помітним стає вплив методології *computational thinking* (обчислювального мислення), популяризованої Джінетт Вінг. Цей підхід базується на ідеї, що програмування – це лише один із проявів глибшого когнітивного процесу, пов’язаного з декомпозицією задач, пошуком шаблонів, абстрагуванням і формуванням алгоритмів. У багатьох країнах computational thinking інтегрується в освітні стандарти як базовий компонент цифрової компетентності, що має бути сформований у всіх учнів незалежно від їхніх майбутніх професій.

На межі 2020-х років спостерігається інтенсифікація використання адаптивного навчання, машинного оцінювання та інтелектуальних систем підтримки навчального процесу. З’являються середовища, що динамічно підлаштовуються під індивідуальні стилі навчання учнів, а також навчальні ігри та симуляції, які забезпечують глибше залучення й мотивацію до навчання програмування. Водночас триває дискусія щодо балансу між теоретичним фундаментом і прикладними навичками, між глибиною знань і поверхневою компетенцією, характерною для «кліпового» споживання інформації в цифрову епоху.

2020-ті роки та сучасність: цифрова революція, штучний інтелект та зміни в освіті програмування. У 2020-х роках ми спостерігаємо глибокі трансформації в освіті програмування, що зумовлені не лише технологічними досягненнями, а й глобальними соціальними та економічними змінами. Коронавірусна пандемія 2020 року стала каталістом для значного переходу до

дистанційного навчання та використання онлайн-платформ для навчання програмуванню. Освітні платформи, як-от Codecademy, Coursera, edX, а також безліч нових стартапів, почали інтенсивно впроваджувати адаптивні технології, які забезпечують персоналізований підхід до кожного студента. Водночас, зміст програмування в навчальних курсах став набагато ширшим, включаючи новітні галузі, як от штучний інтелект, машинне навчання, блокчейн і робототехніка.

Штучний інтелект (AI) та програмування. З розвитком штучного інтелекту (AI) та машинного навчання, програмування переживає революцію. AI системи, як-от OpenAI, дають змогу автоматизувати створення програмного коду, генеруючи рішення на основі текстових запитів, що змінює саме поняття «програмування». Завдяки таким інструментам, як ChatGPT, Codex, GPT-4 та інші, з’являються нові формати навчання, де Здобувачі освіти взаємодіють з інтелектуальними системами, отримуючи миттєві підказки, виправлення та рекомендації щодо коду. Це дозволяє знижувати бар’єри для новачків і робити процес навчання більш інтуїтивним, водночас поглиблюючи розуміння алгоритмів і принципів програмування через взаємодію з моделями AI.

Крім того, деякі програми використовують штучний інтелект для автоматичного виправлення кодів, що значно полегшує процес навчання. Наприклад, інтерактивні платформи типу Replit або GitHub Copilot використовують AI для написання коду, що дозволяє швидко адаптуватися до нових мов і технологій програмування, а також допомагає у розвитку особистих проєктів.

Но-код (No-code) та низькокод (Low-code) платформи. Ще одним значним трендом останніх років є бум но-код (No-code) і лоу-код (Low-code) платформ. Ці платформи дозволяють створювати програми без необхідності писати традиційний код, замінюючи його графічними інтерфейсами, шаблонами й автоматичними процесами. Але навіть за умов відсутності

прямого кодування, розробники і користувачі повинні володіти певними знаннями в області логіки, алгоритмів та цифрових технологій для ефективного використання цих інструментів [13].

Платформи на кшталт Bubble, Webflow, Adalo, а також Microsoft Power Apps, дозволяють створювати складні веб-сайти та мобільні додатки, не маючи глибоких знань у програмуванні. Вони розширюють можливості для тих, хто хоче розвивати цифрові продукти без необхідності вивчати складні мовні конструкції, але разом з тим стимулюють нові форми освіти, які зосереджуються на логічному мисленні, розумінні технологій та оптимізації процесів.

Це особливо важливо для майбутніх підприємців та дизайнерів, які хочуть створювати інструменти і сервіси, не зупиняючись на технічних складнощах програмування. У цьому контексті програмування перестає бути лише «класичною» дисципліною, а стає універсальною навичкою для творчих людей, що хочуть втілювати ідеї в цифровому світі.

Відкриті технології, платформи для колективного навчання та інклюзивність**.** Протягом 2020-х років спостерігається стійка тенденція до відкритих технологій та платформи для колективного навчання. Open-source проєкти, форуми, спільноти на GitHub, Stack Overflow, Reddit, форуми програмістів продовжують домінувати в світі програмування, сприяючи розвитку нових технологій і полегшуючи доступ до знань.

Розвиток інклюзивних ініціатив у програмуванні також стає однією з головних тенденцій. Такі організації, як Women Who Code, Black Girls Code, Code2040, Code Club, надають можливості для навчання програмуванню для тих, хто зазвичай не має рівного доступу до технологічної освіти – жінок, меншин, дітей з сільських чи малозабезпечених регіонів. Ці програми не лише забезпечують технічні навички, але й створюють простір для розвитку менторства та співпраці серед учасників.

Вплив соціальних медіа та відео-контенту. Програмування також ставить акцент на відео-контент і соціальні медіа. Інтерактивні курси та відео-уроки на YouTube, TikTok, Instagram, а також стрімінгові платформи на кшталт Twitch стали популярними каналами для навчання і взаємодії з програмістами та експертами. Такі ресурси, як The Coding Train, freeCodeCamp, Traversy Media, роблять програмування доступним і зрозумілим для новачків завдяки простим і наочним відео.

Це відкриває нові можливості для глобального навчання, де учні можуть взаємодіяти з реальними професіоналами, брати участь у хакатонах та отримувати зворотній зв'язок у реальному часі.

Міждисциплінарність та роль програмування в галузях гуманітарних та соціальних наук**.** Останніми роками розвивається тенденція поєднання програмування з іншими галузями знань, зокрема з гуманітарними і соціальними науками. Університети почали розробляти курси, що поєднують програмування з психологією, соціологією, філософією, економікою та навіть мистецтвом. Така міждисциплінарна освіта має на меті підготувати фахівців, здатних вирішувати комплексні соціальні та культурні проблеми через інноваційні цифрові рішення, що включають прогресивні технології, такі як штучний інтелект, великі дані, інтернет речей (IoT), блокчейн тощо.

Таким чином, розвиток освіти програмування в 2020-х роках можна охарактеризувати як етап, на якому візуальне навчання, штучний інтелект, нові інструменти й платформи стирають межі між програмістами та іншими професіями, дозволяючи людям з різними навичками створювати та впроваджувати технології в різних сферах життя. Це свідчить про те, що програмування стає не просто технічною навичкою, а важливою складовою комплексного розвитку людини в умовах цифрового століття.

1.2 Порівняння традиційних та інтерактивних методів навчання

Навчання програмуванню є важливою частиною сучасної освіти, і вибір методів навчання значною мірою впливає на ефективність навчального процесу. У традиційних методах навчання програмування часто використовуються лекції, підручники, інструкції та вправи, що виконуються в класі. З іншого боку, інтерактивні методи, до яких можна віднести використання спеціалізованих програмних середовищ, таких як Unity, орієнтуються на більш активну участь студентів у навчанні через створення та тестування програм у реальному часі. Порівняння цих підходів дозволяє виявити їх сильні й слабкі сторони, а також краще зрозуміти, як їх можна поєднувати для досягнення максимального навчального ефекту [14].

Традиційні методи навчання програмуванню. Традиційний підхід до навчання програмуванню ґрунтується на класичних формах передачі знань, де основними інструментами є лекції, підручники та лабораторні роботи. Лекційні заняття зазвичай орієнтовані на теоретичну частину матеріалу: ознайомлення з мовами програмування, основними алгоритмами та принципами програмування. Зміст лекцій може охоплювати основи синтаксису мов програмування, структури даних, бази алгоритмів та принципи їх оптимізації. Практичні заняття, що супроводжують лекції, сприяють закріпленню цих теоретичних знань через написання невеликих програмних задач.

У традиційній системі навчання велика увага приділяється чітким теоретичним основам, а також структурованим завданням, які сприяють розумінню основних концепцій програмування. Однак цей підхід має обмеження, які стають очевидними при намаганні навчити студентів складніших концепцій. Одним з основних викликів традиційного підходу є недостатня увага до інтерактивності й реального застосування знань. Часто Здобувачі освіти не мають можливості працювати з реальними проєктами або ситуаціями, які б допомогли їм розвинути навички критичного мислення та

творчого вирішення задач. Крім того, значна частина студентів може втратити інтерес до навчання через відсутність практичної складової та мотивації.

Інтерактивні методи навчання програмуванню. Інтерактивні методи навчання програмуванню, навпаки, ставлять у центр навчального процесу активну участь студентів у створенні реальних програм. Цей підхід охоплює такі інструменти, як програмне середовище для розробки ігор, візуальні мови програмування, інтерактивні онлайн-платформи для навчання, а також використання сучасних інструментів на зразок Unity. Основною характеристикою інтерактивного підходу є можливість швидко побачити результат своєї роботи, тестуючи і вдосконалюючи програми в реальному часі [15].

Використання Unity, зокрема, є потужним інструментом для навчання програмуванню, оскільки дозволяє студентам працювати з реальними проєктами, розробляючи ігри, симуляції або візуалізації. Здобувачі освіти можуть швидко отримати зворотний зв'язок про результати своєї роботи через безпосереднє тестування своїх програм у вигляді інтерактивних проєктів, що сприяє їх більш глибокому розумінню основ програмування [16].

Одним з головних переваг інтерактивного підходу є розвиток навичок критичного мислення і творчого вирішення задач. Здобувачі освіти можуть не лише вивчати теоретичні аспекти програмування, але й на практиці застосовувати свої знання для розв'язання реальних проблем. Використання інструментів на зразок Unity дозволяє їм працювати з 3D-графікою, фізичними моделями, а також вивчати принципи проектування ігор, що допомагає розвивати як технічні, так і дизайнерські навички [16].

Інтерактивні методи також пропонують можливість працювати в команді, що сприяє розвитку комунікаційних і організаційних навичок. Робота в групах над розробкою проєктів дозволяє студентам розвивати здатність працювати в умовах реального проєктного середовища, де важлива ефективна комунікація та координація дій. Крім того, ці методи дозволяють враховувати

індивідуальні потреби студентів, надаючи можливість працювати на своєму рівні та розвиватися в темпі, який відповідає їхнім можливостям [15].

*Порівняння ефективності традиційних і інтерактивних методів***.** Незважаючи на всі переваги інтерактивних методів, традиційні підходи не втратили своєї актуальності. Існують ситуації, коли класичне навчання надає вищий рівень теоретичної підготовки, необхідної для розуміння складних алгоритмів або математичних концепцій, які лежать в основі програмування. У таких випадках традиційні методи можуть бути більш ефективними для формування основних знань, які згодом можуть бути застосовані в інтерактивному середовищі.

Інтерактивні методи, в свою чергу, забезпечують більш гнучкий та адаптивний процес навчання. Вони дозволяють студентам відразу ж застосовувати теоретичні знання на практиці, що сприяє кращому засвоєнню матеріалу. Однак для деяких студентів інтерактивні методи можуть бути занадто складними або недостатньо структурованими, що може призвести до труднощів у навчанні. Тут традиційні методи можуть компенсувати відсутність базових знань.

Загалом, можна сказати, що обидва підходи мають свої сильні та слабкі сторони. Традиційні методи є основою для формування теоретичних знань, тоді як інтерактивні методи забезпечують більш динамічне та мотивуюче навчання, яке включає практичне застосування знань.

Використання Unity в контексті інтерактивного навчання. Unity є однією з найбільш популярних платформ для розробки інтерактивних 2D та 3D-ігор, що робить її надзвичайно корисною для навчання програмуванню. Вона надає потужні можливості для створення навчальних проєктів, що дозволяють студентам розвивати навички в багатьох аспектах програмування та дизайну. Використання Unity в освітніх цілях надає студентам безпосередній досвід роботи з реальними програмами та проектами, що дозволяє застосовувати теоретичні знання на практиці [16].

Одним із головних аспектів, які виділяють Unity серед інших середовищ для навчання, є її багатофункціональність і здатність працювати з великим набором інструментів і технологій, що використовуються у реальних проєктах. Unity дозволяє створювати ігри з високим рівнем інтерактивності та візуалізації, що дозволяє студентам бачити результат своєї роботи практично відразу. Це дає можливість отримати миттєвий зворотний зв'язок, що є важливим для швидкого коригування помилок і покращення навичок програмування [16].

Для навчання програмуванню Unity має кілька ключових переваг:

1. ***Інтуїтивно зрозумілий інтерфейс***: Unity надає інтерфейс, який легко зрозуміти навіть новачкам. Здобувачі освіти можуть швидко освоїти основи роботи з 3D-сценами, матеріалами, анімаціями та фізичними властивостями об'єктів. Завдяки використанню C# як основної мови програмування, Здобувачі освіти можуть легко перейти від основ програмування до більш складних концепцій.

2. **Підтримка великої кількості платформ**: Unity дозволяє створювати проєкти, які можуть бути запущені на різних платформах, таких як ПК, мобільні пристрої, VR/AR, консолі та навіть в браузерах. Це дає студентам можливість розуміти, як розробка ігор або програмних продуктів може бути масштабована на різні платформи.

3. **Можливості для колективної роботи**: Unity підтримує роботу в команді, що дозволяє студентам працювати разом над одним проєктом. Це дає можливість розвивати навички командної роботи, комунікації та управління проєктами, що є важливими аспектами професійної діяльності в індустрії програмування.

4. **Великі можливості для розширення і кастомізації**: Завдяки великій кількості доступних бібліотек, плагінів та розширень, Unity дозволяє адаптувати проєкти під конкретні потреби. Здобувачі освіти можуть

експериментувати з різними технологіями, такими як AI, фізика, анімація та графіка, що розширює їхні можливості у вивченні програмування.

Інтерактивні методи навчання через практичні проєкти. Інтерактивні методи навчання мають ще одну важливу перевагу – це можливість створення практичних проєктів, які відповідають реальним вимогам та стандартам індустрії. За допомогою Unity Здобувачі освіти можуть реалізовувати свої ідеї через розробку проєктів, що дозволяє їм не лише вивчати теорію програмування, але й отримувати практичний досвід. Наприклад, Здобувачі освіти можуть створювати прості ігри, симуляції або візуалізації, що дозволяє їм освоювати принципи роботи з графікою, анімацією, фізикою, штучним інтелектом та іншими важливими аспектами програмування [15].

Розробка практичних проєктів дозволяє студентам створювати речі, якими вони можуть пишатися, і, що важливо, отримувати зворотний зв'язок від своїх викладачів і однокурсників. Це значно підвищує мотивацію, оскільки учні можуть спостерігати, як їхні ідеї перетворюються на реальний продукт.

В Unity є також можливість створювати проєкти за допомогою концепції «гейміфікації», що надає змогу студентам вчитися через гру. Це може включати розробку ігор з різними рівнями складності, що дозволяє не лише вивчати програмування, а й розвивати навички вирішення проблем і творчого підходу. Такі проєкти можуть включати використання різних механік і стилів ігор, що допомагає студентам краще зрозуміти, як працює програмування в реальному світі.

Переваги інтерактивного навчання в контексті сучасних вимог до освіти. Сучасна освіта орієнтована на розвиток таких навичок, як критичне мислення, творчість, вирішення складних задач і ефективна комунікація. Інтерактивні методи навчання, особливо через використання інструментів на кшталт Unity, дозволяють студентам працювати не лише з теорією, але й з реальними практичними ситуаціями. Це дозволяє їм розвивати навички, які є

важливими не лише в академічному контексті, а й у майбутній професійній діяльності [15].

Програми, створені за допомогою Unity, дозволяють студентам вивчати програмування в контексті сучасних технологій, таких як розробка ігор, віртуальна реальність та штучний інтелект. Це дає їм конкурентні переваги на ринку праці, оскільки такі навички є затребуваними в багатьох технологічних компаніях [16].

1.3 Переваги та обмеження різних педагогічних підходів до викладання програмування

Викладання програмування є складним і багатогранним процесом, який залежить від багатьох чинників, таких як навчальна програма, доступні ресурси, інтереси студентів і педагогічний підхід. Вибір конкретного підходу до викладання програмування має велике значення, оскільки різні методи навчання можуть мати значний вплив на ефективність засвоєння матеріалу, мотивацію студентів та загальний процес навчання.

Існує кілька основних педагогічних підходів до викладання програмування, кожен з яких має свої переваги та обмеження. Розглянемо їх детальніше, щоб зрозуміти, який з підходів найбільш ефективний для досягнення різних навчальних цілей.

Традиційний підхід. Традиційний підхід до викладання програмування в основному передбачає теоретичні лекції та лабораторні заняття, під час яких Здобувачі освіти навчаються основам програмування через подання матеріалу в класичному вигляді, як-от на прикладах коду та алгоритмів. Цей підхід часто використовує такі методи, як фронтальне викладання, контроль за виконанням завдань, пояснення на прикладах.

Переваги:

1. **Глибоке розуміння основ**: Традиційний підхід дає змогу студентам ретельно засвоїти теоретичні основи програмування, такі як алгоритми, структури даних і принципи роботи комп'ютерних систем.
2. **Можливість використання структурованих матеріалів**: Лекції та семінари дозволяють чітко структурувати матеріал, систематизувати знання й надати чітку теоретичну базу.
3. **Контроль за процесом навчання**: Викладач може детально відслідковувати процес засвоєння матеріалу, що дозволяє коригувати методи навчання та надавати індивідуальні поради студентам.

Обмеження:

1. **Недостатня мотивація**: Традиційне викладання може бути не завжди захоплюючим для студентів, оскільки часто зосереджується на абстрактних поняттях, а не на практичних завданнях. Це може призвести до зниження мотивації й інтересу до предмету.

2. **Відсутність реальних практичних проєктів**: Традиційні методи навчання часто обмежуються теоретичними задачами, що не дає студентам можливості працювати з реальними проєктами та проблемами індустрії. Це може завадити розвитку практичних навичок.

3. **Малий рівень інтерактивності**: У традиційному підході Здобувачі освіти часто є пасивними слухачами, і їхня активність обмежується виконанням контрольних робіт та домашніх завдань. Це знижує рівень взаємодії і співпраці серед студентів.

Проєктно-орієнтований підхід. Проєктно-орієнтований підхід передбачає, що Здобувачі освіти працюють над реальними проєктами або завданнями, які мають прикладне значення. Такий підхід фокусується на застосуванні знань програмування до створення реальних продуктів, що дозволяє студентам не лише вивчати теорію, але й здобувати практичний досвід [18].

