МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОПОВА КАРОЛІНА ОЛЕКСАНДРІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____2026 р.

РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ У ЖАНРІ ГОЛОВОЛОМКИ

Спеціальність 122 Комп'ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Ірина ФРИЗ, старший викладач
кафедри інформаційних технологій,
канд. фіз.-мат. наук
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2026

АНОТАЦІЯ

Попова К.О. Розробка комп'ютерної гри в жанрі головоломки. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано поняття комп'ютерної відеогри-головоломки та її аналогів, виділено основні принципи та способи реалізації. За допомогою технологій Unity, Microsoft Visual Studio, а також мови програмування C#, розроблена 2D гра-головоломка у стилі Sokoban-подібної.

Ключові слова: комп'ютерна гра, гра-головоломка, Unity, Sokoban, 2D-графіка, скрипт C#.

61 ст., 26 рис., 2 дод., 40 джерел.

ABSTRACT

Popova K.O. Development of a Computer Game in the Puzzle Genre. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification (bachelor's) thesis researches and analyzes the concept of a computer puzzle video game and its analogues, highlighting the main principles and methods of implementation. Utilizing Unity, Microsoft Visual Studio, and the C# programming language, a 2D Sokoban-style puzzle game was developed.

Keywords: computer game, puzzle game, Unity, Sokoban, 2D graphics, C# script.

61 p., 26 figs., 2 app., 40 refs.

ЗМІСТ

ВСТУП.....4
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР-ГОЛОВОЛОМОК.....7
1.1 Поняття комп'ютерної гри.....7
1.2 Загальна характеристика гри жанру «Головоломка».....9
1.3 Аналіз ігор-аналогів комп'ютерної гри-головоломки Sokoban.....12
1.4 Інструменти для створення комп'ютерної гри-головоломки.....16
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ-ГОЛОВОЛОМКИ.....20
2.1 Аналіз інструментів для розробки комп'ютерної гри-головоломки.....20
2.2 Вимоги користувачів і аналіз Sokoban-подібної гри-головоломки.....25
2.3 Візуалізація комп'ютерної гри-головоломки.....29
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ В UNITY.....35
3.1 Інтерфейс розробленої комп'ютерної гри-головоломки.....35
3.2 Процесорна частина комп'ютерної гри-головоломки.....43
3.3 Тестування комп'ютерної гри та перспективи її розвитку.....51
ВИСНОВКИ.....53
СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ.....55
ДОДАТКИ.....58

ВСТУП

Наразі відбувається стрімкий розвиток індустрії комп'ютерних ігор найрізноманітних жанрів. Зростає попит користувачів на інтерактивні розважальні продукти, зокрема ігор жанру «Головоломки». Головоломки виконують не лише розважальну функцію – вони сприяють розвитку уваги, логічного мислення, в них часто необхідно роз'язувати задачі різної складності. Різновидів ігор-головоломок дуже багато, але в даній роботі виділено конкретно Sokoban-подібні головоломки, оскільки вони поєднують в собі ефективність простих ігрових механік та глибокий ігровий процес, є унікальними і актуальними і для користувачів, і для розробників. Одночасно з розвитком індустрії комп'ютерних ігор відбувається розвиток ігрових рушіїв, на яких комп'ютерні ігри розробляються (наприклад, Unity – одний з найпопулярніших ігрових рушіїв, який використовується багатьма великими ігровими компаніями). Використання таких інструментів для розробки дуже спрощує процес створення цифрового продукту і це відкриває багато можливостей для дослідження та реалізації своїх проектів навіть для початківців у програмуванні.

Мета дослідження – розробка комп'ютерної гри у жанрі головоломки на прикладі Sokoban-подібної гри, що демонструє базові принципи створення логічних ігор та їх реалізацію з використанням ігрового рушія Unity.

Завдання дослідження:

- проаналізувати аналоги комп'ютерних ігор у жанрі головоломки, виділити основні механіки та особливості;
- сформувати концепцію комп'ютерної гри-головоломки: правила, логіка рівнів, графічний інтерфейс;
- обгрунтувати інструменти для реалізації комп'ютерної гри-головоломки;
- розробити гру з набором базових рівнів з врахуванням логіки руху об'єктів;
- оцінити роботу комп'ютерної гри і визначити її подальші перспективи розвитку.

Об’єкт дослідження – процес розробки та програмної реалізації комп’ютерних ігор у жанрі «Головоломка».

Предмет дослідження – методи, алгоритми та інструментальні засоби проектування ігрової механіки та логічних рівнів у комп’ютерній 2D грі-головоломці.