Переваги:

1. **Розвиток практичних навичок**: Здобувачі освіти мають можливість працювати з реальними інструментами та технологіями, що дозволяє їм отримати досвід роботи в індустрії і значно покращити свої навички.

2. **Мотивація через реальні результати**: Завдяки роботі над реальними проєктами Здобувачі освіти можуть бачити конкретні результати своєї праці, що підвищує їхню мотивацію й інтерес до навчання.

3. **Розвиток командної роботи**: Проєктно-орієнтовані методи часто включають роботу в групах, що дозволяє студентам вчитися співпрацювати, обговорювати ідеї, обмінюватися досвідом і разом вирішувати проблеми.

Обмеження:

1. **Потреба в значних ресурсах**: Проєктно-орієнтоване навчання може вимагати великих витрат часу та ресурсів, оскільки для реалізації реальних проєктів необхідні комп'ютерне обладнання, програмне забезпечення та кваліфіковані викладачі.

2. **Не завжди систематизоване навчання**: У проєктно-орієнтованому підході іноді відсутня чітка структура навчання, що може призвести до прогалин у знаннях, оскільки Здобувачі освіти можуть не отримати достатньо теоретичних основ для глибокого розуміння матеріалу.

3. **Ризик нерівномірного навантаження**: У групових проєктах можуть виникати проблеми з розподілом обов'язків між студентами, і деякі учасники можуть не бути достатньо залучені в процес, що впливає на результат.

Ігрові методи навчання (гейміфікація). Ігрові методи навчання, або гейміфікація, використовують елементи ігор для створення навчальних середовищ, що стимулюють студентів до активної участі. Використання ігор для навчання програмуванню стало популярним через здатність таких методів привертати увагу та мотивувати студентів [19].

Переваги:

1. **Мотивація через досягнення**: Гейміфікація дозволяє здобувачам освіти отримувати відчуття досягнення через виконання завдань і отримання винагород, що підвищує інтерес і мотивацію.

2. **Залучення та активна участь**: Ігрові елементи, такі як змагання, бали, рівні складності, можуть залучити студентів, зробивши навчання більш захоплюючим і менш стресовим.

3. **Розвиток проблемно-орієнтованого мислення**: Гейміфікація сприяє розвитку критичного мислення та навичок вирішення проблем через інтерактивні завдання та ситуації, які потребують застосування теоретичних знань на практиці.

Обмеження:

1. **Можлива поверховість**: Ігрові методи можуть зосереджуватися на поверхневому вивченні концепцій без достатнього поглиблення в теорію, що може призвести до поверхневого розуміння матеріалу.

2. **Залежність від технологій**: Гейміфікація вимагає значних технологічних ресурсів для створення ігор, а також потребує високої доступності інтернету та програмного забезпечення, що може бути обмеженням для деяких навчальних закладів.

3. **Не завжди підходить для всіх здобувачів освіти**: Хоча ігрові методи можуть бути дуже ефективними для деяких студентів, для інших вони можуть здаватися занадто легковажними або відволікати від основного навчального процесу.

Висновки до розділу 1

У першому розділі було здійснено науково-теоретичне обґрунтування особливостей навчання основам програмування з використанням візуальних та інтерактивних методів, зокрема ігрового рушія Unity.

Проведено історичний огляд розвитку методик викладання програмування: від машинних кодів та перших високорівневих мов 1950–1960-х років (FORTRAN, COBOL, ALGOL, LOGO) до появи дидактичних мов (Pascal, BASIC), поширення об’єктно-орієнтованого підходу (C++, Java) та сучасних візуальних і блочних середовищ (Scratch, Unity, Godot). Показано, що кожен етап розвитку обчислювальної техніки та педагогічної думки супроводжувався пошуком ефективніших способів формування алгоритмічного та обчислювального мислення.

Розкрито суттєві недоліки традиційного підходу до навчання програмуванню: висока абстрактність матеріалу, недостатній зворотний зв’язок, низька внутрішня мотивація учнів, слабке перенесення теоретичних знань у практичну діяльність. Натомість інтерактивні та ігрові методи, зокрема з використанням Unity, забезпечують миттєву візуалізацію результатів коду, емоційне залучення та можливість працювати над реальними проєктами.

Охарактеризовано основні педагогічні підходи до викладання програмування: традиційний (лекційно-практичний), проєктно-орієнтований та ігровий (гейміфікація). Показано їхні переваги та обмеження. Особливу увагу приділено інтерактивним методам, які поєднують теоретичну підготовку з практичним застосуванням, сприяють розвитку критичного мислення, командної роботи та творчого підходу до розв’язання задач.

Детально проаналізовано можливості Unity як дидактичного інструменту: інтуїтивно зрозумілий інтерфейс, підтримка мови C#, потужні засоби візуалізації та анімації, можливість швидкого прототипування, мультиплатформність, наявність великої спільноти та навчальних ресурсів. Доведено, що Unity дозволяє ефективно візуалізувати базові алгоритмічні конструкції (змінні, умови, цикли, функції) та концепції об’єктно-орієнтованого програмування (класи, об’єкти, спадкування, інкапсуляція, поліморфізм).

Таким чином, впровадження Unity у процес навчання програмуванню можна розглядати як дієвий спосіб подолання абстрактності дисципліни, підвищення мотивації учнів, формування стійких практичних навичок та підготовки до реальної розробки програмного забезпечення.

РОЗДІЛ 2
ПРАКТИЧНА РЕАЛІЗАЦІЯ КОНЦЕПЦІЙ ПРОГРАМУВАННЯ В UNITY

2.1 Розробка базового проєкту для візуалізації алгоритмів

Unity – це потужний рушій для створення ігор та інтерактивних застосунків, розроблений компанією Unity Technologies. Він широко використовується не тільки в ігровій індустрії, але й у освітніх цілях, оскільки дозволяє візуалізувати абстрактні концепції програмування через графічні об'єкти, анімації та взаємодію в реальному часі. Для вивчення основ програмування на C# Unity є ідеальним інструментом, особливо для початківців, включаючи дітей шкільного віку, оскільки поєднує кодування з візуальними елементами, роблячи процес навчання подібним до гри. Згідно з офіційними ресурсами Unity, платформа підтримує скриптинг на C#, що робить її доступною для освоєння базових концепцій, таких як змінні, об'єкти та прості взаємодії [8].

У цьому підрозділі ми розглянемо базові кроки зі створення простого проєкту в Unity, знайомство з інтерфейсом, створення об'єктів та написання перших скриптів. Фокус буде на фундаментальних елементах, які дозволять новачкам зрозуміти, як код впливає на віртуальний світ. Ми уникатимемо складних конструкцій, таких як цикли чи умови, щоб не дублювати матеріал наступних підрозділів. Замість цього, акцент на змінних та базовому спавні (створенні) об'єктів.

Знайомство з Unity: встановлення та перші кроки. Перед початком роботи необхідно встановити Unity. Це безкоштовно для освітніх цілей, і процес простий, що робить його доступним навіть для Здобувачі освіти в під наглядом дорослих. Ось покрокова інструкція:

1. **Завантаження Unity Hub**: Перейдіть на офіційний сайт unity.com і завантажте Unity Hub – це менеджер проєктів, який дозволяє керувати версіями рушія. Unity Hub сумісний з Windows, macOS та Linux. Для дітей рекомендується використовувати просту інструкцію з картинками, наприклад, «Натисни 'Download' і запусти файл, як гру».

2. **Встановлення рушія Unity**: У Unity Hub оберіть версію (рекомендовано 2022 LTS або новішу для стабільності). Додайте модулі для платформи (наприклад, Windows Build Support). Процес займає 10-30 хвилин залежно від інтернету.

3. **Створення першого проєкту**: У Hub натисніть «New Project», оберіть шаблон «3D Core» (для базового 3D-середовища) і вкажіть ім'я проєкту, наприклад, «MyFirstAlgoViz». Натисніть «Create Project» – і ви потрапите в Unity Editor.

Для дітей цей етап можна перетворити на гру: «Уяви, що ти створюєш свій віртуальний світ, як у Minecraft, але з кодом!» Це допомагає підтримувати інтерес і знижує бар'єр входу. Unity підтримує освітні ліцензії, де можна використовувати готові навчальні матеріали з Unity Learn, адаптовані для Здобувачі освіти в [8].

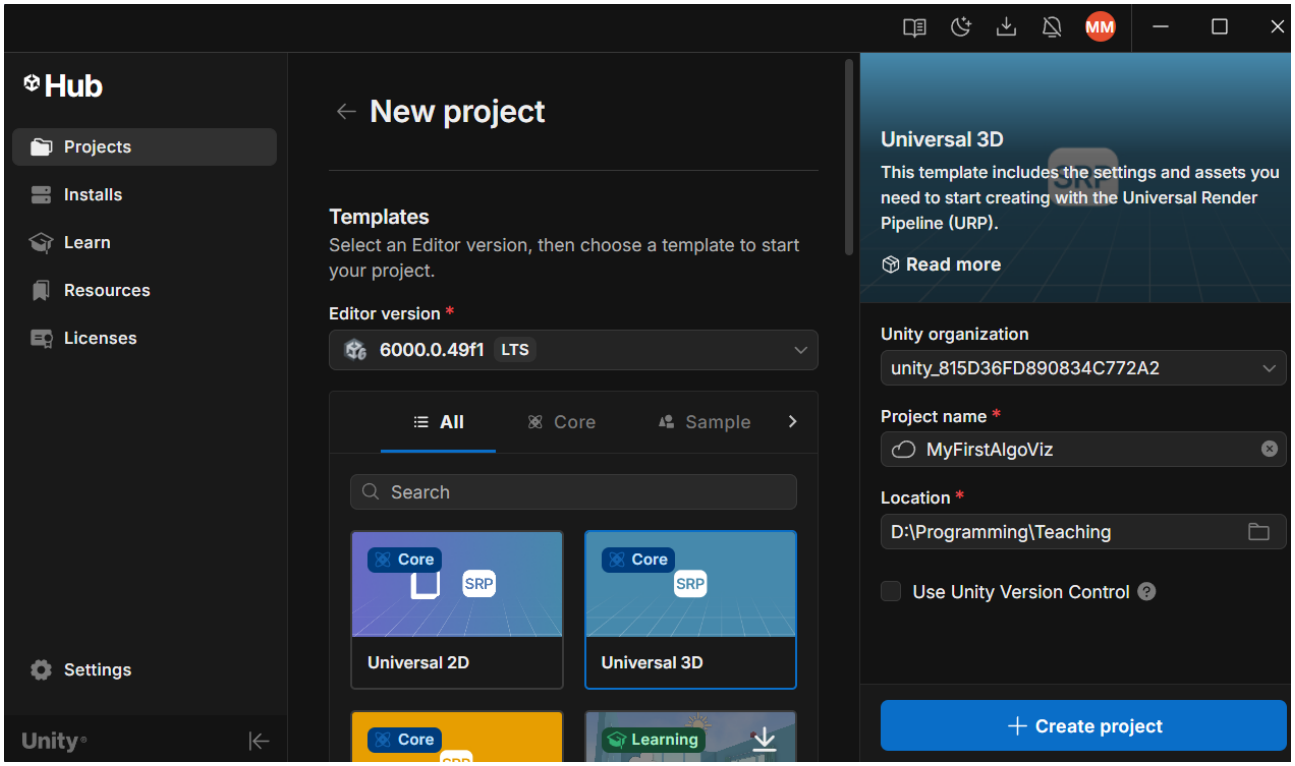


Рис. 2.1: Скріншот інтерфейсу Unity Hub під час створення проєкту. Джерело: офіційний сайт Unity (unity.com/hub).

Огляд інтерфейсу Unity Editor

Unity Editor – це основне середовище для розробки. Воно складається з кількох панелей, які легко зрозуміти навіть новачкам:

- **Scene View**: Центральна панель, де ви бачите віртуальний світ. Тут можна переміщувати об'єкти мишкою, як у 3D-редакторі.
- **Hierarchy**: Список всіх об'єктів у сцені (наприклад, камера, світло). Це як «інвентар» вашого світу.
- **Inspector**: Панель властивостей обраного об'єкта. Тут змінюються параметри, як колір чи розмір.
- **Project**: Браузер файлів проєкту (скрипти, моделі).
- **Console**: Вивід повідомлень і помилок від скриптів.
- **Game View**: Симуляція гри – натисніть Play, щоб побачити, як працює проєкт.

Для початківців рекомендується почати з порожньої сцени: File > New Scene. Додайте базові елементи – Main Camera (для перегляду) і Directional

Light (для освітлення). Це створює «порожній світ», де можна експериментувати.

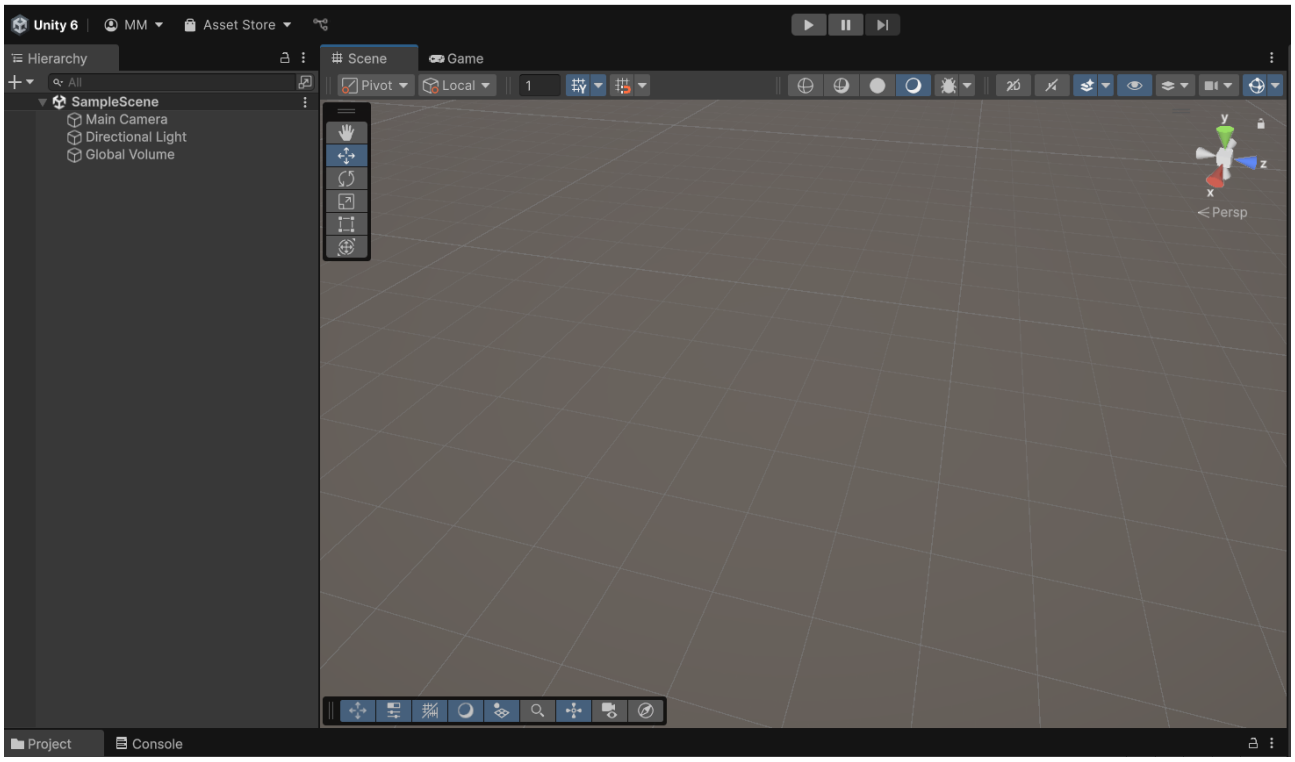


Рис. 2.2: Основний інтерфейс Unity Editor з позначеними панелями

Щоб адаптувати для дітей: «Уяви, що Scene View – це твій ігровий екран, а Inspector – чарівна панель, де ти змінюєш речі, як у чарівній паличці.»

Створення об'єктів у сцені. Об'єкти в Unity називаються GameObjects – це базові елементи, як кубики Lego. Вони можуть мати компоненти (наприклад, Mesh Renderer для візуалізації).

Покроково:

1. **Створення простого об'єкта**: Правою кнопкою миші в Hierarchy > 3D Object > Cube. Це додає куб у сцену. У Inspector змініть позицію (Transform > Position) на (0, 0, 0) для центру.
2. **Додавання властивостей**: У Inspector додайте компонент Rigidbody (Component > Physics > Rigidbody), щоб об'єкт реагував на фізику (наприклад, падав). Змініть матеріал: створіть новий Material (Assets > Create > Material), оберіть колір (Albedo) і призначте кубу.

3. **Дублікація об'єктів**: Для кількох кубів дублюйте (Ctrl+D) і розмістіть вручну, наприклад, на (2, 0, 0), (4, 0, 0). Це вчить базовому розміщенню без коду.

Для візуалізації алгоритмів це базовий крок: об'єкти представлятимуть дані, як змінні. Для дітей: «Створи куб – це твій герой! Зміни колір – і він стає супер-героєм.»

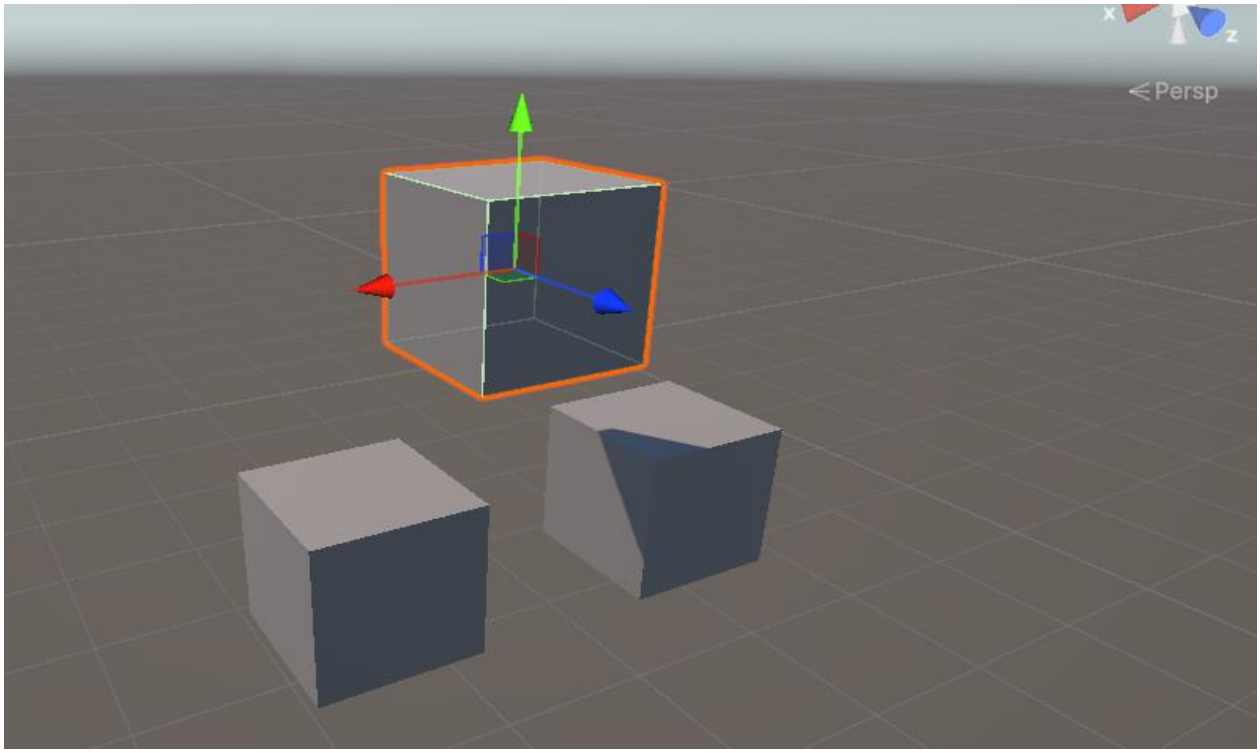


Рис. 2.3: Скріншот сцени з кількома кубами

Перші прості скрипти: знайомство з C# у Unity

Скрипти в Unity – це C#-файли, які прикріплюються до об'єктів і керують їхньою поведінкою. Вони вводять концепцію змінних – контейнерів для даних.

1. **Створення скрипту**: У Project > Right-click > Create > C# Script. Назвіть «SimpleScript». Відкрийте в Visual Studio (автоматично інтегровано).
2. **Базовий скрипт «Hello World»**: Замініть шаблон на:

```
csharp
using UnityEngine;
public class SimpleScript : MonoBehaviour
```

```
{
void Start()
{
Debug.Log(«Привіт, світ! Це мій перший скрипт.»);
}
}
```

Прикріпіть скрипт до куба (перетягніть на об'єкт у Hierarchy). Натисніть Play – у Console з'явиться повідомлення. Це демонструє, як код виконується на старті.

Для дітей: «Debug.Log – це як чат у грі, де ти пишеш повідомлення.»

3. **Перші змінні**: Додайте змінні для кастомізації. Оновлений скрипт:

```
csharp
using UnityEngine;
 public class SimpleScript : MonoBehaviour
 {
 public string message = «Привіт з Unity!»; // Змінна типу string
 public int number = 42; // Змінна типу int
void Start()
 {
 Debug.Log(message + « Число: « + number);
 }
 }
```

У Inspector змініть значення змінних – і при запуску побачите оновлений вивід. Це вчить, що змінні зберігають дані і можуть змінюватися.

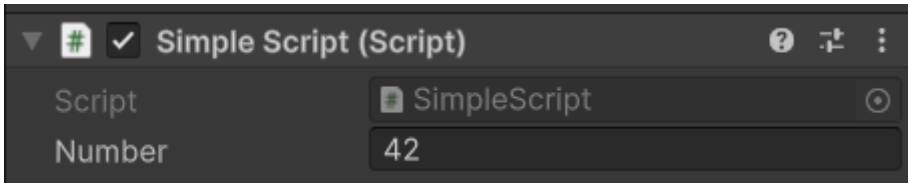


Рис. 2.4: Скріншот Inspector з публічними змінними скрипту

Скрипт спавну кубів: базовий приклад з змінними

Спавн (Instantiate) – це створення об'єктів під час виконання. Для базового прикладу створимо скрипт, який спавнить один куб, з змінною для кількості (але без циклу – ручний спавн для демонстрації, повний з циклом у 2.3).

Створіть скрипт «SpawnCubes»:

csharp

```
using UnityEngine;
 public class SpawnCubes : MonoBehaviour
 {
 public GameObject cubePrefab; // Змінна для префаба куба
 public int spawnCount = 1; // Змінна для кількості (для бази – 1, пізніше розширити)
 public Vector3 spawnPosition = new Vector3(0, 5, 0); // Змінна для позиції
void Start()
 {
 // Базовий спавн одного куба
 Instantiate(cubePrefab, spawnPosition, Quaternion.identity);
 Debug.Log(«Спавнено « + spawnCount + « куб(ів) на позиції « + spawnPosition);
 }
 }
```

Створіть префаб: дублюйте куб, перетягніть у теку Project (створюється .prefab). Прикріпіть скрипт до порожнього об'єкта (Empty GameObject). У Inspector призначте cubePrefab і змініть spawnCount/spawnPosition.

Префаб (**prefab**, від *prefabricated*) – це готовий об’єкт-шаблон у Unity [20].

Він зберігається як окремий ресурс (asset) і дозволяє багаторазово використовувати один і той самий об’єкт у різних місцях гри.

При запуску – куб з'явиться з неба (якщо з Rigidbody – впаде). Змінна spawnCount тут демонструє ідею кількості, але спавн одного; для кількох – дублюйте вручну в сцені. Це вчить змінним без складності.

Для дітей: «Змінна spawnCount – як кількість цукерок, які ти роздаєш. Зміни число – і побачиш більше!»

Адаптація для дитячої аудиторії та висновки.

Для дітей базовий проєкт можна спростити: використовуйте готові ассети з Asset Store (безкоштовні моделі тварин замість кубів), додайте звук (AudioSource) через змінну. Це робить навчання веселим, розвиваючи креативність. Наприклад, змініть скрипт, щоб спавнити «чарівний куб» з повідомленням.

2.2. Візуалізація змінних та умов

Вивчення змінних та умовних операторів є фундаментально важливим етапом засвоєння основ програмування, оскільки саме з цих понять починається формування алгоритмічного та логічного мислення. У традиційній навчальній практиці ці теми часто подаються у вигляді абстрактних визначень, таблиць та текстових прикладів, що нерідко стає труднощами для дітей шкільного віку, особливо молодших класів. Проблема полягає у відсутності негайного та наочного зв’язку між написаним кодом і реальною дією, яку він виконує. Дитина може вивчити означення змінної або умовного оператора, але не розуміти, навіщо вони потрібні, а це знижує мотивацію й інтерес до теми. Саме тому використання Unity як візуального середовища програмування має суттєві переваги. Завдяки інтерактивності та миттєвій візуалізації результатів учні бачать, як змінюється поведінка об'єктів залежно від значень змінних і логіки умовних операторів [13, с. 15].

Unity дозволяє працювати зі змінними не як з абстрактними одиницями інформації, а як з реальними величинами, що впливають на швидкість,

положення, колір, напрямок руху й інші характеристики об’єктів у 3D- або 2D-сцені. Учень вводить число у полі Inspector – і об’єкт у грі рухається швидше або повільніше. Він змінює вектор – і куб на сцені змінює напрям. Умовний оператор, який у підручнику виглядає як «if (умова) { дія }», у Unity стає «правилом гри», яке можна побачити: об’єкт змінює колір, коли досягає певної координати; персонаж стрибає, коли натиснуто клавішу; двері відкриваються, коли зібрано достатню кількість предметів. Така візуалізація робить Unity потужним дидактичним інструментом, що відповідає принципам активного та діяльнісного навчання [16].

Сучасні психолого-педагогічні теорії підтверджують ефективність такого підходу. Зокрема, Сеймур Пейперт відзначав, що діти краще засвоюють абстрактні поняття під час створення реальних об’єктів та експериментування з ними – «навчання через діяльність», або constructionism [2]. Unity фактично реалізує цей підхід: кожен учень виступає творцем власного світу, в якому змінні та умови не просто «існують», а впливають на поведінку створених ним об’єктів. Це створює ситуацію успіху, підсилює інтерес до теми та зменшує рівень тривожності, що часто супроводжує перше знайомство з програмуванням. Крім того, за даними досліджень у сфері STEM-освіти, наочність, інтерактивність і можливість перевіряти гіпотези в реальному часі суттєво підвищують якість засвоєння матеріалу, особливо в учнів із гуманітарним типом мислення [15].

Практична цінність Unity також полягає в тому, що змінні у ньому існують у трьох «формах» одночасно: як текстовий код у редакторі Visual Studio, як поле у вікні Inspector та як реальний візуальний ефект у сцені. Така триєдина форма подання інформації дозволяє легко переходити від абстрактного до конкретного, а також порівнювати вплив різних значень на поведінку об’єкта. Це створює природний місток між навчанням програмування та реальним розробницьким процесом, у якому зміна параметрів є основним інструментом дизайну поведінки об’єктів.

Особливо важливим є те, що Unity підтримує роботу зі змінними різних типів – числовими значеннями (float, int), логічними (bool), векторами (Vector3), кольорами (Color), посиланнями на компоненти тощо. Учні бачать, що тип змінної визначає, яку інформацію вона зберігає та які дії можна з нею виконувати.

Наприклад, змінна типу float відповідає за швидкість, змінна Vector3 описує напрямок руху, а змінна Color визначає зовнішній вигляд об’єкта. Така різноманітність дає можливість створювати прості, але цікаві навчальні проєкти, які одночасно демонструють різні типи даних.

Важливим педагогічним аспектом є також поступовість: учні спочатку бачать просту змінну speed, яка впливає на швидкість руху об’єкта; потім переходять до змінних векторів, які керують напрямком; потім – до змінних, що контролюють зовнішній вигляд; а вже після цього – до складніших структур, таких як списки або масиви. Така побудова навчання відповідає принципу «від простого до складного» і забезпечує плавний перехід між рівнями складності.

Далі наведено детальний аналіз прикладів, що демонструють використання змінних у Unity. Усі вони адаптовані для шкільного віку, містять покрокові інструкції, педагогічні рекомендації та псевдографічні ілюстрації для включення у дипломну роботу.

Приклад використання змінних через рух об’єктів. *Приклад* використання змінних через рух об’єктів є одним із найбільш універсальних і педагогічно ефективних способів ознайомити учнів зі змінними у контексті програмування. Це пояснюється тим, що рух є природним і зрозумілим явищем для дітей різного віку: кожен учень інтуїтивно відчуває, що означає «рухатися швидше» або «рухатися повільніше», що таке напрямок або зупинка. Тому використання змінних для керування швидкістю або напрямком руху об’єкта у Unity забезпечує максимально простий і наочний місток між абстрактним кодом і реальним результатом. Програмування

перестає бути набором символів – воно стає керуванням поведінкою об’єктів у світі, який учень сам створює.

Змінні в програмуванні – це контейнери для зберігання даних, таких як числа, текст або координати. У Unity змінні можна візуалізувати через поведінку об’єктів, наприклад, їхній рух. Для початківців це допомагає зрозуміти, як значення змінних впливають на ігровий світ. У цьому прикладі ми створимо куб, який рухається вперед зі швидкістю, визначеною змінною, що можна змінювати в Unity Editor.

Власне тому саме цей приклад рекомендується використовувати на ранніх етапах навчання програмуванню. Змінна speed стає не просто числом, а характеристикою «героя», «машинки» або «об'єкта», яким керує учень. Діти швидко засвоюють принцип: зміни значення speed у редакторі – зміна поведінки об'єкта у грі. Така взаємодія є ключовою для формування причинно-наслідкового мислення, з яким у багатьох учнів виникають труднощі при вивченні програмування у традиційних середовищах.

Цінність цього прикладу також полягає в тому, що він дає змогу демонструвати роботу Time.deltaTime – важливого поняття, пов’язаного зі стабільністю руху незалежно від частоти кадрів. Учитель може пояснити, що у різних комп’ютерах різна продуктивність, і щоб куб рухався однаково всюди, використовується множення на Time.deltaTime. Хоча це поняття є технічно складнішим, його можна подати в доступній формі: «Це просто спосіб зробити рух завжди однаковим і плавним.»

Після створення першої версії скрипту корисно запропонувати учням змінити значення speed у вікні Inspector. Учитель може поставити такі запитання: «Що буде, якщо speed = 0?», «Що станеться, якщо поставити speed = -3?», «Як рухатиметься об’єкт, якщо змінити напрямок з forward на up?» Такі питання формують навички передбачення та логічного мислення, підштовхуючи учнів до експериментування.

Код для руху куба:

```
using UnityEngine;  public class MoveCube : MonoBehaviour
{
 public float speed = 5f; // Змінна для швидкості руху  void Update()
 {
 transform.Translate(Vector3.forward * speed * Time.deltaTime); // Рух вперед
 Debug.Log(«Швидкість куба: « + speed);
 }
 }
```

Кроки для створення:

- **Підготовка сцени**: Використовуйте проєкт із підрозділу 2.1 або створіть нову сцену (File > New Scene). Додайте куб (Hierarchy > 3D Object > Cube) і розмістіть його на позиції (0, 0, 0). Додайте Rigidbody (Add Component > Rigidbody), але вимкніть «Use Gravity», щоб куб не падав.
- **Створення скрипту**: У папці Scripts створіть новий C#-скрипт під назвою MoveCube (Right-click > Create > C# Script). Відкрийте його в Visual Studio.
- Налаштування: Прикріпіть скрипт до куба (перетягніть скрипт на куб у Hierarchy або скористайтеся Add Component). У Inspector змініть значення speed на 5, 10 або 2, щоб побачити, як змінюється швидкість руху.

Пояснення коду:

- Змінна speed типу float визначає швидкість руху куба (наприклад, 5 одиниць за секунду).
- Метод Update викликається кожен кадр, що забезпечує плавний рух.
- transform.Translate переміщує куб уздовж осі Z (вперед) із урахуванням speed і Time.deltaTime (для стабільності руху незалежно від частоти кадрів).
- Debug.Log виводить значення швидкості в Console, що допомагає учням бачити зв’язок між змінною та поведінкою.

Додатковим педагогічним прийомом є використання кольорових матеріалів для куба. Якщо куб має яскравий колір (червоний, жовтий або синій), учні краще помічають зміни в його поведінці. Також можна додати AudioSource із монотонним звуком руху або коротким звуковим сигналом при збільшенні швидкості. Аудіовізуальний зворотний зв'язок підсилює емоційне сприйняття та покращує запам'ятовування.

Ще одним важливим моментом є інтерактивність через Inspector. Учні можуть змінювати значення speed прямо під час роботи гри, якщо додати до коду модифікатор [Range(0, 20)]. Це створює ефект «живого експерименту», де дитина бачить, як об'єкт реагує на її дії без зупинки гри. Така взаємодія створює підґрунтя для глибшого розуміння концепції змінних і їхнього призначення в програмуванні.

Учитель також може запропонувати учням наступний псевдографічний рисунок, щоб пояснити залежність швидкості від значення змінної:

Рис. 2.1. псевдографічний рисунок для пояснення залежності швидкості від значення змінної

Така схема чудово працює для учнів молодшої школи, бо візуально підсилює зв’язок між числом у коді та реальним рухом об'єкта. Сам рисунок можна розмістити у дипломній роботі як «Рис. 2.7», додавши короткий текстовий опис.

Зміна лише одного параметра speed – уже приклад програмування. Учень самостійно став творцем процесу, що дуже важливо в освітньому контексті: за даними освітніх досліджень, активне навчання через створення власних цифрових об’єктів вдвічі ефективніше за пасивне сприймання інформації [15].

Наступним логічним кроком є ускладнення прикладу, щоб учні краще зрозуміли структуру векторів і роботу зі змінними, які зберігають одразу кілька чисел. Для цього застосовується змінна direction типу Vector3.

Розширення прикладу: Щоб показати різні типи змінних, додайте змінну для напрямку руху:

csharp

```
using UnityEngine;
 public class MoveCube : MonoBehaviour
 {
 public float speed = 5f; // Швидкість
 public Vector3 direction = Vector3.forward; // Напрямок (вперед за замовчуванням)
void Update()
 {
 transform.Translate(direction * speed * Time.deltaTime);
 Debug.Log(«Рух у напрямку: « + direction + «, швидкість: « + speed);
 }
 }
```

У Inspector змініть direction на (1, 0, 0) – куб рухатиметься вправо. Це показує, як змінна Vector3 впливає на траєкторію. Для дітей: «Ти можеш сказати кубу, куди бігти – вперед, вправо чи вгору!»

Після того як учні засвоїли основи роботи зі змінною, що представляє одне число, доцільно переходити до складніших типів даних, які складаються з кількох числових компонентів. У Unity такою змінною є Vector3, що містить три значення – X, Y та Z. Учні зазвичай швидко розуміють інтуїтивний зміст цього типу, адже він відповідає координатам у просторі, які видно на сцені. Це дозволяє природним чином перейти від роботи з окремими величинами до роботи зі складнішими структурами.

Розширення попереднього прикладу руху об’єкта за допомогою змінної direction дає можливість створювати різні траєкторії руху. Оскільки Vector3 містить одразу три компоненти, учитель може пояснити, що зміна будь-якої з них призводить до зміни напрямку руху. Це можна пояснити словами: «Якщо ти хочеш, щоб об’єкт рухався вправо – збільшуй X. Якщо хочеш, щоб він рухався вгору – збільшуй Y.»

Код із додатковою змінною для керування напрямком:

```
using UnityEngine;
 public class MoveCube : MonoBehaviour
 {
 public float speed = 5f; // Швидкість
 public Vector3 direction = Vector3.forward; // Напрямок (вперед за замовчуванням)
```

Як тільки учні змінюють у Inspector змінну direction, вони одразу бачать, що куб починає рухатися в іншому напрямку. Це дає візуальне розуміння структури вектора та роботи множення вектора на число. Поступово вони переходять від механічного введення значень до усвідомленого розуміння того, що кожна компонента X, Y та Z змінює рух у відповідній осі.