Теоретичне та практичне значення одержаних результатів полягають у розгорнутому огляді теоретичних та практичних аспектів, пов’язаних з розробкою комп’ютерної гри-головоломки. Зокрема наведений порівняльний аналіз низки методів, програмного забезпечення та інструментів для розробки, принципи їх роботи, їхні складові інструменти. В роботі описано алгоритм створення комп’ютерної 2D гри-головоломки виду Sokoban.

Апробація результатів дослідження: результати дослідження доповідалися на VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології», 15 травня 2026 року. Донецький національний університет імені Василя Стуса, м. Вінниця, тема доповіді: «Розробка комп’ютерної гри у жанрі головоломки».

Структура кваліфікаційної (бакалаврської) роботи складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні основи створення комп’ютерних ігор-головоломок, проведено аналіз існуючих ігор-аналогів, їх механік та особливостей побудови ігрового процесу, досліджено сучасні програмні засоби для розробки комп’ютерних ігор. У другому розділі описано концепцію програмного продукту, архітектуру системи, вимоги до користувачів і технічної складової, організацію даних та особливості побудови логіки гри. У третьому розділі наведено процес реалізації програмного продукту у середовищі Unity, описано інтерфейс гри, програмну логіку, систему взаємодії об’єктів, а також результати тестування розробленої гри. Список використаних джерел містить 40 найменувань. Додатки містять ілюстративні матеріали, фрагменти програмного коду та зображення інтерфейсу гри. Робота містить 26 рисунків, 1 таблицю та 2 додатки.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР-ГОЛОВОЛОМОК

1.1 Поняття комп'ютерної гри

Комп'ютерні ігри є дуже затребуваними, популярними та з одного боку виглядають як проста розвага. Насправді, комп'ютерна гра є складною інтерактивною системою, що поєднує технічні, художні та логічні компоненти для створення віртуального середовища та забезпечення взаємодії користувача з ним [1-4, 19].

Комп'ютерна гра – спеціалізований програмний продукт, призначений для організації взаємодії користувача з віртуальним ігровим середовищем за допомогою комп'ютерної техніки та інших цифрових пристроїв. Основною метою комп'ютерної гри є створення інтерактивного процесу, у якому гравець виконує якісь дії відповідно до встановлених правил для досягнення визначеної мети.

Поява комп'ютерних ігор починається у другій половині XX століття разом з розвитком обчислювальної техніки та програмування. Перші ігри мали дуже просту графіку та обмежені технічні можливості, але з розвитком комп'ютерних технологій ігрова індустрія перетворилася на одну з найбільших сфер цифрових технологій та мультимедійних розваг. Сучасні комп'ютерні ігри поєднують у собі програмування, графічний дизайн, анімацію, моделювання та елементи сценарної розробки. Розвиток ігрових технологій сприяв широкому поширенню комп'ютерних ігор та їх перетворенню на важливу частину сучасної цифрової культури.

Основою будь-якої комп'ютерної гри є ігровий процес або gameplay, який визначає спосіб взаємодії гравця з грою. Ігровий процес включає правила, систему керування, завдання, механіку взаємодії об'єктів та умови перемоги або програшу. Саме механіка гри визначає, які дії може виконувати гравець, яким

чином реагує ігрове середовище та як відбувається розвиток подій під час проходження гри.

Графічний інтерфейс є важливим елементом комп'ютерної гри, який забезпечує візуальне відображення ігрового світу та взаємодію користувача з елементами гри. Залежно від типу гри використовується двовимірна (2D) або тривимірна (3D) графіка. За жанровими особливостями комп'ютерні ігри поділяються на аркади, стратегії, симулятори, рольові ігри, платформери, пригодницькі ігри, головоломки та інші.

Комп'ютерні ігри можуть виконувати не лише розважальну функцію, але й навчальну, пізнавальну та розвивальну, які сприяють розвитку уваги, пам'яті, швидкості реакції, просторового мислення та навичок прийняття рішень. Саме тому зараз ігрові технології активно використовуються у навчанні, моделюванні та професійній підготовці.

Процес створення відеогри на перший погляд цікавий, але насправді складний, який потребує багато ресурсів та спеціалістів [5-11]. Створенням відеоігор займається розробник відеоігор – одна людина або велика компанія з мільйонним бюджетом та сотнями працівників. Фінансуванням гри займаються розробники ігор або його спонсорує видавець відеоігор. Тривалість розробки проекту та його вартість зростає з його складністю. За розроблення повністю готової для публікування і користування відеогри відповідають спеціалісти:

Gamedesigner – спеціаліст, який розробляє концепцію та правила майбутньої відеогри, її механіки та особливості, персонажів, продумує майбутні івенти для неї та бонуси тощо. Програміст – спеціаліст, який займається розробкою кодової бази для відеогри. Програмісти відповідають кожен за той інструмент, який вони окремо розробляють, наприклад, ігровий рушій, gameplay, інтерфейс користувача, ввід даних тощо. Тестер – спеціаліст, який перевіряє якість та знаходить баги у ПЗ перед його випуском на платформи, визначає правильність роботи гри та виявляє помилки, які впливають на роботу відеогри.

Менеджер – головний начальник всієї команди розробників, який розподіляє обов'язки, контролює роботу розробників, планує фінансові витрати

та визначає кінцеві строки виконання завдань. Піар-менеджер займається просуванням та популяризацією гри, створенням рекламу, участю у виставках або презентаціях, де є можливість прорекламувати випуск нової відеогри.

Розробка відеоігор складається з трьох головних етапів: пре-продакшн, продакшн та пост-продакшн. До своєї остаточної версії відеогра проходить через альфа та бета версії, які тестувальники сильно перевіряють. Іноді розробники запускають відкриті або закриті бета-тести для гравців, і за результатами та враженнями від них розробники можуть доповнити щось в відеогру, що гравці попросили, або навіть переробити цілком те, що не сподобалось або піддалось сильної критиці (сюжет, персонажі, локації).

Завершена локалізована відеогра записується на DVD-диски, глобальна або обмежено глобальна відеогра надається для завантаження з Інтернету або таких маркетплейсів як PlayMarket та AppleStore. Глобальна гра може бути повністю безкоштовною, або наполовину (наприклад, для повної версії потрібно купляти Premium або для покупки ігрової валюти використовувати справжню), або платною. Після офіційного релізу відеогра продовжує періодично оновлюватись. Для цього є певні причини: по-перше, кожного року виходять нові моделей персональних комп'ютерів, смартфонів, тому й ігри мають підтримуватись для таких пристроїв; по-друге, після релізу можуть бути знайдені баги гри, поява вразливості безпеки; останнє, розробникам вигідно нарощувати та підтримувати сталу користувацьку базу, тому вони випускають оновлення з додаванням нових ігрових фішок, щоб тримати інтерес гравців до гри.

1.2 Загальна характеристика гри жанру «Головоломка»

Комп'ютерні ігри-головоломки залишаються одним із найпопулярніших жанрів відеоігор протягом тривалого часу. Для них характерні особливості, що визначають специфіку головоломок серед інших жанрів комп'ютерних ігор [12].

Головоломки – це жанр відеоігор, головна ідея яких це розв'язання певних задач, найчастіше на логіку. Під час гри у головоломку гравці мають знайти ключ

до розв'язання головоломки – використовувати виграшну стратегію, інтуїцію або ерудицію, логічно мислити, мати мінімальні наукові знання, уважно стежити за тим, що відбувається в ігровому віртуальному просторі тощо. Якщо такі жанри відеоігор, як екшен або пригоди, здебільшого орієнтуються на рефлекси гравця чи його бойові навички, то головоломки – на розумову діяльність гравців. Важливо те, що до головоломок не відносяться такі ігри, які мають елементи удачі, везіння, дій на швидкість.

Головоломки прості в контексті графіки, процесорного навантаження, деталями в gameplay, але вимагають використання критичного мислення та мінімальних наукових знань для розв'язання. Більшість таких відеоігор мають чіткі правила та цілі, для досягнення яких гравцям потрібно їх дотримуватися і планувати розв'язок, багато думати. Завдяки цьому користувачі дійсно пишаються собою і відчувають себе «переможцями», коли їм вдається розв'язати певну складну задачу.

Відеогра може мати або міні-ігри/елементи головоломки, або вона сама по собі є головоломкою, в усіх її етапах. Рольові ігри (бойовики) часто використовують головоломки в своєму gameplay, щоб розбавити бойову атмосферу або навпаки – підкреслити її, додати додаткової напруженості .