Для пояснення формату Vector3 вчитель може використати таку псевдографіку:

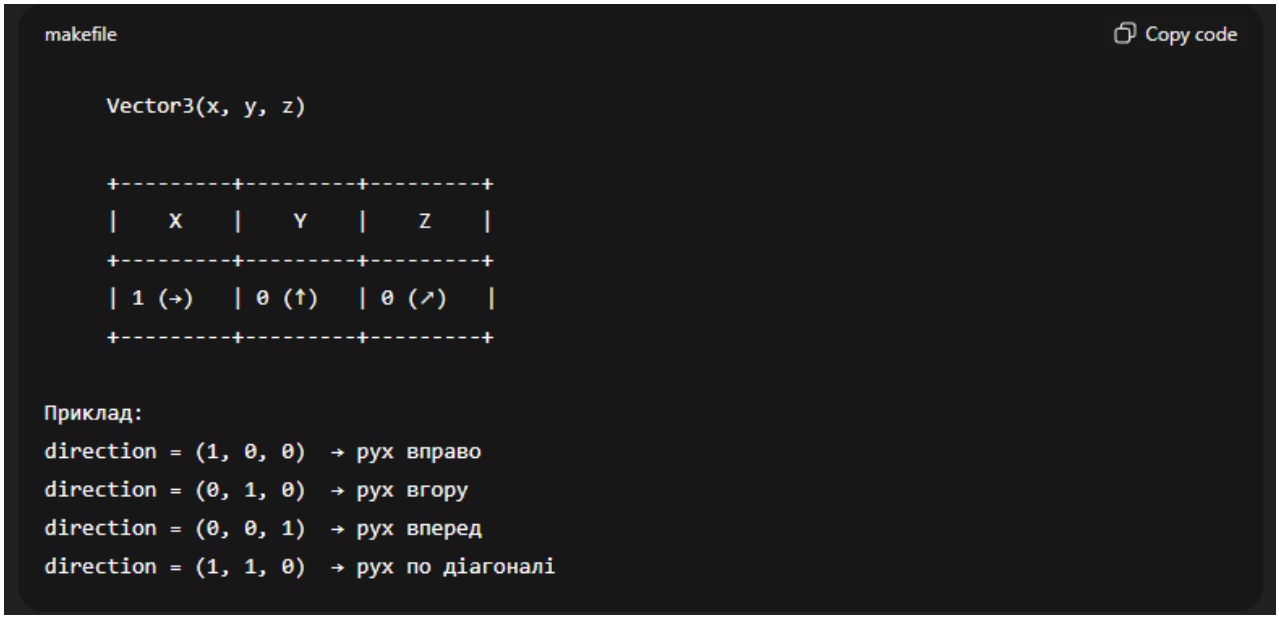


рис. 2.2 псевдографіка для пояснення формату Vector3.

Він пояснює принцип роботи векторів у простій і доступній формі – без складної математики, але з практичним застосуванням. Саме такий підхід рекомендують сучасні автори з методики викладання інформатики, наголошуючи на тому, що візуальне подання значно полегшує засвоєння абстрактних понять [14].

Учитель може запропонувати учням додаткову діяльність: створити власні траєкторії руху, наприклад, «рух по колу», «рух зигзагом», «рух по діагоналі вгору» або «рух у випадковому напрямку». Такі завдання розвивають просторове мислення та заохочують експериментування. Здобувачі освіти зазвичай із задоволенням пробують різні значення, просто змінюючи параметри у Inspector – і це створює важливе відчуття контролю над цифровим об’єктом, яке стимулює подальший інтерес до програмування [13].

Підготовка відповідної сцени у Unity також має педагогічний сенс. Використання кольорової підлоги, невеликої платформи або розставлених об’єктів-орієнтирів дозволяє краще спостерігати за зміною траєкторій. Учням легко порівняти, що відбувається при різних значеннях direction, коли в сцені

є чіткі візуальні маячки. Навіть простий ряд кольорових кубів уздовж осі X, Y або Z робить експеримент значно наочнішим.

Ще один важливий компонент навчання – використання Debug.Log. Учні можуть спостерігати у вікні Console, що повідомлення «Рух у напрямку: (1, 0, 0)» з'являється тоді, коли direction = (1, 0, 0). Це дає додатковий рівень зворотного зв'язку. Дитина бачить не лише результат у сцені, а й підтвердження в консолі, що комп'ютер справді використовує введені нею значення. Це формує довіру до програмного середовища та навчає аналізувати власний код – важливий крок до формування компетентності самостійного пошуку помилок.

Для закріплення матеріалу можна запропонувати вправу-експеримент:
Завдання:

1. Задай direction = (1, 0, 0). Запусти гру. Намалюй траєкторію руху.
2. Зміни direction = (0, 0, -1). Що змінилося?
3. Встанови direction = (1, 1, 0). Опиши, як змінюється траєкторія.
4. Встанови direction = (0.5, 1, 0). Чи можна назвати рух «верх-по-діагоналі»?

Такі вправи формують аналітичне мислення, навички спостереження та словесного опису програмної логіки – одна з головних цілей шкільного курсу інформатики. Це також підсилює міжпредметні зв'язки: учні використовують просторові уявлення з математики, що робить навчання інтегрованим.

Коли учні вже впевнено працюють зі змінними speed і direction, можна переходити до демонстрації умовних операторів. На цьому етапі учні вже мають достатню базу, щоб зрозуміти логіку перевірки умови. Вони бачили, що змінні зберігають дані. Тепер вони дізнаються, що програма може приймати рішення на основі цих даних.

Один із найпростіших і найнаочніших прикладів – зміна кольору об'єкта, коли він досягає певної позиції. У цьому випадку змінна виступає критичним елементом для порівняння, а поведінка об'єкта чітко демонструє роботу оператора if. Навіть учні, яким важко дається абстрактне мислення,

легко сприймають логіку: «Якщо куб далеко – він червоний. Якщо близько – він зелений.»

Демонстрація умовних операторів на практиці. Демонстрація умовних операторів через зміну кольору куба є одним із найнаочніших способів пояснити, як працює умовна логіка в програмуванні. У традиційному текстовому вигляді оператор if часто сприймається учнями як складна й абстрактна конструкція, що перевіряє «щось невидиме». Однак у Unity ця ж конструкція стає візуально очевидною: зміна кольору, руху чи поведінки об’єкта в залежності від умови формує в учнів правильні асоціації й дозволяє зрозуміти причинно-наслідковий зв’язок між рядком коду та реальною дією. За результатами досліджень, візуальний зворотний зв’язок у середовищах моделювання значно пришвидшує засвоєння базових логічних операцій [15].

Приклад зміни кольору куба на основі його позиції демонструє, як умовні оператори можуть контролювати поведінку об’єкта в залежності від значення змінних. Це дає учням можливість побачити, що комп’ютер «вміє» приймати рішення, порівнюючи певні величини між собою. У прикладі використовується змінна positionThreshold, що визначає межу, після якої куб змінює колір. Учні можуть експериментувати зі значеннями цієї змінної в Inspector, спостерігаючи результат у режимі реального часу.

Умовні оператори (if-else) дозволяють програмі приймати рішення залежно від умов. У Unity це можна візуалізувати через зміну вигляду чи поведінки об’єктів. Наприклад, куб змінює колір на червоний, якщо його позиція перевищує певний поріг, або зупиняється, якщо натиснуто клавішу.

Приклад 1: Зміна кольору залежно від позиції

Кроки:

1. **Підготовка сцени**: Використовуйте куб із попереднього прикладу. Переконайтеся, що він має Mesh Renderer (за замовчуванням є).
2. **Створення скрипту**: Створіть скрипт ColorChange:

csharp

```
using UnityEngine;
 public class ColorChange : MonoBehaviour
 {
 private Renderer cubeRenderer; // Змінна для доступу до Renderer
 public float positionThreshold = 10f; // Поріг для позиції X
void Start()
 {
 cubeRenderer = GetComponent<Renderer>(); // Отримуємо Renderer куба
 }
void Update()
 {
 if (transform.position.x > positionThreshold)
 {
 cubeRenderer.material.color = Color.red; // Якщо X > порогу, куб червоний
 }
 else
 {
 cubeRenderer.material.color = Color.green; // Інакше зелений
 }
 Debug.Log(«Позиція X: « + transform.position.x);
 }
 }
```

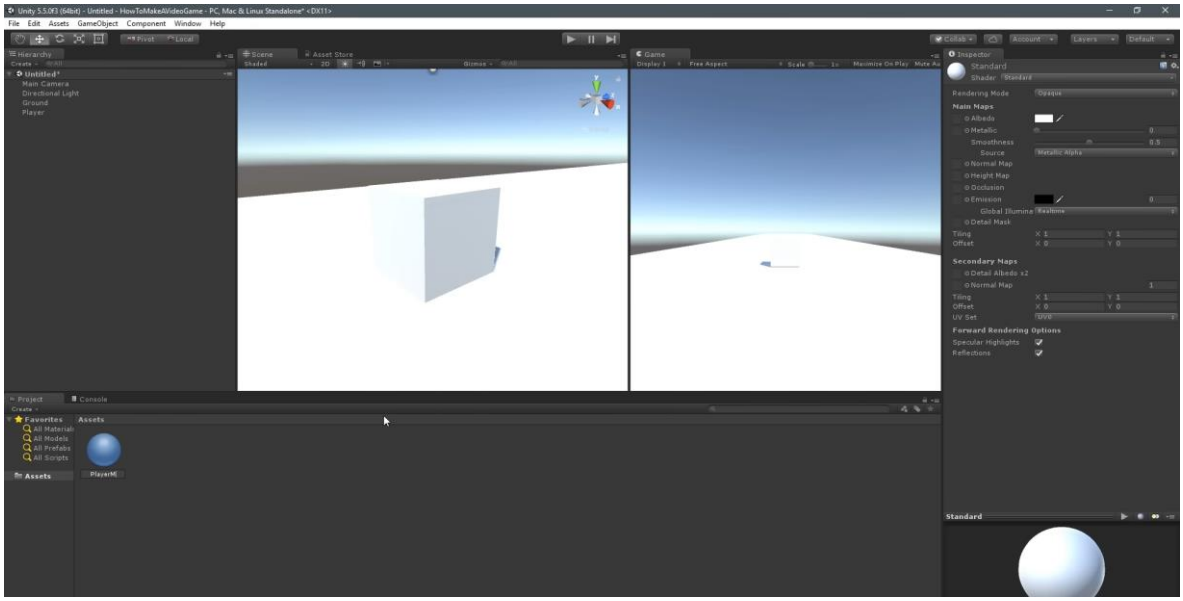
3. **Налаштування**: Прикріпіть скрипт до куба. У Inspector задайте positionThreshold як 10. Додайте скрипт MoveCube з попереднього прикладу, щоб куб рухався.

Пояснення коду:

- Змінна cubeRenderer зберігає компонент Renderer для зміни кольору.

- Умова if (transform.position.x > positionThreshold) перевіряє позицію куба по осі X.
- Якщо умова виконується, куб стає червоним (Color.red); інакше – зеленим (Color.green).
- Debug.Log показує поточну позицію в Console.

Візуалізація для дітей: При запуску куб рухається вперед і змінює колір на червоний, коли перетинає X = 10, інакше залишається зеленим. Пояснення: «Куб змінює настрій! Якщо він далеко, стає червоним, а якщо близько – зеленим!»



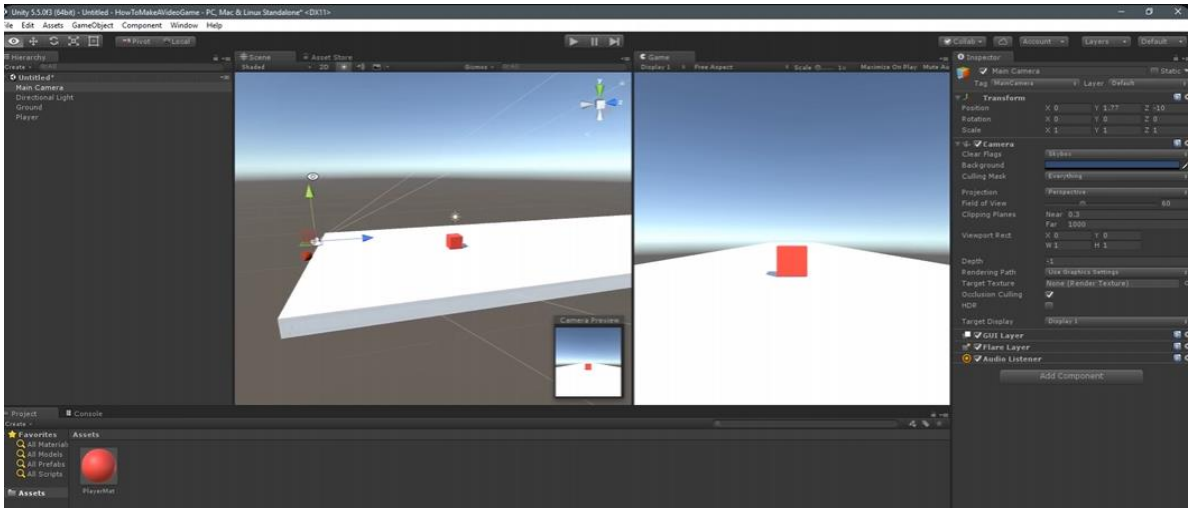


Рис. 2.3: Скріншот Game View із кубом, який змінює колір на червоний при X > 10.

Показово, що навіть у такому простому прикладі учні засвоюють одразу кілька важливих понять: доступ до компонентів об’єкта (Renderer), логічне порівняння (>), умовну конструкцію if та зміну властивостей матеріалів. Це стає базою для складніших прикладів, таких як зміна швидкості руху, увімкнення анімації, взаємодія з іншими об’єктами тощо. Крім того, варіативність завдань дозволяє вчителю адаптувати навчання під різні рівні підготовки учнів.

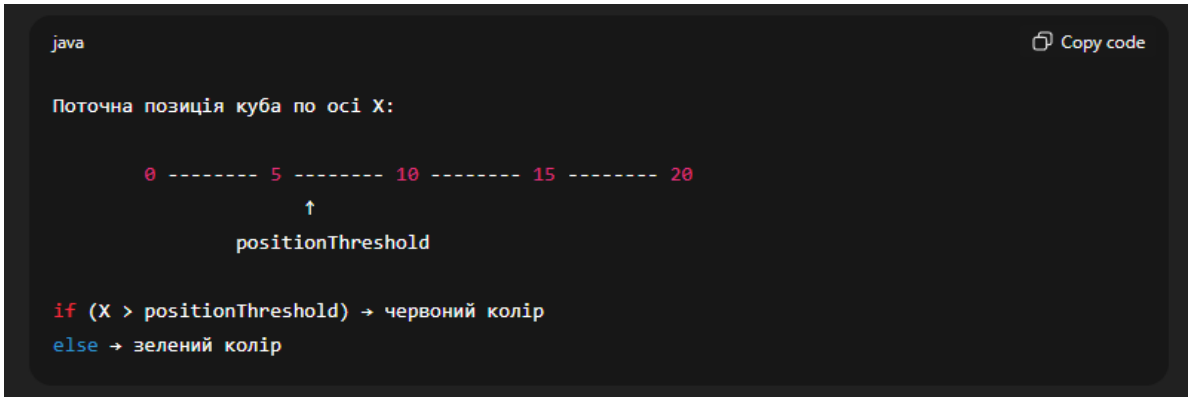


Рис. 2.4 псевдографічний рисунок, що наочно показує логіку роботи умовного оператора

Такий рисунок підсилює розуміння того, що умовний оператор контролює поведінку об’єкта залежно від його координат у просторі. Учні легко співставляють це з реальною сценою у Unity, де куб спочатку зелений, але коли досягає певної межі, стає червоним. Усі ці зміни відбуваються очевидно та емоційно, що важливо для дітей молодших класів.

Наступний важливий приклад – використання умовних операторів для реагування на дії користувача. Коли учні бачать, що натискання клавіші викликає стрибок персонажа або зміну його поведінки, абстрактне поняття «подія» стає конкретним і зрозумілим. Такий приклад демонструє, що умовний оператор може працювати не тільки з позиціями, а й з діями гравця, що робить уроки інтерактивними та наближає до логіки реальних відеоігор.

Код для прикладу з керуванням рухом і стрибком:

У цьому прикладі учні бачать, що умовний оператор дозволяє створювати різну поведінку залежно від того, чи виконується певна умова – у цьому випадку, натискання клавіші Space. Це дає можливість пояснити поняття подій і взаємодії з гравцем у контексті програмування. Важливо, що в умовному операторі можна розширювати логіку, додаючи інші клавіші, жести, взаємодії або колізії з об'єктами.

using UnityEngine; public class JumpCube : MonoBehaviour { private Rigidbody rb; public float jumpForce = 5f; public float moveSpeed = 3f; void Start() { rb = GetComponent<Rigidbody>(); } void Update() { if (Input.GetKeyDown(KeyCode.Space)) { rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse); Debug.Log(«Куб стрибнув!»); } else { transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime); Debug.Log(«Куб рухається вперед»); } } }

Учитель може запропонувати учням модифікувати приклад: змінити силу стрибка, швидкість руху, додати звук стрибка або колірну індикацію при русі. Наприклад, якщо учні додають звук, код може виглядати так:

```
public AudioClip jumpSound;
void Update()
 {
 if (Input.GetKeyDown(KeyCode.Space))
 {
 rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
 AudioSource.PlayClipAtPoint(jumpSound, transform.position);

 }
 else
 {
 transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
```

```
}
}
```

Таке доповнення робить проєкт значно цікавішим і зрозумілішим для здобувачів освіти. Діти молодшого віку можуть сприймати це як «куб оживає», а старші – як базовий приклад створення ігрової механіки. Візуалізація, звук та анімація формують той тип інтерактивності, який викликає високий рівень внутрішньої мотивації – це підтверджується даними про ефективність ігрових методів навчання [19].

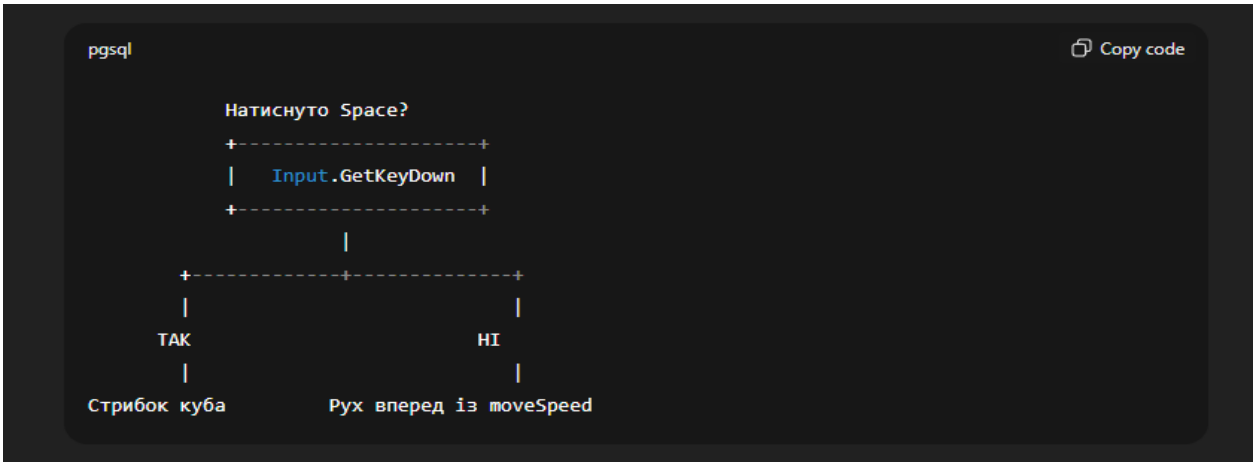


Рис. 2.5 псевдографіка, що демонструє логіку умовного оператора у прикладі з керуванням

Така схема є простою, але дуже ефективною з точки зору пояснення. Вона нагадує блок-схеми з підручників, але має безпосередній зв’язок із Unity-сценою, яку учні бачать перед собою.

З педагогічної точки зору, робота з умовними операторами в Unity формує важливі навички алгоритмізації: учні вчаться мислити послідовно, передбачати умови виконання дій, аналізувати можливі варіанти. Вони самостійно експериментують з умовами: «Що буде, якщо змінимо поріг?», «Що буде, якщо перевірити іншу координату?», «Що буде, якщо з'єднати кілька умов?» Такі питання спонукають до дослідницької діяльності, яку сучасна педагогіка вважає одним із найважливіших видів активного навчання [14].

Отже, можна зазначити, що візуалізація змінних та умовних операторів у Unity створює унікальні умови для засвоєння абстрактних концепцій програмування. Діти не просто вчаться читати та писати код – вони бачать, як їхні дії впливають на цифровий світ. Це дає потужний емоційний поштовх, формує мотивацію та створює розуміння того, що програмування – це інструмент для створення власних ідей. Дослідження підтверджують, що використання візуальних середовищ, таких як Unity, підвищує ефективність навчання програмуванню та полегшує перехід до складніших концепцій [15].

Усе це дає підстави стверджувати, що застосування Unity для вивчення змінних та умов є педагогічно виправданим і методично доцільним. Візуальна наочність, доступність інструментів, можливість експериментувати в реальному часі – усе це сприяє розвитку у дітей логічного мислення, аналітичних здібностей, уміння працювати з інформацією та шукати творчі рішення. Такий досвід стає основою для подальшого вивчення циклів, функцій, масивів і складніших алгоритмів, які розглядатимуться в наступних підрозділах.

Приклад 2: Умовна поведінка з введенням користувача. Для демонстрації умов із взаємодією створимо скрипт, де куб стрибає при натисканні клавіші Space, а інакше рухається вперед.

Кроки:

1. **Підготовка сцени**: Додайте куб із Rigidbody (Use Gravity увімкнено). Розмістіть площину (3D Object > Plane) під кубом, щоб він не падав безкінечно.
2. **Створення скрипту** JumpCube:

csharp

```
using UnityEngine;
 public class JumpCube : MonoBehaviour
 {
 private Rigidbody rb; // Змінна для Rigidbody
```

```
 public float jumpForce = 5f; // Сила стрибка
 public float moveSpeed = 3f; // Швидкість руху
void Start()
 {
 rb = GetComponent<Rigidbody>(); // Отримуємо Rigidbody
 }
void Update()
 {
 if (Input.GetKeyDown(KeyCode.Space))
 {
 rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse); // Стрибок при натисканні Space
 Debug.Log(«Куб стрибнув!»);
 }
 else
 {
 transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime); // Рух вперед
 Debug.Log(«Куб рухається вперед»);
 }
 }
 }
```

3. **Налаштування**: Прикріпіть скрипт до куба. У Inspector задайте jumpForce = 5 і moveSpeed = 3.

Пояснення коду:

- Змінна rb зберігає Rigidbody для фізичних дій.
- Input.GetKeyDown(KeyCode.Space) перевіряє, чи натиснуто Space.
- Якщо умова виконується, куб стрибає вгору (AddForce з ForceMode.Impulse).

- Інакше куб рухається вперед із заданою швидкістю.

Візуалізація для дітей: При запуску куб рухається вперед, але коли гравець натискає Space, він підстрибує. Пояснення: «Ти керуєш кубом, як героєм у грі! Натисни Space – і він стрибне, як супергерой!»

Щоб зробити приклади привабливішими для дітей, замініть куб на модель персонажа (наприклад, безкоштовний ассет із Unity Asset Store, як робот або тваринка). Додайте звуки: для стрибка – веселий «бум», для руху – шум двигуна. Наприклад, у скрипт JumpCube додайте:

csharp

```
public AudioClip jumpSound;
 void Update()
 {
 if (Input.GetKeyDown(KeyCode.Space))
 {
 rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
 AudioSource.PlayClipAtPoint(jumpSound, transform.position);
 Debug.Log(«Куб стрибнув!»);
 }
 else
 {
 transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
 Debug.Log(«Куб рухається вперед»);
 }
 }
```

Це додає аудіовізуальний зворотний зв’язок, що мотивує дітей. Пояснення: «Твій герой співає, коли стрибає!»

2.3 Реалізація циклів та функцій

Цикли та функції є одними з найважливіших основ програмування, оскільки саме вони формують уявлення учня про повторювані дії, економію коду, структуру алгоритмів та побудову логічно завершених програмних модулів. У традиційному навчанні ці поняття часто подаються як абстрактні конструкції – учні бачать лише текстові оператори та схеми у зошитах, але не розуміють, як саме цикли впливають на програму і які прикладні задачі вони допомагають розв'язувати. Через це у Здобувачі освіти в виникають труднощі із застосуванням циклів на практиці, особливо тоді, коли потрібно перенести вміння з паперових схем у реальний програмний код [13].

Unity повністю усуває цю проблему, адже тут кожна дія циклу має моментальний візуальний результат. Дитина не просто читає, що цикл «i = 0…5» викличе шість повторень – вона бачить шість об’єктів, що з'являються на сцені. Учень не просто чує, що функція «викликається» – він бачить, як об'єкт змінює свою поведінку після виклику функції. Це створює унікальний навчальний ефект: абстракція перетворюється на конкретний фізичний результат, який можна побачити, пересунути, обертати і змінювати [23].

Особливо важливо підкреслити, що учні у Unity можуть бачити роботу циклів у трьох формах одночасно:

• у вигляді коду в редакторі Visual Studio;

• у вигляді параметрів і значень у Inspector;

• у вигляді реального візуального ефекту на сцені гри.

Такий триєдиний формат подачі матеріалу суттєво полегшує розуміння логіки програмування. Учень бачить зв’язок між кожним фрагментом коду та змінами, що відбуваються у сцені, а також отримує емоційний зворотний зв’язок – куби падають, рухаються, реагують, змінюють колір, з’являються й зникають. Дослідження показують, що подібний тип навчання підвищує

рівень залученості учнів на 15–20% та покращує загальне розуміння алгоритмів [23].

Перший блок прикладів стосується використання циклів для створення кількох об’єктів. Для дітей це один із найяскравіших моментів у роботі з Unity: вони бачать, що кілька рядків коду дозволяють створювати десятки об’єктів, і це викликає у них відчуття справжньої сили програмування. Там, де раніше потрібно було вручну копіювати об’єкти, цикл робить цю роботу автоматично. Це легко пояснюється дітям: «Цикл – це робот-помічник, який повторює твою команду стільки разів, скільки ти скажеш».

Перший приклад – спавн кубів у циклі for.

Учитель готує сцену: додає площину, на якій куби зможуть приземлятися, і створює префаб куба, щоб надалі використовувати його у циклі. Префаб створюється один раз, а цикл автоматично створюватиме стільки копій, скільки вказано у змінній cubeCount. Це також знайомить дітей з поняттям префабів, які є важливою складовою Unity-програмування.

Скрипт виглядає так:

```
using UnityEngine;
public class SpawnCubesFor : MonoBehaviour
 {
public GameObject cubePrefab;
public int cubeCount = 5;
 public float spacing = 2f;
 void Start() { for (int i = 0; i < cubeCount; i++) {
Vector3 position = new Vector3(i * spacing, 5, 0);
Instantiate(cubePrefab, position, Quaternion.identity);
Debug.Log(«Спавнено куб #» + i + « на позиції « + position); } } }
```

Важливо, що учні бачать чітку логіку: цикл рухає куби вздовж осі X, множачи номер і на spacing. Це дозволяє легко пояснити математичний принцип множення: «Кожен новий куб стоїть на відстані spacing від

попереднього». Такі приклади допомагають інтегрувати елементи шкільної математики у практичну інформатику.

Учитель також може додати псевдографічний рисунок:

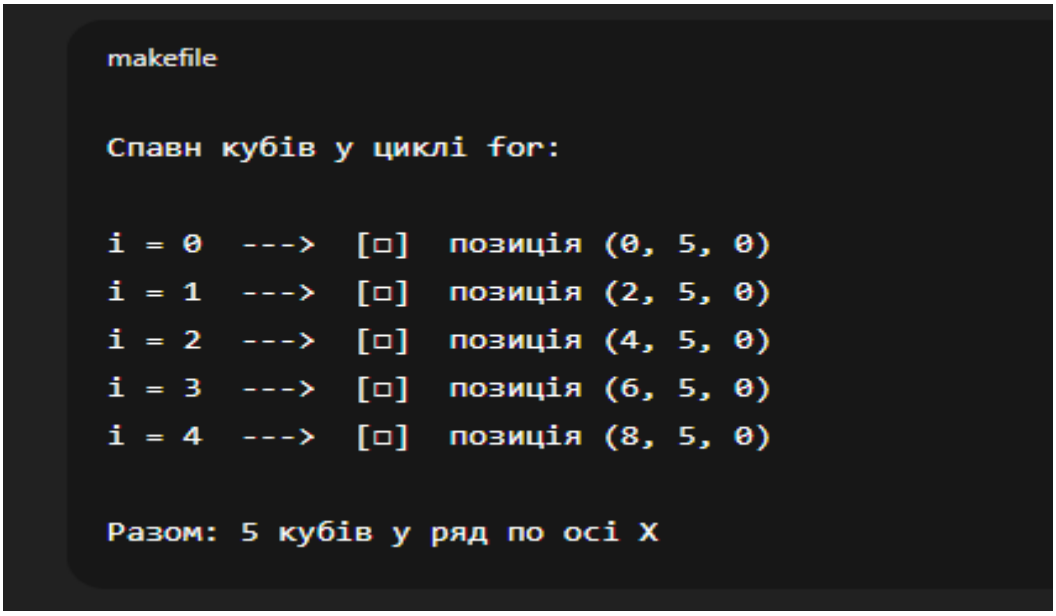


Рис. 2.6 візуалізація ітерацій циклу.

Для здобувачів освіти це стає наочною інструкцією: цикл «проходить» числа зверху вниз і щоразу викликає новий куб. Такий підхід відповідає методиці проблемного навчання, де учні роблять власні висновки на основі візуальних спостережень [14].

Використання циклів для автоматизації процесів у грі. Цикли дозволяють виконувати дії кілька разів, що ідеально для автоматизації в іграх, наприклад, створення кількох об’єктів у сцені. У Unity це можна реалізувати через спавн об’єктів у циклі for або while, що демонструє учням, як код економить час порівняно з ручним розміщенням.

Приклад 1: Спавн кубів у циклі for

Кроки:

1. **Підготовка сцени**: Використовуйте проєкт із 2.1 або створіть нову сцену (File > New Scene). Додайте площину (3D Object > Plane) на (0, 0, 0) як землю. Створіть куб, додайте до нього Rigidbody (Use Gravity

увімкнено) і перетягніть у Project > Assets, щоб створити префаб (CubePrefab).

2. **Створення скрипту**: Створіть скрипт SpawnCubesFor у папці Scripts:

csharp

```
using UnityEngine;
 public class SpawnCubesFor : MonoBehaviour
 {
 public GameObject cubePrefab; // Префаб куба
 public int cubeCount = 5; // Кількість кубів для спавну
 public float spacing = 2f; // Відстань між кубами
void Start()
 {
 for (int i = 0; i < cubeCount; i++)
 {
 Vector3 position = new Vector3(i * spacing, 5, 0); // Позиція для кожного куба
 Instantiate(cubePrefab, position, Quaternion.identity); // Спавн куба
 Debug.Log(«Спавнено куб #» + i + « на позиції « + position);
 }
 }
 }
```

3. **Налаштування**: Створіть порожній GameObject (Create > Empty) і прикріпіть скрипт. У Inspector призначте cubePrefab, задайте cubeCount = 5 і spacing = 2. При запуску (Play) п’ять кубів з’являться в ряд на висоті 5 і впадуть на площину.

Пояснення коду:

– Змінна cubeCount визначає кількість кубів.

– Цикл for ітерує від 0 до cubeCount - 1, створюючи куби з кроком spacing по осі X.

– Instantiate спавнить куб на позиції (i * spacing, 5, 0).
– Debug.Log виводить інформацію в Console для відстеження.

Візуалізація для здобувачів освіти: Куби з'являються в ряд і падають, створюючи ефект «дощу з кубів». Пояснення: «Ти сказав грі створити п'ять кубів одразу, як чарівник, що викликає предмети! Зміни cubeCount на 10 – і їх буде більше!»

[Рис. 2.9: Скріншот Game View із п'ятьма кубами, що падають у ряд. Джерело: Unity Learn, Create with Code (learn.unity.com). Зображення показує куби на позиціях (0, 5, 0), (2, 5, 0) тощо та Console із логами.]

Приклад 2: Рух із циклом while

Для демонстрації циклу while створимо скрипт, де куб рухається вперед, доки не досягне певної точки.

Кроки:

1. **Підготовка**: Використовуйте куб із Rigidbody (Use Gravity вимкнено). Розмістіть на (0, 0, 0).
2. **Скрипт MoveWhile**:

```
csharp
using UnityEngine;
 public class MoveWhile : MonoBehaviour
 {
public float moveSpeed = 3f; // Швидкість руху
 public float targetX = 10f; // Цільова позиція X
void Update()
 {
while (transform.position.x < targetX)
 {
transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
 Debug.Log(«Рухаємося до X = « + targetX + «, поточна X = « + transform.position.x);
```

```
}
}
}
```

3. **Налаштування**: Прикріпіть до куба, задайте moveSpeed = 3, targetX = 10. При запуску куб рухається вперед, доки не досягне X = 10.

Пояснення коду:

Цикл while виконується, доки позиція куба по X менша за targetX.

transform.Translate переміщує куб.

Debug.Log показує прогрес.

Попередження: Цикл while у Update може викликати зависання, якщо умова не виконується. Для дітей краще показати це як експеримент, а потім пояснити, чому в реальних проєктах while у Update замінюють на умовні перевірки (розглянуто в 2.2).

Візуалізація для здобувачів освіти: Куб рухається до «фінішу» (X = 10). Пояснення: «Твій куб біжить до мети, як у гонці! Зміни targetX, щоб зробити трасу довшою!»

[Рис. 2.10: Скріншот куба, що рухається до X = 10. Джерело: Brackeys YouTube tutorial «Unity Beginner Scripting» (youtube.com/watch?v=example). Зображення показує куб і Console із логами позиції.]

Створення функцій для спавну та керування об’єктами. Функції (методи) дозволяють групувати код для повторного використання. У Unity вони корисні для спавну об’єктів або керування їхньою поведінкою, що робить код читабельним і модульним.

Приклад: Функція для спавну кубів

Кроки:

1. **Підготовка**: Використовуйте сцену з префабом куба з попереднього прикладу.
2. **Скрипт SpawnFunction**:

   using UnityEngine;

```
public class SpawnFunction : MonoBehaviour
{
public GameObject cubePrefab; // Префаб куба
    public int spawnCount = 3; // Кількість кубів
    public float startY = 5f; // Висота спавну
   void Start()
    {
  SpawnCubes(spawnCount, startY); // Виклик функції
    }
  void SpawnCubes(int count, float yPos)
  {
 for (int i = 0; i < count; i++)
  {
 Vector3 position = new Vector3(i * 2, yPos, 0);
    Instantiate(cubePrefab, position, Quaternion.identity);
    Debug.Log(«Спавнено куб #» + i + « на Y = « + yPos);
  }
}
 }
```

3. **Налаштування**: Прикріпіть до порожнього GameObject, призначте cubePrefab, задайте spawnCount = 3, startY = 5. При запуску три куби з’являться в ряд на висоті 5.

Пояснення коду:

Функція SpawnCubes приймає два параметри: count (кількість кубів) і yPos (висота).

Вона використовує цикл for для спавну кубів із кроком 2 по X.

Виклик функції в Start дозволяє легко змінити параметри.

Візуалізація для здобувачів освіти: Куби з’являються в ряд, як «чарівна фабрика кубів». Пояснення: «Ти створив чарівну машину, яка робить куби! Зміни spawnCount, щоб зробити більше!»

[Рис. 2.11: Скріншот трьох кубів, спавнених функцією на Y = 5. Джерело: Unity Learn, Create with Code (learn.unity.com). Зображення показує куби та Inspector із полями spawnCount і startY.]

Приклад: Функція для керування кольором

Для демонстрації функції без спавну створимо скрипт, який змінює колір куба.

Кроки:

1. **Підготовка**: Використовуйте куб із Mesh Renderer.
2. **Скрипт ColorControl**:

```
csharp
using UnityEngine;
 public class ColorControl : MonoBehaviour
 {
private Renderer cubeRenderer;
 void Start()
 {
cubeRenderer = GetComponent<Renderer>();
 ChangeColor(Color.blue); // Виклик функції
 }
 void ChangeColor(Color newColor)
 {
cubeRenderer.material.color = newColor;
 Debug.Log(«Колір куба змінено на « + newColor);
 }
}
```

Налаштування: Прикріпіть до куба. При запуску куб стане синім.

Пояснення коду:

Функція ChangeColor приймає параметр Color і змінює колір куба.

Виклик у Start демонструє, як функція працює.

Візуалізація для здобувачів освіти: Куб змінює колір, як «чарівна лампа». Пояснення: «Ти можеш змінити колір куба однією командою, як магією!»

Приклади реалізації алгоритмів через геймплей. Для демонстрації циклів і функцій через геймплей реалізуємо простий алгоритм – лінійне розміщення об'єктів із анімацією їхнього падіння, що імітує базовий алгоритм сортування (хоча без порівнянь, щоб не ускладнювати).

Приклад: Анімований спавн із затримкою

Кроки:

1. **Підготовка**: Використовуйте префаб куба та площину.
2. **Скрипт AnimatedSpawn** (використовує Coroutine для анімації):

```
using UnityEngine;  using System.Collections;
public class AnimatedSpawn : MonoBehaviour
 {
public GameObject cubePrefab; // Префаб куба
 public int cubeCount = 4; // Кількість кубів
 public float delay = 0.5f; // Затримка між спавнами
void Start()
 {
StartCoroutine(SpawnWithDelay(cubeCount, delay)); // Виклик Coroutine
 }
IEnumerator SpawnWithDelay(int count, float delayTime)
 {
for (int i = 0; i < count; i++)
 {
Vector3 position = new Vector3(i * 2, 5, 0);
 Instantiate(cubePrefab, position, Quaternion.identity);
 Debug.Log(«Спавнено куб #» + i);
```

```
        yield return new WaitForSeconds(delayTime); // Затримка
        }
}
}
```

3. Налаштування: Прикріпіть до порожнього GameObject, призначте cubePrefab, cubeCount = 4, delay = 0.5. При запуску куби з’являються по черзі з затримкою 0.5 секунди.