Відеоігри-головоломки підтримуються на всіх можливих платформах – від Android та ігрових приставок до потужного PC, аркадних автоматів, існують браузерні версії відеоігор-головоломок. Вікова категорія ігор жанру «головоломка» дуже широка – від 3+ до 99+. Вважається, що маленьким дітям корисно грати в головоломки, оскільки вони розвивають у них критичне мислення, логіку та увагу. Любителі інших жанрів ігор не відмовляться від розв'язання головоломки під час проходження відеогри іншого жанру. Різноманіття головоломок приваблює найрізноманітніших користувачів, і воно тримає популярність цих ігор десятиліттями.

Відеоігри в жанрі «Головоломка» виділяє дуже різноманітна та яскрава класифікація. Вони можуть мати дуже різні правила, ідею, сюжет (якщо є), графічні елементи та об'єкти, що робить головоломки особливими серед інших

жанрів комп'ютерних ігор. Розглянемо типи класифікацій комп'ютерних ігор-головоломок, чим вони відрізняються один від одного, які є особливості та правила [13, 14].

Мовні/словесні головоломки. У словесних головоломках основним об'єктом взаємодії з користувачем є слова або літери. Такі головоломки спрямовані на розвиток словникового запасу та здатності до аналізу текстової інформації; вони акцентують увагу на лінгвістичних моментах і від гравця потребується вміння працювати з мовою (рідною або іноземною). Під час розв'язання мовних задач гравець повинен аналізувати символи, встановлювати між ними зв'язки, формувати правильні слова або фрази (залежно від правил гри). Сучасні комп'ютерні словесні головоломки поєднують класичні механіки з інтерактивними елементами, наприклад підказки, система рівнів, обмеження часу, бонуси, підвищує зацікавленість гравця. Прикладами мовних головоломок є кросворди, анаграми, «Пошук слів».

Графічні головоломки можна сміливо назвати одним із найпоширених типів ігор на логіку. В них основна увага зосереджується на сприйнятті візуалу та взаємодії з графічними об'єктами – формами, зображеннями, візуальними структурами, для того, щоб розв'язати поставлену задачу. До графічних головоломок належать різні підвиди, зокрема ігри зі складання фігур, пазли, лабіринти; логічні задачі, пов'язані з переміщенням об'єктів у просторі.

Математичні головоломки в основному працюють із числовими задачами, числами, обчисленнями, закономірностями. Такі головоломки розвивають аналітичне мислення і потребують базових математичних знань гравця. Суть математичних головоломок – це їхнє розв'язання шляхом виконання правильних обчислень або логічних операцій. Задачі можуть мати один або більше розв'язків, можливо потрібно бути знайти оптимальний варіант, якщо є така вимога. Також існують задачі на комбінаторику та теорію ймовірностей. Приклади математичних головоломок: судоку, магічні квадрати.

Фізичні головоломки. У фізичних головоломках геймплей базується на моделюванні фізичних явищ і законів – гравітація, інерція, сила, тертя, баланс,

взаємодія об’єктів. Відмінність фізичних головоломок від інших видів головоломок є те, що результат гри залежить від динамічної поведінки елементів середовища гри. Іншими словами, одна й та сама задача має різні способи вирішення, що відрізняється зі строго визначеними правилами та способами виграшу в інших головоломках. Гравець повинен правильно використовувати ігрові об’єкти для досягнення мети, враховувати фізичні закономірності, спрогнозувати ймовірний рух та взаємодію об’єктів. На відміну від інших головоломкок, фізичні головоломки дозволяють експериментувати з грою. У процесі програмування фізичні головоломки є складнішими ніж інші види головоломок, оскільки розробнику треба враховувати фізику об’єктів та правильно спрограмувати її. З’являється необхідність використовувати ігрові рушії для розробки даних типів ігор [15-18].

1.3 Аналіз ігор-аналогів комп’ютерної гри-головоломки Sokoban

Для розробки обрано Sokoban-подібну гру-головоломку. Розглянемо детально її ідею, правила, особливості ігрового процесу, які описані у [19-22].

Sokoban – головоломка, в якій гравець штовхає ящики, щоб доставити їх на місця зберігання. Гру створив японський програміст Хіроюкі Імабаяші у 1981 році. Вона має такі правила:

- ящики можна тільки штовхати, не тягнути, і штовхати тільки один ящик за раз;
- гравець не повинен собі ящиками закрити прохід, поставити себе в глухий кут, інакше рівень гри вважається непройденим або таким, що неможливо пройти.