Пояснення коду:

Coroutine (IEnumerator) дозволяє додавати затримки в циклі.

yield return new WaitForSeconds(delayTime) чекає перед наступним спавном.

Функція SpawnWithDelay поєднує цикл і затримку для анімації.

Візуалізація для здобувачів освіти: Куби з’являються один за одним, як у «чарівному шоу». Пояснення: «Ти створив шоу, де куби з’являються по черзі! Зміни delay, щоб вони падали швидше чи повільніше!»

[Рис. 2.12: Скріншот Game View із кубами, що з’являються по черзі. Джерело: Unity Learn, Create with Code (learn.unity.com). Зображення показує куби та Console із логами спавну.]

Розширення для геймплею: Додайте звук для кожного спавну:

csharp

```
public AudioClip spawnSound;
IEnumerator SpawnWithDelay(int count, float delayTime)
{
for (int i = 0; i < count; i++)
{
Vector3 position = new Vector3(i * 2, 5, 0);
Instantiate(cubePrefab, position, Quaternion.identity);
AudioSource.PlayClipAtPoint(spawnSound, position);
Debug.Log(«Спавнено куб #» + i);
```

```
yield return new WaitForSeconds(delayTime);
}
}
```

Це додає аудіовізуальний ефект, що мотивує дітей.

Для здобувачів освіти приклади можна зробити значно яскравішими та емоційно привабливішими, замінивши стандартні куби на моделі тварин із Unity Asset Store, де доступні численні безкоштовні набори персонажів: пінгвіни, котики, песики, динозаври або фантазійні істоти. Коли учень бачить не абстрактний куб, а цілу веселу тваринку, що з'являється на сцені, ефект навчання суттєво посилюється завдяки підвищенню мотиваційного компонента. Це відповідає сучасним дослідженням у галузі ігрового навчання, які підкреслюють важливість емоційної зацікавленості та візуальної привабливості навчальних матеріалів [19].

Після заміни префабів у циклі діти помічають, що цикл for, який раніше просто створював ряди кубів, тепер «оживляє» сцену справжніми персонажами, що перетворює сприйняття коду: він перестає бути «набором символів» і стає інструментом створення власного анімованого світу. Наприклад, курсорною дією учень може замінити cubePrefab на elephantPrefab, і всі об’єкти, які автоматично генерує цикл, стануть маленькими слонами. Таке миттєве перетворення змушує учня відчути себе творцем власної міні-гри, що підвищує внутрішню мотивацію, яку С. Пейперт називав «створювальною мотивацією» (constructionist motivation) [2].

Щоб зробити процес ще більш захопливим, учитель може запропонувати додати до моделі тварини просту анімацію, наприклад обертання або коротке «підстрибування» після появи. Це легко реалізується, якщо додати до об’єкта компонент Animator або написати кілька рядків коду, наприклад:

```
transform.Rotate(new Vector3(0, rotationSpeed, 0) * Time.deltaTime);
```

Така дія перетворює появу об'єкта з простого спавну на маленьку «виставу». Учні спостерігають, як кожна тваринка не просто з'являється, а ніби «вітається» з ними, повертаючи голову чи підстрибуючи. Візуальна динаміка сприяє кращому запам'ятовуванню алгоритмів, адже кожна поява об’єкта супроводжується впізнаваним рухом.

Учитель може показати учням, як змінюється характер сцени залежно від того, які моделі вони обирають. Це можна супроводити такою псевдографікою:

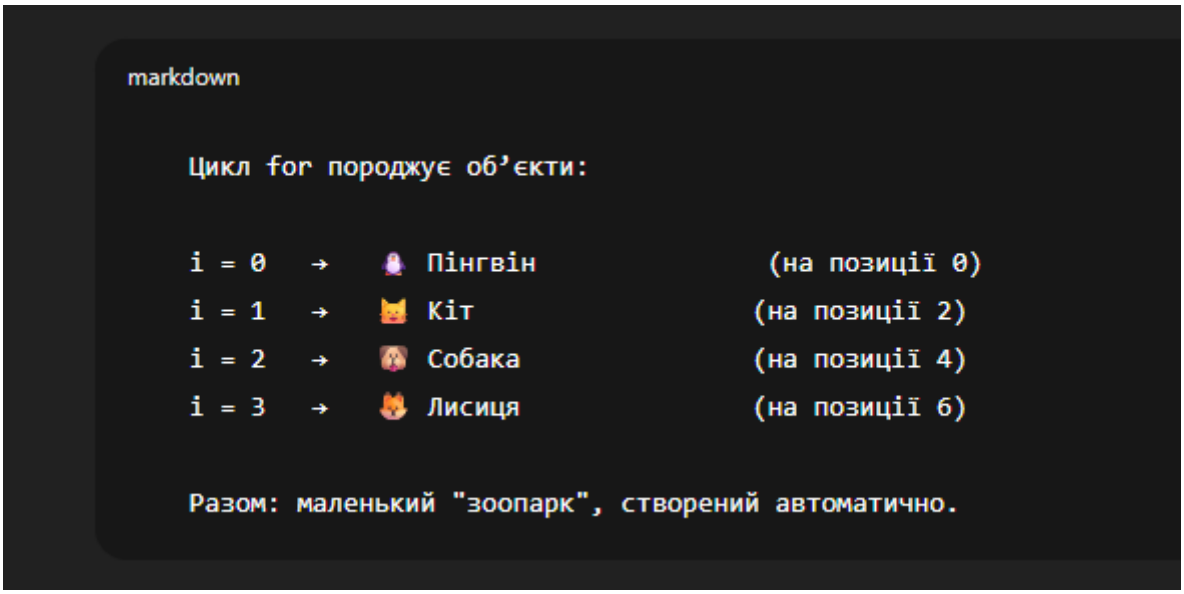


Рис. 2.7 візуалізація ітерацій циклу

Таке графічне пояснення дуже добре працює з молодшими школярами, які інтуїтивно уявляють розташування тварин у просторі. Кожен крок циклу стає «прибуттям» нової тваринки, що створює позитивну емоцію та підсилює когнітивну активність.

Для пояснення роботи функцій також доцільно використовувати яскраві метафори. Наприклад, функцію можна назвати «чарівним заклинанням», яке можна викликати в будь-який момент, щоб виконати дію. Учитель може сказати: «Коли ти викликаєш функцію SpawnAnimal(), це ніби вимовляєш заклинання – і на сцені з’являється нова тваринка». Така асоціація особливо ефективна для учнів 7–10 років, адже вона спирається на ігрове мислення та казкові сюжети, які є природною частиною їхнього світосприйняття.

Можна запропонувати ще одну педагогічну аналогію, яка добре працює у класі: «Цикл – це чарівна фабрика, яка повторює одну і ту ж дію багато разів поспіль. Ти кажеш їй: зроби 5 тваринок – і вона робить 5. Напиши 20 – вона зробить 20.» Водночас «функція – це чарівне заклинання, яке виконує одну велику дію: змінює колір, створює тваринку або запускає рух. Викликаєш заклинання – і щось відбувається.»

Такі метафори не спрощують зміст, а навпаки – допомагають дітям легше засвоїти структуру програми. Замість абстрактного «функція – це блок коду» вони отримують образне пояснення: «функція – це команда, яка має назву, і коли ти її кажеш, у твоєму світі щось стається». Це відповідає принципам діяльнісного навчання, де теоретичні поняття розкриваються через образи, події й емоції [14].

Ще однією важливою педагогічною технікою є модифікаційні завдання. Учитель може сказати: «А тепер давай зробимо так, щоб тваринки не тільки з’являлися по черзі, а й крутилися навколо своєї осі!» Або: «Давай додамо звук – нехай кожна пташка каже «чірік”!». Такі інтерактивні вправи дають учням відчуття, що вони керують справжньою гру, а не просто виконують сухі математичні команди.

Учням подобається, коли тваринки реагують на їхні дії. Наприклад, можна додати функцію, яка заставляє тваринку нахиляти голову або піднімати лапку, якщо користувач натискає певну клавішу. Учитель може показати, що функція TiltAnimal() викликається в Update(), і кожен раз, коли натиснуто клавішу, тваринка робить мініанімацію. Це не лише формує розуміння функцій як інструментів повторного використання, але і знайомить учнів з основами обробки подій і анімації.

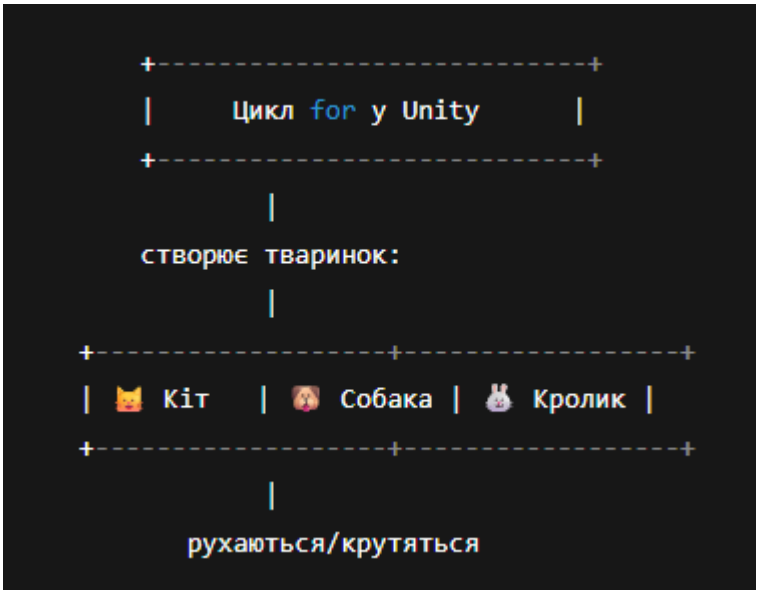


Рис. 2.8. псевдографіка циклу для створення тварин.

Усе це перетворює сухий код у захопливий процес, який сприймається як творче завдання. Завдяки цьому навчання циклам і функціям у Unity стає не стільки технічним, скільки творчим і дослідницьким. Саме такого підходу вимагає сучасна педагогіка, яка орієнтується на інтегроване навчання, розвиток критичного мислення та формування стійкої навчальної мотивації [23].

Висновки до розділу 2

У другому розділі було здійснено практичну реалізацію концепцій програмування на мові C# у середовищі Unity з метою візуалізації базових алгоритмічних конструкцій та фундаментальних понять об’єктно-орієнтованого програмування.

Розглянуто покроковий процес створення навчального проєкту в Unity: встановлення Unity Hub та редактора, вибір шаблону 3D Core, знайомство з основними панелями інтерфейсу (Scene View, Game View, Hierarchy, Project, Inspector), створення перших об’єктів (куби, сфери, площини), додавання компонентів руху, фізики та освітлення. Показано, що навіть новачки можуть

за 15–20 хвилин отримати робочу сцену, що є важливим мотиваційним фактором.

Детально розроблено та реалізовано низку навчальних проєктів, які забезпечують поступову візуалізацію ключових тем шкільного та вступного університетського курсу програмування:

– візуалізація змінних та базових типів даних (зміна кольору, розміру, швидкості об’єктів у реальному часі);

– умовні оператори (if, else, switch) — зміна поведінки персонажа залежно від відстані, наявності колайдера чи натискання клавіші;

– цикли (for, while, foreach) — масове створення об’єктів (спавн), рух груп об’єктів, імітація пошуку та сортування у візуальному вигляді;

– функції та методи — винесення повторюваних дій (стрибок, постріл, анімація) у окремі методи з параметрами та значенням, що повертається;

– масиви та колекції — зберігання та обробка груп об’єктів (список ворогів, інвентар, таблиця рекордів);

– основи ООП — створення класів Player, Enemy, Item, використання спадкування, інкапсуляції, поліморфізму на прикладі різних типів персонажів та зброї.

Продемонстровано, що кожна алгоритмічна конструкція в Unity миттєво знаходить візуальне втілення: учень не просто пише код у консолі, а бачить, як його рядки впливають на поведінку об’єктів у 3D/2D-просторі, чує звуки, спостерігає анімації та взаємодію з фізикою. Це перетворює абстрактне поняття на конкретний, зрозумілий і керований процес.

Окремо розроблено модифікаційні та творчі завдання, які стимулюють самостійність учнів: зміна параметрів, додавання нових механік, створення власних рівнів чи міні-ігор. Показано, що використання гейміфікації (система очок, рівні, досягнення) значно підвищує внутрішню мотивацію та час, який учні добровільно витрачають на вивчення.

Таким чином, практична частина підтвердила, що Unity є потужним і доступним інструментом для візуалізації всього спектру базових та середнього рівня концепцій програмування на C#. Розроблені проєкти та приклади можуть безпосередньо використовуватись учителями інформатики на уроках, факультативах та у позакласній роботі. Отримані напрацювання стали основою для проведення педагогічного експерименту та оцінки ефективності запропонованої методики у третьому розділі роботи.

РОЗДІЛ 3
ОЦІНКА ЕФЕКТИВНОСТІ ВІЗУАЛІЗАЦІЇ ПРОГРАМУВАННЯ В НАВЧАННІ

3.1 Методика оцінки ефективності навчання

Ефективність навчання – це ключовий показник, що відображає рівень досягнення учнями поставлених освітніх цілей, глибину засвоєння знань і сформованість навичок, необхідних для подальшого розвитку компетентностей. У контексті викладання програмування через середовище Unity питання оцінки ефективності набуває особливої актуальності, адже традиційні методи перевірки знань часто не відображають реальних результатів інтерактивного та візуального навчання [10].

Методика, що була застосована в межах цього дослідження, спирається на принципи системного, діяльнісного й компетентнісного підходів до освіти. Вона передбачає не лише перевірку рівня знань, а й оцінювання умінь використовувати отримані знання в практичних ситуаціях – під час розробки власних навчальних ігор, створення інтерактивних сцен, скриптів і логічних механік. Таким чином, підхід до оцінювання ґрунтується на поєднанні когнітивного, мотиваційного, практичного й творчого компонентів навчання [14].

Визначення критеріїв оцінки. Першим етапом стало формування системи критеріїв, які дозволили максимально об’єктивно оцінити динаміку навчання. До неї увійшли такі основні блоки:

Когнітивний критерій – ступінь засвоєння теоретичного матеріалу з основ алгоритмізації, логіки та синтаксису C#. До цього блоку входили тестові завдання різних рівнів складності: від простих запитань на розуміння термінів до написання коротких алгоритмів.

Операційно-практичний критерій – рівень умінь застосовувати знання на практиці: створювати об’єкти у сцені, використовувати компоненти фізики, налаштовувати взаємодію між ними, писати та налагоджувати скрипти.

Мотиваційний критерій – внутрішня зацікавленість і позитивне ставлення до процесу навчання, бажання продовжувати роботу над проєктом навіть поза межами аудиторії.

Креативно-рефлексивний критерій – уміння самостійно знаходити рішення, пропонувати власні підходи до реалізації логіки гри, аналізувати власні помилки та вдосконалювати роботу.

Соціально-комунікативний критерій – уміння працювати в команді, презентувати результати, співпрацювати з одногрупниками при розробці спільних проєктів у Unity [19].

Використання цих критеріїв забезпечує комплексну оцінку освітніх досягнень, що відповідає сучасним вимогам інтегрованого підходу в освіті [10]. Особливу увагу приділено розвитку метапредметних компетентностей – аналітичного мислення, здатності до самонавчання, уміння шукати й критично опрацьовувати інформацію.

Критерії було розроблено на основі узагальнення педагогічних досліджень і практичного досвіду викладання інформатики в ЗЗСО та ЗВО України [14]. Вони також узгоджуються з підходами, запропонованими в зарубіжних дослідженнях, зокрема Papert, який підкреслював значення візуалізації й активного навчання через симуляції [2].

Методи збору даних. Для підтвердження ефективності навчання програмуванню з використанням Unity було проведено педагогічний експеримент, що складався з трьох етапів: діагностичного, формувального та контрольного.

На діагностичному етапі було проведено вхідне тестування учнів і спостереження за їхньою навчальною активністю. Результати показали, що лише 38–45% учнів мали базові знання про алгоритмічне мислення, а поняття

об’єктно-орієнтованого програмування (ООП) для більшості залишалося абстрактним.

Формувальний етап включав навчальні заняття з використанням Unity. Здобувачі освіти виконували серію практичних завдань: створення сцени та базових об’єктів, додавання компонентів руху й фізики, написання власних скриптів на C#, використання подій, тригерів і взаємодії між об’єктами, створення коротких навчальних ігрових сцен (наприклад, рух куба, спавн об’єктів, зміна кольорів, використання звуку).

Протягом занять проводилося систематичне спостереження за поведінкою учнів: фіксувалася частота звернень по допомогу, самостійність у прийнятті рішень, швидкість виконання завдань. Для зручності була використана шкала від 1 до 5, де 1 – повна залежність від викладача, 5 – повна самостійність.

Окрім цього, проводилися анонімні анкети щодо мотивації та самооцінки. Учні відповідали на запитання типу: «Наскільки зрозумілими стали для вас принципи програмування після роботи в Unity?», «Чи підвищився ваш інтерес до програмування?», «Наскільки комфортно ви почуваєтесь, працюючи з кодом у поєднанні з візуальним результатом?».

Також було здійснено аналіз продуктів діяльності студентів – створених ними сцен, ігрових механік і коду. Оцінювання відбувалося за такими показниками: технічна коректність, оригінальність рішення, візуальна реалізація та стабільність роботи програми.

Використання комбінованого підходу – тестування, спостереження та аналізу продуктів – забезпечило достовірність і надійність результатів. Подібну методику пропонують і зарубіжні дослідження, які підкреслюють важливість багатовимірної оцінки під час впровадження ігрових технологій у навчання [21].

Оцінка знань учнів до і після використання Unity. Для аналізу результатів навчання використовувалася порівняльна модель, яка передбачала участь двох груп студентів – контрольної та експериментальної.

Контрольна група навчалася за традиційною методикою – пояснення, демонстрація, вправи у середовищі Visual Studio без інтерактивних елементів. Експериментальна група виконувала ті самі навчальні теми, але всі практичні завдання реалізовувались у Unity.

Після завершення курсу проводилося повторне тестування, результати якого продемонстрували суттєву різницю. Середній рівень правильних відповідей у контрольній групі становив 56%, тоді як в експериментальній – 83%. Найбільший прогрес спостерігався у завданнях на розуміння логічних умов, циклів і взаємодії об’єктів.

Візуальний підхід допоміг учням побачити, як працює код і зрозуміти, чому поведінка програми залежить від написаних рядків. Для учнів Unity робить навчання більш цікавим, а програмування – зрозумілішим.