Ігрове поле головоломки має вигляд зверху, складається зі стін і підлоги, які поділені на квадрати (клітинки). Клітинка на підлозі може бути порожньою або зайнятою гравцем/коробкою. Деякі квадрати на підлозі є місцями зберігання, кількість місць зберігання дорівнює кількості коробок (рис. 1.1). Гравець може переміщатися горизонтально або вертикально по одній клітинці за раз на

порожній квадрат підлоги. Коробки та стіни блокують рух гравця: гравець може підійти до коробки і штовхнути її на порожню клітинку прямо за нею, але якщо коробку притиснути до стіни або іншої коробки, то вона не буде рухатись.

Рисунок 1.1 – Скріншот ігрового процесу головоломки Sokoban

Гра розвиває критичне мислення у гравців та навички планування, є простою, але водночас продуманою та логічною. Sokoban-головоломку можна віднести до категорії фізичних головоломок, оскільки в її gameplay враховується фізичний процес штовхання коробок, відсутність руху при контакту зі стінами або іншими ящиками.

Головоломка набрала високу популярність і має багато ігор аналогів зокрема, Vextorial, Sokowand, які описані у [23]. Дані ігри реалізують класичну механіку Sokoban, засновану на переміщенні коробок у визначені цільові позиції. Основними особливостями таких реалізацій є використання покрокового переміщення персонажа, системи логічних рівнів та поступового ускладнення ігрового процесу. Деякі аналоги також містять власні графічні стилі, додаткові механіки та розширені системи рівнів. Наведемо їх опис нижче.

Vextorial – гра-головоломка, що поєднує в собі фізичні та просторові механіки, елементи класичних головоломок. Гра складніша ніж класичний Sokoban, тому що вона має в своєму ігровому процесі маніпулювання гравітацією, можливість зміни перспективи, перемикання між 2D та 3D

просторами. Такі особливості геймплею дають багато можливостей проходження рівнів та підвищують їхню складність.

MazezaM – головоломка, що поєднує лабіринти та елементи Sokoban. Основна ідея гри полягає у проходженні рівнів, які складаються з заплутаних структур, де гравець повинен правильно взаємодіяти з елементами середовища для досягнення цілі. Тут так само як в Sokoban – є потреба в переміщенні об’єктів, але це ускладнюється лабіринтами.

Box Kid Adventures – динамічна головоломка, яка відрізняється від Sokoban наявністю швидкої реакції на процеси гри та обмеженого часу на проходження головоломки, має в ігровому процесі пастки та ворогів, що також впливає на складність рівнів. Як і Sokoban, дана головоломка має вигляд зверху ігрового поля.

Baba Is You – логічна головоломка, у якій правила гри представлені у вигляді інтерактивних блоків. Основна механіка гри полягає у тому, що гравець може маніпулювати цими блоками, змінюючи логіку взаємодії об’єктів на рівні та самі правила його проходження. Завдяки можливості змінювати правила, гравець може зробити ворога дружнім, змінити властивості об’єктів або створити нові способи досягнення мети, що робить кожен рівень даної головоломки багатоваріантним і динамічним, простиставляючи Sokoban, де правила проходження рівнів є незмінними та статичними.

Гампі та Гемікубе – головоломка в стилі Sokoban, що додатково має механіку гравітації та телепортації. Об’єкти в цій грі мають власну гравітацію, що дозволяє гравцеві ходити по стінах або маніпулювати об’єктами у незвичайних площинах, а при цьому інші елементи залишаються на своїх поверхнях.

Випробування телепортації – головоломка в стилі Sokoban, яка поєднує механіки переміщення об’єктів із додатковими елементами порталів, бомб та танків. Геймплей полягає у комбінуванні цих механік, що створює багаторівневий ігровий процес, де гравець повинен не лише переміщувати об’єкти, а й враховувати взаємодію між різними елементами середовища.

Sokowand включає традиційні завдання Sokoban, такі як штовхання коробок для відкриття дверей, і більш складні інтерактивні механіки, наприклад маніпулювання сигналами, перенаправлення лазерів за допомогою дзеркал, перепроводження електрики для активації цільових механізмів.

Super Box Land Demake – 2D-головоломка, включає штовхання коробок для досягнення цільових позицій, має кооперативний режим, який додає елемент командної взаємодії та спільного планування дій.

Sokobond Express поєднує елементи пошуку шляхів із навчальними та науковими механіками хімічних зв'язків. Метою гравця є створення правильних молекул, для цього переміщуючи атоми та формуючи зв'язки. Гра вимагає логічного мислення та планування послідовності дій.