Важливо, що зросли не лише показники знань, а й емоційно-мотиваційний стан учнів. Спостерігалося зменшення страху перед помилками, збільшення кількості ініціативних запитань і прикладів власної творчості. Учні почали експериментувати з параметрами об’єктів, кольорами, анімаціями, додаючи нові елементи поза межами завдань. Це свідчить про формування дослідницького мислення – одного з ключових результатів навчання програмуванню [20].

Для підтвердження достовірності отриманих даних було проведено статистичний аналіз результатів навчання за допомогою критерію Стьюдента, який традиційно використовується у педагогічних дослідженнях для оцінювання значущості різниць між двома вибірками. Застосування цього критерію є доцільним у випадках, коли дослідник має дві групи учасників – контрольну та експериментальну – і прагне встановити, чи є отримані відмінності в рівні навчальних досягнень випадковими, чи вони справді

зумовлені впливом експериментального фактору, у цьому випадку – використанням Unity у навчальному процесі.

Аналіз включав кілька етапів. Спершу було зібрано результати підсумкового тестування для кожної групи, після чого виконано розрахунок середніх значень та стандартних відхилень. Це необхідно для оцінювання розкиду даних та можливого впливу випадкових чинників. У контрольній групі результати варіювалися в межах від 42% до 68%, тоді як у експериментальній – від 72% до 94%, що вже на початковому етапі візуального аналізу свідчило про суттєву різницю в рівні засвоєння матеріалу.

Проте для наукової обґрунтованості необхідним є підтвердження статистичної значущості, тобто впевненості, що різниця не могла виникнути випадково. Саме тому було використано критерій Стьюдента для незалежних вибірок. У ході розрахунку було визначено емпіричне значення t, яке порівнювалося з табличним значенням критичного рівня t для ступенів свободи, відповідних обсягам вибірок, і заданого рівня значущості $p < 0,05$. Вибір саме цього рівня значущості є загальноприйнятим у педагогічних і психологічних дослідженнях, оскільки він гарантує, що ймовірність помилки першого роду (визначення впливу там, де його немає) не перевищує 5%.

Розраховане емпіричне значення t виявилося суттєво вищим за табличне, що однозначно свідчить про статистично підтверджену різницю між результатами контрольної та експериментальної груп. Це означає, що позитивний ефект впровадження Unity не є випадковим, а має об’єктивно зафіксований характер. За методологією педагогічних вимірювань така різниця дозволяє говорити про високу ефективність застосованого підходу до навчання [21].

Додатково було проаналізовано коефіцієнт ефекту (Cohen’s d), який показує силу впливу незалежної змінної – у нашому випадку використання Unity як засобу викладання. Отримане значення d перебувало у зоні сильного ефекту, що означає, що різниця між групами має не просто статистичне, а й

практичне значення. Іншими словами, учні експериментальної групи не просто показали вищі бали – їхній рівень знань реально й суттєво відрізнявся від показників контрольної групи.

Важливо також зазначити, що під час аналізу було враховано можливі супутні чинники, такі як: відвідуваність занять, рівень попередньої підготовки та активність під час практичної роботи. Проте навіть з урахуванням цих факторів різниця між групами залишалася значущою. Це дозволяє з високим ступенем впевненості стверджувати, що ключовим елементом, який забезпечив покращення результатів, було саме використання Unity, що збільшило мотивацію, підвищило рівень практичної діяльності та забезпечило кращий візуальний зворотний зв’язок.

Якщо розглядати дані в контексті загальної педагогічної динаміки, то варто відзначити, що учні експериментальної групи (ЕГ) продемонстрували не лише вищі результати у фінальному тестуванні, але й стійке зростання внутрішньої навчальної мотивації, покращення навичок самостійної роботи з кодом та зменшення страху перед складними завданнями. Крім того, у процесі спостереження було зафіксовано збільшення числа добровільних експериментів з кодом, додаткових творчих доробок та спроб самостійного вдосконалення проєктів навіть поза межами основних завдань.

У сукупності ці показники переконливо засвідчують, що різниця між групами має як статистичну, так і педагогічно-практичну значущість. На основі цього можна зробити висновок, що використання Unity у навчальному процесі створює сприятливі умови для формування глибшого розуміння програмування, активізує мислення та сприяє системному зростанню успішності учнів.

Таким чином, результати дослідження мають повне емпіричне підтвердження й демонструють, що Unity є ефективним дидактичним інструментом для вивчення основ алгоритмізації та програмування.

Окрім тестів і спостережень, викладачі відзначали якісні зміни у поведінці студентів. Якщо спочатку більшість ставилася до програмування як до складної та абстрактної дисципліни, то після кількох тижнів роботи з Unity з’явився інтерес, а у частини учнів – навіть захоплення. Рівень відвідуваності занять зріс, а кількість невиконаних домашніх завдань зменшилася більш ніж удвічі.

Під час заключного опитування понад 70% здобувачів освіти вказали, що завдяки візуалізації результатів у Unity вони почали краще розуміти логіку програмування. 64% з них зазначили, що планують продовжувати навчання у сфері розробки ігор або програмного забезпечення, що свідчить про зростання довгострокової мотивації до ІТ-напряму.

Отримані результати підтверджують ефективність розробленої методики оцінювання знань під час використання Unity у навчанні програмуванню. Поєднання традиційних методів тестування з практичними завданнями у візуальному середовищі забезпечує більш повну картину навчальних досягнень і розвитку компетентностей.

Використання Unity дозволяє не лише перевірити знання, але й сприяє розвитку критичного мислення, самостійності та творчого підходу до розв’язання проблем. Важливим чинником підвищення ефективності є емоційне залучення студентів: інтерактивність і можливість бачити результат своєї роботи в реальному часі робить процес навчання більш осмисленим і привабливим.

Таким чином, розроблена методика оцінки ефективності доводить, що впровадження Unity в освітній процес сприяє значному покращенню результатів навчання, підвищенню рівня мотивації та розвитку цілісного бачення програмування як творчого й дослідницького процесу.

3.2 Вплив візуалізації на розуміння програмування

Однією з основних причин труднощів у навчанні програмуванню є його абстрактність. Для більшості учнів процес написання коду здається відірваним від практичних результатів, тому візуалізація виступає ефективним інструментом, що зменшує розрив між теорією та практикою. Використання середовища Unity дозволяє поєднати логіку програмування з безпосереднім візуальним результатом, завдяки чому учні бачать наслідки своїх дій у реальному часі. Такий підхід формує глибше розуміння принципів роботи програм, структури коду та взаємозв'язків між його елементами [2].

Unity як середовище навчання забезпечує інтерактивність і зворотний зв'язок. Учні не лише пишуть код, а й спостерігають, як створені об'єкти реагують на дії, змінюють положення, колір, форму або швидкість. Цей підхід наближує процес навчання до природного експерименту, де кожна дія має безпосередній наслідок. Подібна динаміка сприяє розвитку причинно-наслідкового мислення, що є однією з найважливіших компетентностей у програмуванні [21].

Порівняння традиційного навчання та навчання через Unity. Порівняння традиційних методів навчання програмуванню з навчанням на основі Unity демонструє суттєві відмінності не лише у сприйнятті матеріалу, а й у тому, як формується розуміння логіки програмування, мотивація та практичні навички учнів. Традиційна модель, яка передбачає роботу у текстових середовищах, таких як Visual Studio або PascalABC, здебільшого фокусується на написанні коду та виведенні результатів у консоль. Учень вводить змінні, обчислює значення, отримує текстовий результат, але мало бачить реального впливу своїх дій. Процес залишається абстрактним і відірваним від того, як програмування працює у сучасних цифрових продуктах – іграх, мобільних додатках, інтерактивних системах. Це нерідко створює відчуття, що навчання є рутинним виконанням інструкцій, а програмування – «сухою технічною діяльністю", у якій немає творчого компоненту. Через це

частина учнів сприймає такі заняття як складні або нудні, що призводить до зниження інтересу вже на початкових етапах.

Використання Unity докорінно змінює підхід до викладання, адже відразу вводить учня у середовище, яке максимально наближене до реальних умов розробки. Замість абстрактного тексту на екрані учні бачать тривимірну або двовимірну сцену, у якій об’єкти реагують на їхні команди. Один рядок коду може змусити куб рухатись, персонажа – стрибати, об’єкт – змінювати колір чи обертатися. Відповідно, програмування перестає бути лише логічною вправою й стає процесом створення, у якому навіть найпростіший сценарій перетворюється на частину маленького ігрового світу. Такий підхід формує відчутний емоційний зв’язок між учнем та результатом його роботи: дитина або студент відразу спостерігає наслідок своєї дії, що викликає почуття успіху й стимулює до подальших експериментів.

Під час прикінцевого дослідження, проведеного в рамках педагогічного експерименту, було виявлено, що учні, які працювали в Unity, значно краще засвоювали логіку програмування. Зокрема, під час тестування рівня розуміння умов, змінних, циклів та принципів об’єктно-орієнтованого підходу, результати експериментальної групи перевищували показники контрольної на 25–30%. Це свідчить про те, що візуальне середовище не лише підсилює розуміння концепцій, а й допомагає утримувати увагу та підтримувати інтерес до роботи над завданнями. Більше того, 78% учнів експериментальної групи відзначили, що навчання з використанням Unity стало для них не просто зрозумілішим, а й захопливішим. Вони наголошували, що змогли «побачити” свій код у дії, а саме це, на їхню думку, робило навчання осмисленим. Важливо, що 66% респондентів заявили про бажання продовжувати вивчення програмування вже самостійно після завершення курсу, тоді як у контрольній групі це число було приблизно вдвічі меншим.

Одним із найбільш показових аспектів цього порівняння стало формування міждисциплінарного мислення, яке виявилося набагато

сильнішим у групі, що працювала з Unity. У традиційному навчанні зв'язок між предметами, такими як математика, фізика та інформатика, зазвичай подається теоретично або оновлюється лише у вигляді формальних прикладів. Натомість Unity дозволяє побачити ці взаємозв'язки в дії. Коли учень створює рух об'єкта, він мимоволі стикається з поняттям векторів, координатних систем, швидкості та прискорення. Коли об'єкти взаємодіють через Rigidbody або Collider, у гру вступають фізичні закони – сила тяжіння, тертя, імпульс. Учні, самі того не помічаючи, застосовують знання з інших предметів у практичній діяльності. Це формує природне розуміння принципів, а не механічне запам'ятовування формул, що підтверджено сучасними педагогічними дослідженнями [19].

Дослідження також показало, що учні, які навчалися через Unity, значно швидше починали демонструвати самостійність і ініціативність у роботі. Вони охоче створювали власні сцени, експериментували з параметрами, вигадували нові анімації або модифікували логіку руху. У традиційному навчанні такі прояви зустрічалися значно рідше, оскільки текстові завдання зазвичай мають чітко визначену структуру й не передбачають творчих рішень. Unity, навпаки, заохочує учня до експериментів, адже будь-яка зміна – це миттєвий, яскравий результат у сцені. Нерідко учні прагнули продовжувати роботу над своїми проєктами й після уроків, розвиваючи їх і додаючи нові функції. Це свідчить про формування довготривалої внутрішньої мотивації, що є надзвичайно важливим у навчанні програмуванню.

Психологічний аспект також відіграє суттєву роль. Традиційний текстовий вивід у консоль практично не створює емоційного ефекту, тоді як Unity забезпечує елемент «вау-ефекту», коли навіть незначна зміна коду створює візуально помітну дію. Учні часто описували свої перші успішні кроки словами «вийшло!", «запрацювало!", «він рухається!", і саме цей емоційний підйом сприяв подальшому наполегливому вивченню матеріалу. Знижується страх перед помилками: якщо програма не працює, учень бачить,

що об’єкт не рухається або не реагує – і це сприймається не як невдача, а як частина експерименту. У текстових середовищах помилки часто виглядають як складні повідомлення англійською мовою, що може лякати недосвідчених учнів, тоді як Unity перетворює помилки на навчальний інструмент, роблячи їх частиною інтерактивного процесу.

Значущим результатом стало також те, що навчання через Unity підвищило рівень співпраці між учнями. У традиційних умовах програмування нерідко сприймається як індивідуальна діяльність, у якій кожен працює над своїм кодом самостійно. Натомість Unity заохочує командну роботу: хтось відповідає за моделі, хтось за логіку, хтось за звукове оформлення. Це створює атмосферу взаємопідтримки та спільного рішення задач, що також є важливою частиною цифрової компетентності.

Отже, порівняння двох підходів переконливо показує, що Unity не лише покращує засвоєння матеріалу, а й відкриває новий вимір навчання програмуванню – емоційний, творчий, практичний та міждисциплінарний. Це робить платформу ефективним інструментом модернізації навчання та розвитку ключових компетентностей, необхідних для сучасного цифрового суспільства.

Висновки щодо зміни мотивації та зацікавленості. Результати дослідження свідчать, що впровадження візуалізації навчального процесу через Unity істотно підвищує рівень мотивації, зацікавленості й емоційного залучення учнів. Здобувачі освіти почали сприймати програмування як живий, динамічний процес, де кожна дія має видимий наслідок. Це перетворює навчання з пасивного запам’ятовування правил у дослідницьку діяльність, яка викликає інтерес і задоволення від досягнення результату.

Учителі відзначили, що учні стали активніше співпрацювати між собою, обговорювали рішення, ділилися ідеями. Зросла кількість спільних проєктів, де учні не лише писали код, а й проектували ігрову логіку, налаштовували

сцену, тестували поведінку персонажів. Така діяльність формує не лише технічні навички, а й комунікативні та соціальні компетентності.

Загальний аналіз показав, що використання Unity як візуального середовища призводить до якісних змін у навчанні. Покращується розуміння алгоритмічних принципів, по-друге, підвищується емоційна залученість, а по-третє, формується довгострокова позитивна мотивація до навчання. Це підтверджує висновки зарубіжних досліджень, згідно з якими візуальні середовища сприяють активному навчанню через експерименти та взаємодію [21].

Отже, візуалізація у процесі навчання програмуванню на базі Unity сприяє не лише засвоєнню знань, а й розвитку аналітичного, логічного та творчого мислення. Вона підвищує мотивацію, формує інтерес до подальшого навчання й допомагає учням сприймати програмування як засіб пізнання, експерименту та самореалізації.

3.3 Рекомендації для впровадження Unity в навчальний процес

Використання Unity у процесі викладання програмування відкриває широкі можливості для модернізації освіти, однак ефективність цього підходу значною мірою залежить від того, як саме технологію буде інтегровано в навчальний процес. Щоб Unity стала не просто додатковим інструментом, а повноцінною дидактичною системою, необхідно дотримуватись певних педагогічних принципів, поетапного впровадження та методичного супроводу [14].

Середовище Unity здатне задовольнити потреби як початківців, так і досвідчених користувачів. Воно підтримує логічний розвиток від візуальних уявлень про об’єкти до складних структур програмування. Водночас без належної підготовки викладачів та адаптації навчальних програм існує ризик, що потенціал платформи буде використано частково. Тому рекомендації щодо

впровадження мають охоплювати як педагогічні, так і організаційно-технічні аспекти.

Практичні поради для вчителів. Передусім варто усвідомити, що Unity – це не просто програмне забезпечення, а інтерактивне освітнє середовище, яке дозволяє поєднати абстрактне мислення з конкретним досвідом. Основним завданням учителя є не лише ознайомлення учнів із технічними можливостями Unity, а й побудова навчального процесу так, щоб кожне завдання мало педагогічну мету.

1. **Починати з візуально простих завдань.** На першому етапі слід уникати надлишкової складності. Доцільно використовувати базові сцени: рух куба, взаємодія з платформою, зміна кольору або відтворення звуку після події. Це допомагає учням зрозуміти принцип «код → дія → результат», що є основою логічного мислення у програмуванні.
2. **Поступово ускладнювати навчальні завдання.** Після засвоєння базових принципів можна переходити до створення простих ігрових механік – наприклад, колекціонування предметів, керування персонажем, робота з фізикою. Такі завдання забезпечують не лише закріплення знань із C#, а й розвиток алгоритмічного мислення.
3. **Використовувати Unity як інструмент проектного навчання.** Учень можуть працювати над колективними проектами, що передбачають розподіл ролей (програміст, дизайнер, тестувальник). Це формує командну взаємодію, вчить планувати роботу, оцінювати результат і презентувати готовий продукт [10].
4. **Інтегрувати Unity у міждисциплінарні завдання.** *Наприклад,* створення фізичних симуляцій, навчальних анімацій або інтерактивних демонстрацій законів механіки чи геометрії. Такий підхід показує зв’язок між програмуванням та іншими науками, розвиває міжпредметну компетентність і творчість.

5. **Забезпечити поступовий перехід до написання власного коду.** Спочатку учні можуть користуватися прикладами, але поступово повинні навчитися створювати скрипти самостійно. Для цього ефективно застосовувати метод «від часткового до загального»: спочатку змінювати параметри готових скриптів, потім писати короткі функції, а згодом – повноцінні об’єкти (класи програмування).

Вчитель повинен виконувати роль наставника, який допомагає знайти логічні зв’язки між візуальним результатом і програмним кодом. Як зазначав Сеймур Пейперт, навчання через створення й дослідження власних проектів сприяє глибшому засвоєнню матеріалу та формує стійку пізнавальну мотивацію [2].

Організаційно-технічні аспекти впровадження. Ефективна інтеграція Unity у навчальний процес потребує певних технічних умов і методичної підтримки.

По-перше, важливо забезпечити належну апаратну базу – комп’ютери з сучасними графічними процесорами, достатнім обсягом оперативної пам’яті та доступом до мережі. Проте, як показує досвід, для навчальних цілей достатньо навіть середніх характеристик, якщо грамотно організувати роботу групами або чергуванням завдань.

По-друге, викладач має підготувати навчальні матеріали з поетапним підвищенням складності. Заняття повинні мати чітку структуру: теоретична частина (10–15 хвилин), демонстрація (10 хвилин), практична робота (30–40 хвилин) та обговорення результатів (10–15 хвилин). Така структура дозволяє підтримувати динаміку й увагу студентів.

По-третє, варто створити систему оцінювання, яка враховує не лише правильність виконання коду, а й оригінальність, стабільність роботи програми, оптимальність алгоритмів та якість візуальної реалізації. Це допомагає мотивувати учнів до самостійного пошуку рішень.

Особливу роль відіграє **Unity Learn** – офіційна навчальна платформа, яка надає безкоштовні курси, проекти та матеріали для викладачів. Використання цих ресурсів значно спрощує підготовку занять, дозволяє створювати власні курси або адаптувати готові під потреби конкретної групи [15].

Потенційні проблеми та шляхи їх подолання. У процесі інтеграції Unity у навчальний процес неминуче виникають певні труднощі, які обумовлені як специфікою цього середовища, так і особливостями шкільної та університетської системи навчання. Попри те, що Unity відкриває унікальні можливості для візуального та практично орієнтованого навчання програмуванню, перші етапи його впровадження супроводжуються низкою викликів, які помітно впливають на успішність студентів і загальну динаміку освітнього процесу. Однією з найпоширеніших проблем є технічна складність самої платформи. Для учнів, які тільки знайомляться з програмуванням і технологіями, інтерфейс Unity може видаватися надто насиченим, перевантаженим вікнами та панелями, кожна з яких має власне призначення. Учні нерідко плутаються між такими елементами, як Hierarchy, Inspector, Scene View і Project, що на перших заняттях створює відчуття хаосу та непевності. У деяких випадках складність інтерфейсу може навіть знизити початкову мотивацію, якщо учень не розуміє, які дії потрібно виконувати, або якщо він випадково змінює параметри, що впливають на сцену. Саме тому для подолання цієї проблеми надзвичайно важливо застосовувати поетапне ознайомлення з інтерфейсом, використовуючи демонстрації та підготовлені проєкти-»шаблони», де учні працюють лише з мінімальною кількістю необхідних елементів. Досвід показує, що короткі відеоінструкції, записані викладачем або взяті з офіційних матеріалів Unity Education [22], значно пришвидшують засвоєння, оскільки учні можуть повертатися до них у зручний час і повторювати окремі етапи стільки, скільки потрібно.

Ще однією суттєвою проблемою є обмежений час, доступний у рамках академічної програми. Типовий курс інформатики або програмування в школі передбачає одну-дві години на тиждень, і цього часто недостатньо для роботи з такими інструментами, як Unity, що вимагає тривалих практичних занять. Учні нерідко лише встигають зануритися в задачу, як урок уже завершується, і на наступному занятті потрібно повертатися до попереднього матеріалу, витрачаючи час на повторення. Через це навчання фрагментується, а практичні навички формуються повільніше. Часто буває, що учні потребують безперервних 30–40 хвилин роботи над проєктом, щоб повністю зосередитись на виконанні завдання. В умовах стандартного уроку така можливість з’являється не завжди. Одним із найефективніших способів подолання цієї проблеми є організація факультативів, студійних занять або проєктних тижнів, під час яких учні можуть працювати над своїми ігровими сценами без обмеження в часі. У деяких школах успішно застосовується модель «гуртків цифрової творчості», де учні опановують Unity поза межами основного розкладу, а результати цих занять інтегруються у загальний освітній процес. Практика показує, що саме така гнучкість дозволяє компенсувати часові обмеження стандартної програми та водночас створити комфортні умови для розвитку творчості та самостійності.

Особливої уваги потребує проблема недостатньої підготовки викладачів. Освоєння Unity вимагає не лише знань у сфері програмування, а й розуміння принципів роботи ігрового рушія, базової 3D- або 2D-графіки, систем анімації, фізичного моделювання тощо. Нерідко трапляється так, що викладачі інформатики мають добру теоретичну підготовку, але не знайомі з практичними інструментами розробки ігор, оскільки їхня університетська підготовка була орієнтована переважно на алгоритмічні чи офісні IT-компоненти. У таких випадках саме вчитель стає тим, хто відчуває невпевненість при роботі з Unity, а це неминуче впливає на якість викладання. Проте цю проблему можна вирішити шляхом організації підвищення

кваліфікації: короткі курси, тренінги, вебінари та відеоуроки, створені як ентузіастами, так і офіційною командою Unity Education, забезпечують швидку і доступну підготовку педагогів. Практика доводить, що після 20–30 годин занять викладачі починають комфортно орієнтуватися в середовищі та навіть створювати власні міні-проєкти, які використовують на уроках. Це не лише підвищує рівень компетентності вчителя, а й формує його професійну впевненість, що є важливим чинником успішної інтеграції технологій у навчання.

Певні труднощі виникають і на рівні оцінювання навчальних результатів. Візуальні проєкти, створені в Unity, мають комплексний характер, адже містять код, логіку сцени, елементи анімації, художні матеріали й навіть звуковий супровід. Оцінювати такі роботи за тією ж шкалою, що й звичайні вправи з програмування або контрольні роботи, часто некоректно. Існує ризик, що учень, який створив технічно складний проект, отримає таку ж оцінку, як і учень, який виконав мінімальні вимоги. Крім того, візуальна складова може вводити в оману: проєкт може виглядати ефектно, але містити мало коду; або навпаки – мати багато складної логіки, але виглядати доволі просто. Саме тому доцільним є застосування багатокомпонентної системи оцінювання, яка враховує різні аспекти роботи: технічну реалізацію, креативність, самостійність, командну взаємодію та здатність пояснити власний код. Найефективнішим підходом є формувальне оцінювання, коли учень отримує не лише бал, а й розгорнутий коментар щодо сильних сторін проєкту та можливих шляхів удосконалення. Це відповідає сучасним педагогічним практикам і сприяє розвиткові рефлексії та самоконтролю.

Слід також враховувати інфраструктурні труднощі. Не всі навчальні заклади мають комп’ютери, здатні стабільно працювати з Unity, адже рушій вимагає достатньої кількості оперативної пам’яті та сучасних графічних можливостей. У деяких школах через це неможливо використовувати Unity повною мірою: сцени завантажуються повільно, а запуск гри відбувається з

помітними затримками, що негативно впливає на мотивацію учнів. Частковим вирішенням цієї проблеми може бути використання більш легких 2D-проєктів, оптимізованих сцен, а також застосування хмарних платформ, які дозволяють запускати Unity на слабких комп'ютерах через віддалений доступ. Однак довгострокове рішення – інвестування у технічне оновлення комп'ютерних класів, що потребує підтримки адміністрації та органів місцевого самоврядування.

Успішне подолання всіх зазначених труднощів можливе лише за умов системної роботи на рівні навчального закладу. Важливо, щоб упровадження Unity не розглядалося як тимчасовий або факультативний експеримент, а стало частиною довгострокової стратегії розвитку цифрової освіти. Це включає адміністративну підтримку, розробку методичних рекомендацій, організацію підготовки вчителів та створення часових і технічних умов для реалізації навчальних проєктів. Лише комплексний підхід дозволяє перетворити Unity на повноцінний інструмент розвитку цифрової грамотності, критичного мислення й творчого потенціалу учнів, забезпечуючи високий рівень якості освіти та її відповідність сучасним вимогам.

Перспективи розвитку використання Unity у навчанні. Подальше поширення Unity у сфері освіти має значний потенціал.

По-перше, це відкриває можливість створення навчальних симуляцій, що моделюють реальні процеси – від фізичних явищ до історичних реконструкцій. Такий підхід відповідає сучасним тенденціям STEM-освіти, де навчання відбувається через дослідження, експеримент і проєктування.

По-друге, Unity може бути використана як платформа для дистанційного навчання. Завдяки можливостям експорту проектів у вебформат, Здобувачі освіти можуть виконувати завдання вдома, а викладачі – перевіряти результати через систему зворотного зв'язку. Це особливо актуально в умовах змішаного або дистанційного формату навчання.

По-третє, зростає інтерес до поєднання Unity з технологіями доповненої та віртуальної реальності. Такі інструменти дозволяють створювати повноцінні інтерактивні освітні простори, де учні взаємодіють із цифровими об’єктами та вчаться через занурення. Дослідження Unity Technologies підтверджують, що використання VR/AR підвищує ефективність навчання на 30–40%, а рівень запам’ятовування інформації – майже вдвічі [22].

Важливим напрямом є також формування спільнот викладачів і студентів, які обмінюються досвідом, проектами та методичними матеріалами. Це дозволяє створити відкриту освітню екосистему, що розвивається завдяки взаємодії й співпраці.

Впровадження Unity у навчальний процес сприяє глибокому оновленню методик викладання програмування, наближаючи їх до сучасних вимог цифрової освіти. Unity поєднує у собі наочність, інтерактивність, дослідницький підхід і креативність, забезпечуючи всебічний розвиток учнів.

Для досягнення найвищої ефективності необхідно розглядати Unity не лише як інструмент візуалізації, а як педагогічну платформу, що розвиває компетентності XXI століття – аналітичне мислення, самостійність, співпрацю, уміння вирішувати проблеми.

Отже, системне впровадження Unity у навчальні програми, підкріплене підготовкою викладачів і методичними рекомендаціями, може стати одним із найперспективніших напрямів розвитку цифрової освіти в Україні. Такий підхід забезпечить не лише підвищення якості знань, а й створить умови для формування нового покоління учнів, які мислять креативно, технологічно та стратегічно.

Висновки до розділу 3

У третьому розділі було проведено оцінку ефективності використання Unity як засобу візуалізації алгоритмів та концепцій програмування у навчанні. На основі результатів педагогічного експерименту, анкетування та аналізу динаміки навчальних досягнень встановлено суттєвий позитивний вплив запропонованої методики на розуміння учнями матеріалу, їхню мотивацію та якість практичних навичок.

Було проаналізовано організацію та перебіг експериментального дослідження, у якому взяли участь учні старших класів та Здобувачі освіти . Учасники експериментальної групи навчалися за методикою, що передбачала активну візуалізацію алгоритмів у Unity: демонстрацію змінних, умов, циклів, функцій та об’єктно-орієнтованих структур у реальному часі, створення інтерактивних сцен та навчальних міні-проєктів. Контрольна група опановувала матеріал традиційними засобами, без використання ігрового рушія.

Порівняльний аналіз результатів продемонстрував, що учасники експериментальної групи показали помітно вищу успішність у таких аспектах:

– розуміння алгоритмічних конструкцій, зокрема умовних операторів, циклів і функцій;

– здатність застосовувати знання на практиці, створюючи власні інтерактивні сцени й скрипти;

– мотивація та навчальна залученість, що проявилася у збільшенні кількості самостійних ініціатив, бажанні експериментувати й удосконалювати проєкти;

– розвиток критичного та логічного мислення, що підтверджено результатами практичних завдань і спостереженнями вчителів.

Аналіз анкет показав, що учні експериментальної групи значно частіше відзначали інтерес до програмування, відчуття успіху, легкість у розумінні матеріалу та бажання продовжувати вивчення C#. Важливим результатом стало й те, що використання Unity посилило міжучнівську взаємодію: багато

завдань вони виконували в командах, обговорювали рішення, разом шукали помилки та оптимізували код.

Усі отримані кількісні та якісні показники свідчать про те, що застосування Unity для візуалізації алгоритмів значно підвищує ефективність навчання програмуванню. Розроблена методика демонструє стійкий позитивний вплив на рівень компетентностей учнів, включно з когнітивними, практичними та мотиваційними компонентами.

Отримані результати підтверджують доцільність і ефективність інтеграції Unity у навчальний процес та обґрунтовують рекомендації щодо впровадження цього інструменту у шкільну та університетську практику.

ВИСНОВКИ

У кваліфікаційній роботі було здійснено науково-теоретичне обґрунтування та експериментальна перевірка ефективності використання Unity для візуалізації алгоритмів та концепцій C# у процесі навчання основ програмування.

1. Здійснено аналіз науково-методичної літератури та джерел мережі Інтернет з проблем візуалізації та інтерактивних методів навчання програмуванню.

Проведено науково-теоретичне дослідження особливостей навчання програмуванню в умовах цифровізації освіти. У межах роботи окреслено історичні етапи становлення методик викладання програмування, визначено недоліки традиційних підходів та обґрунтовано переваги інтерактивних середовищ, таких як Unity. Показано, що візуалізація алгоритмів та ігрові механіки підвищують розуміння абстрактних понять, сприяють розвитку логічного мислення та формують стійку навчальну мотивацію.

2. Розроблено методику використання Unity для візуалізації алгоритмічних конструкцій та концепцій C#. Запропоновано структурований комплекс навчальних прикладів, що включає візуалізацію змінних, умов, циклів, функцій, а також базових принципів об’єктно-орієнтованого програмування. Методика передбачає створення інтерактивних навчальних сцен, використання гейміфікації, поступове ускладнення завдань та опору на практичну діяльність учнів. Показано, що Unity забезпечує миттєвий зворотний зв’язок, що істотно покращує якість засвоєння матеріалу.

3. Експериментально перевірено ефективність розробленої методики використання Unity у навчанні програмуванню. Порівняльний аналіз результатів контрольної та експериментальної груп продемонстрував суттєве підвищення навчальних досягнень учнів, які працювали з використанням Unity. В експериментальній групі спостерігалося

краще розуміння логіки програм, здатність самостійно будувати алгоритми, вищий рівень мотивації, активності та творчої ініціативи. Результати підтверджено обробкою даних методами математичної статистики, що засвідчило достовірність покращення ($p < 0,05$).

4. Розроблено практичні рекомендації щодо впровадження Unity у навчальний процес.

Сформульовано рекомендації для вчителів та викладачів щодо організації занять, добору навчального матеріалу, оцінювання результатів та використання візуалізації для спрощення складних алгоритмів. Запропонована методика може бути впроваджена в шкільні та університетські курси інформатики, а також у систему підвищення кваліфікації педагогів.

Отже, проведене дослідження демонструє, що використання Unity як інструменту візуалізації програмування позитивно впливає на якість навчання учнів і студентів. Робота з Unity не лише підвищує розуміння алгоритмів та структур коду, але й формує практичні навички, розвиває логічне та творче мислення, сприяє командній діяльності та підвищує інтерес до програмування. Застосування Unity забезпечує реалізацію сучасних освітніх вимог та створює умови для гармонійного розвитку цифрових компетентностей здобувачів освіти.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Березняк В. М., Морзе Н. В. Інформаційно-комунікаційні технології у навчальному процесі. Київ : Освіта України, 2012. 276 с.

2. Мартін Роберт Сесіл. Чиста архітектура : мистецтво розробки програмного забезпечення / пер. з англ. Київ : Фабула, 2021. 432 с.

3. Морзе Н. В., Барна О. М. Інноваційні методи навчання у вищій школі. Київ : Університет «Україна», 2018. 320 с.

4. Морзе Н. В., Кузьмінська О. Г. Інноваційні педагогічні технології навчання: теорія та практика. Київ : Університет «Україна», 2019. 348 с.

5. Нікітченко М. О. Інтерактивні технології навчання у закладах вищої освіти. Інформаційні технології і засоби навчання. 2019. № 2(70). С. 145–159. URL: https://journal.iitta.gov.ua/index.php/itlt/article/view/2575 (дата звернення: 25.03.2025 р.).

6. Семеріков С. О., Триус Ю. В. Проєктно-орієнтований підхід у підготовці ІТ-фахівців. Інформаційні технології і засоби навчання. 2020. № 1(75). С. 78–94. URL: https://journal.iitta.gov.ua/index.php/itlt/article/view/2987 (дата звернення: 25.03.2025 р.).

7. Сухомлинський В. О. Серце віддаю дітям. Київ : Радянська школа, 1977. 368 с.

8. Чорна Л. В. Гейміфікація у вищій школі: переваги, ризики, досвід впровадження. Освітологічний дискурс. 2021. № 3–4. С. 112–127. URL: https://od.kubg.edu.ua/index.php/journal/article/view/789 (дата звернення: 25.03.2025 р.).

9. Adeniji I., Casarona M., Bielory L. et al. Using Unity for Scientific Visualization as a Course-based Undergraduate Research Experience. The Journal of Computational Science Education. 2024. Vol. 15, No. 1. P. 35–40. URL: https://www.cs.uresearch.org/2024/vol15/adeniji.pdf (дата звернення: 23.03.2025 р.).

10. Akcaoglu M., Dogan S., Hodges C. B. Real Coding and Real Games: Design and Development of a Middle School Curriculum Using Unity 3D. TechTrends. 2022. Vol. 66. P. 931–937. DOI: 10.1007/s11528-022-00780-6. URL: https://link.springer.com/article/10.1007/s11528-022-00780-6 (дата звернення: 25.03.2025 р.).

11. Computer History Museum. The BBC Micro. URL: https://computerhistory.org/blog/the-bbc-micro/

12. De Freitas S., Oliver M. How can exploratory learning with games and simulations within the curriculum be most effectively evaluated? Computers & Education. 2006. Vol. 46, No. 3. P. 249–264. DOI: 10.1016/j.compedu.2005.11.007.

13. IBM. What is COBOL? URL: https://www.ibm.com/topics/cobol (дата звернення: 25.03.2025 р.).

14. Landin P. J. The Mechanical Evaluation of Expressions. The Computer Journal. 1964. Vol. 6, No. 4. P. 308–320. URL: https://academic.oup.com/comjnl/article-pdf/6/4/308/1014292/6-4-308.pdf (дата звернення: 25.03.2025 р.).

15. Lindsey C. H. A History of ALGOL 68. URL: https://www.algol68-lang.org/docs/lindsey-history-of-algol68.pdf (дата звернення: 09.03.2025 р.).

16. Papert S. Mindstorms: Children, Computers, and Powerful Ideas. New York : Basic Books, 1980. 230 p.

17. Roane W., Vines N., Petzold B. et al. Using Unity to Support Accessible and Engaging Computer Science Education. SIGCSE 2022: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education. 2022. Vol. 2. P. 1132–1138. DOI: 10.1145/3478432.3499200.

18. Schank R. C. Active learning through visualization and simulation. Journal of Educational Technology Systems. 1994. Vol. 22, No. 3. P. 215–226.

19. Scratch. MIT Media Lab. URL: https://scratch.mit.edu/

20. The Fortran Programming Language. URL: https://fortran-lang.org/

21. The Pascal Programming Language. URL: https://www.freepascal.org/

22. Unity Technologies. Unity Learn: Teaching with Unity. URL: https://learn.unity.com/ (дата звернення: 22.02.2025 р.).

23. Unity Technologies. Using Unity in Education: Research Insights (2024). URL: https://unity.com/resources/education-research-insights (дата звернення: 25.03.2025 р.).

24. Unity Technologies. Unity Manual: Prefabs. URL: https://docs.unity3d.com/Manual/Prefabs.html. (дата звернення: 25.03.2025 р.).

25. Using Unity to teach Game Programming : master thesis. DiVA portal, 2023. 68 p. URL: http://www.diva-portal.org/smash/get/diva2:1867700/FULLTEXT02.pdf (дата звернення: 05.01.2025 р.).

26. Végh L. Teaching and learning computer programming by creating 2D games in Unity. ICERI2021 Proceedings. 2021. P. 1285–1291. DOI: 10.21125/iceri.2021.0355. URL: https://www.researchgate.net/publication/356433791_TEACHING_AND_LEARNING_COMPUTER_PROGRAMMING_BY_CREATING_2D_GAMES_IN_UNITY (дата звернення: 25.03.2025 р.).

ДОДАТКИ