Отже, можна зробити висновок, що розробники ігор-аналогів додали свої механіки, які ускладнили та зрізноманітнили класичну головоломку Sokoban – наприклад, додали елементи фізики та гравітації, ворогів або пастки, переключення між вимірами та навіть режими спільного проходження гри. Іншими словами – взяли класичний Sokoban як основу для створення своїх комп'ютерних головоломок, при цьому додаючи свої елементи або протиставляючи до елементів, що наявні в класичній головоломці. Водночас такі нововведення можуть виглядати перегруженими та зайвими, тому що сама головоломка Sokoban має головну ідею у здатності планувати свої подальші ходи, без використань порталів або взаємодій з фізикою, тобто головною наявністю саме логіки.

На основі проведеного аналізу існуючих аналогів сформульовано функціональні вимоги до розроблюваного програмного продукту, метою визначення яких є створення збалансованої гри-головоломки, яка усуває надмірну статичність або переускладненість розглянутих систем.

Основними функціональними вимогами до гри є забезпечення чіткого дискретного переміщення ігрового персонажа у двовимірній сітці за допомогою клавіатурного введення та реалізація надійних фізичних колізій взаємодії з об'єктами. Програма повинна суворо контролювати правила штовхання блоків,

унеможливлюючи їх протягування назад або одночасне зміщення кількох коробок. Система автоматичного обліку прогресу має динамічно перераховувати кількість заповнених цільових зон через механізми Trigger та миттєво реагувати на зміну стану ігрового поля. Для вирішення проблеми безвихідних логічних ситуацій, що є критичним недоліком багатьох головоломок, обов'язковою вимогою є впровадження функції миттєвого перезавантаження рівня гарячою клавішею, і паралельно створення навігаційного інтерфейсу для запуску додатка, переходу між рівнями та повернення до головного меню.

1.4 Інструменти для створення комп’ютерної гри-головоломки

Для розробки комп’ютерних ігор-головоломок використовуються доступні програмні засоби, які дозволяють реалізувати логіку гри і її візуальну складову. Такими засобами є ігрові рушії та середовища програмування, які забезпечують повний цикл розробки програмного продукту. Sokoban-подібна комп’ютерна гра-головоломка буде розроблятись за допомогою ігрового рушія **Unity**, мови програмування **C#** та програмного середовища **Microsoft Visual Studio**, оскільки ці інструменти є взаємопов’язаними, популярними та не дуже складними, що допомагає кожному програмісту-початківцю створити гру, не витрачаючи зайвих важких зусиль.

Інші, окрім Unity, ігрові рушії для створення ігор – Godot та Unreal Engine. Ігровий рушій Godot є повністю безкоштовним продуктом із відкритим кодом, вирізняється своєю легкістю та системою побудови сцени на основі вузлів, що робить його зручнішим за Unity для швидкої розробки 2D-ігор, але має меншу базу готових ассетів. Unreal Engine орієнтований на створення високоестетичних AAA-проектів із використанням потужних візуальних інструментів та мови C++, що забезпечує значно вищу графічну продуктивність у порівнянні з Unity, але рушій потребує складного налаштування та потужного апаратного забезпечення. Головна відмінність Unity від цих рушіїв полягає у його універсальності та

використанні мови C#, що дозволяє зберігати баланс між простотою розробки, як у Godot, та широкими можливостями для 3D, як у Unreal Engine (табл. 1.1).

Таблиця 1.1 – Узагальнене порівняння Unity, Godot та Unreal Engine

	Unity	**Unreal Engine**	**Godot**
Мова програмування	C#	C++, Blueprint	GDScript, C#
Переваги	Простий у використанні. Велика кількість навчальних матеріалів. Підтримка 2D та 3D. Велика бібліотека ассетів. Зручний інтерфейс	Потужна графіка. Професійні інструменти. Висока продуктивність	Безкоштовний. Відкритий код. Легкий та швидкий. Зручний для 2D-ігор
Недоліки	Деякі розширені функції потребують додаткового налаштування	Високі системні вимоги та складний для початківців	Менша кількість готових ассетів та навчальних матеріалів
Використання	Підходить для створення 2D та 3D ігор різних жанрів	Частіше використовується для AAA-проєктів та складних 3D-ігор	Ефективний для невеликих 2D-проєктів та інді-розробки

Для реалізації програмної частини гри обрано середовище розробки Microsoft Visual Studio, що забезпечує зручні інструменти для написання, налагодження та тестування коду мовою C#. У порівнянні з іншими середовищами програмування Rider або Visual Studio Code, Microsoft Visual Studio має більш глибоку інтеграцію з ігровим рушієм Unity, що дозволяє

ефективно працювати з проєктами на основі C# та забезпечує зручну взаємодію з компонентами гри. Порівнюючи з VS Code, Microsoft Visual Studio містить розширені засоби налагодження, аналізу коду та підтримку великих проєктів. Visual Studio є більш поширеним і доступним серед початківців-розробників та має широку документацію і підтримку, ніж JetBrains Rider. Також Microsoft Visual Studio має в наявності вбудований інструмент Unity Tools, який забезпечує покрокове налагодження скриптів безпосередньо під час виконання гри в редакторі рушія, що дає змогу відстежувати стан змінних у реальному часі, оперативно виявляти логічні помилки в алгоритмах руху ігрових об’єктів та оптимізувати роботу програми. Наявність інтелектуальної системи підказок IntelliSense суттєво прискорює написання коду завдяки автодоповненню специфічних для Unity класів та методів.

Для реалізації логіки Sokoban-подібної головоломки обрано мову **C#**. C# є об’єктно-орієнтованою мовою з високим рівнем безпеки типів, автоматичним керуванням пам’яттю та відносно низьким порогом входження для розробників у порівнянні з C++. У контексті розробки на Unity, мова C# забезпечує найвищу продуктивність написання скриптів управління, побудови матричних сіток для переміщення ігрових об’єктів та взаємодії з користувацьким інтерфейсом, вона дозволяє гнучко реалізувати алгоритми перевірки умов перемоги та обробки колізій без ризику виникнення критичних помилок витоку пам’яті.

На основі проведеного аналізу предметної області та обраного інструментарію, для подальшої реалізації програмного продукту сформульовано такі завдання:

1. Розробити загальну архітектуру комп’ютерної гри-головоломки та спроектувати схему взаємодії основних програмних компонентів (систем переміщення, обліку цілей та керування станами гри).
2. Створити та налаштувати 2D-сцену в середовищі Unity, включаючи налаштування компонентів фізичної взаємодії (Collider2D, Rigidbody2D) та шарів колізій для ігрових об’єктів.

3. Програмно реалізувати алгоритм дискретного покрокового руху гравця та логіку штовхання коробок із перевіркою перешкод за допомогою методів фізичного перекриття.
4. Розробити централізований менеджер гри (GameManager) для динамічного відстеження кількості активованих цілей, реалізації ігрового таймера зворотного відліку та виведення графічного інтерфейсу перемоги чи програшу.
5. Перевірити роботу розробленого програмного забезпечення на предмет коректності роботи логіки головоломки.

Отже, у першому розділі здійснено детальний аналіз жанру комп'ютерних ігор-головоломок та досліджено специфіку побудови Sokoban-подібних ігор, визначено ключові механіки гри, які базуються на дискретному переміщенні об'єктів у замкненому просторі за певними правилами, проведено порівняльний аналіз сучасних ігрових рушіїв (Unity, Godot, Unreal Engine) та середовищ програмування (Visual Studio, VS Code, Rider). Обґрунтовано вибір ігрового рушія Unity, мови програмування C# та середовища розробки Microsoft Visual Studio як найбільш оптимального, універсального та ефективного інструментарію для реалізації поставлених завдань, що закладає основу для практичного проектування та розробки програмного продукту на наступних етапах дослідження.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ-ГОЛОВОЛОМКИ

2.1 Аналіз інструментів для розробки комп'ютерної гри-головоломки

У підрозділі 1.4 визначено, що для розробки комп'ютерної гри-головоломки типу Sokoban найбільш доцільним інструментарієм є Unity, C# та MS Visual Studio. Наведемо більш детальний аналіз цих інструментів з точки зору мети роботи на основі їх огляду у вже існуючих програмних продуктів [24-28].

UNITY є багатоплатформовим середовищем розробки, що поєднує інструменти для створення відеоігор і застосунків із ігровим рушієм, який забезпечує їх функціонування. Програми, створені за допомогою цього середовища, можуть працювати на персональних комп'ютерах, мобільних пристроях і ігрових консолях, підтримуючи як двовимірну, так і тривимірну графіку, технології віртуальної та доповненої реальності.

Основні характеристики Unity:

1) наявність графічного рушія реального часу, що забезпечує можливість швидкого створення, редагування та тестування двовимірних і тривимірних сцен із використанням сучасних технологій освітлення, HDR та візуальних ефектів;

2) зручний та гнучкий інтерфейс користувача, який включає панелі ресурсів, ієрархії об'єктів, властивостей та інструменти налагодження, та який дозволяє налаштовувати робоче середовище як потрібно розробнику;

3) компонентна архітектура: об'єкти сцени (GameObject) можуть отримувати різні компоненти, наприклад скрипти (script), фізичні властивості, анімації та матеріали, що забезпечує високу гнучкість у створенні функціоналу;

4) підтримка популярних мов програмування, в основному C#, і є можливість використання візуального програмування для створення ігрової логіки без написання коду;

5) рушій має магазин ресурсів (Asset Store), який дозволяє використовувати готові моделі, текстури, плагіни та інші елементи (рис. 2.1);

Рисунок 2.1 – Магазин ресурсів Asset Store

6) інтеграція з популярними програмами для створення графіки та анімації Blender, Maya, 3ds Max, Cinema 4D і Photoshop, що спрощує імпорт і редагування ресурсів;

7) різноманітність ліцензій: наявна безкоштовна версія для навчальних і невеликих проєктів, і професійні версії для бізнесу;

8) кросплатформеність Unity дозволяє експортувати створені проекти на велику кількість платформ із мінімальними змінами та забезпечує сумісність із різними пристроями.

У середовищі Unity проект складається зі сцен (рівнів), які являють собою окремі файли, що містять ігровий простір із власним набором об’єктів, сценаріїв і налаштувань. Кожна сцена може включати як об’єкти з геометрією (ландшафт, персонажів або елементи оточення), так і порожні ігрові об’єкти, які не мають візуального представлення, але виконують службові функції, наприклад виступають тригерами подій або точками збереження. Ігрові об’єкти можуть бути

трансформовані у просторі: переміщені, повернуті, масштабовані, доповнені скриптами для реалізації потрібної поведінки. Кожен об’єкт має ім’я (при цьому можна використовувати однакові назви для різних об’єктів); об’єкт може мати тег і належати до якогось шару відображення.

Базовим компонентом будь-якого об’єкта є компонент Transform, який може визначати положення, орієнтацію та розміри об’єкта у тривимірному просторі. Для об’єктів із видимою геометрією за замовчуванням використовується компонент Mesh Renderer, що забезпечує їх відображення на сцені. Різні моделі можуть бути об’єднані в спеціальні набори ресурсів (асети), і це спрощує їх зберігання та повторне використання.

Unity підтримує фізичне моделювання, а саме фізику твердих тіл, тканин, систему типу Ragdoll (ця система імітує поведінку об’єктів, подібних до «ганчіркових ляльок»).

У середовищі розробки реалізовано ієрархічну структуру об’єктів, де дочірні елементи успадковують позицію, обертання та масштаб батьківського об’єкта. Скрипти в Unity додаються до об’єктів у вигляді окремих компонентів, що визначають їх поведінку. Для створення двовимірних ігор використовуються спрайти, у тривимірних проєктах застосовуються меші (тривимірні моделі, на які накладаються текстури, що визначають зовнішній вигляд поверхні); також середовище має матеріали, які задають властивості взаємодії об’єкта з освітленням, і шейдери – спеціальні програми, що обчислюють колір кожного пікселя залежно від заданих параметрів. Один шейдер може містити кілька варіантів реалізації – це дозволяє системі автоматично обирати оптимальний варіант залежно від характеристик графічного обладнання. У проєктах всіх вимірів застосовуються системи частинок для відтворення ефектів диму, вогню або рідини.

Дане програмне забезпечення підтримує стиснення текстур, міпмапінг та адаптацію до різних роздільностей екрана. Unity реалізував ефекти бамп-мапінг, відбивальні карти, паралакс-мапінг, екранне затінення навколишнього освітлення, динамічні тіні, рендеринг у текстуру та повноекранну постобробку

зображення – ефекти зернистості, глибини різкості, розмиття в русі, оптичні ефекти (наприклад, відблиски та ореоли навколо джерел світла). В середовищі Unity відображення зображення (або рендеринг) здійснюється за допомогою віртуальної камери. Розташування об’єктів у робочій області редактора буває довільним, але кінцеве зображення формується за заданим положенням та параметрів камери. У межах однієї сцени може використовуватися декілька камер, які здатні слідувати за персонажем або рухатися за заданою траєкторією.

Скриптова система Unity базується на платформі Mono, яка є відкритою реалізацією .NET Framework. Для створення шейдерів використовується мова ShaderLab, що підтримує програми, написані на GLSL або Cg. Для програмування використовуються мови програмування C#, UnityScript або Boo, основною та найбільш поширеною є саме C#. Для налагодження коду раніше використовувалося середовище MonoDevelop, у сучасних версіях інтегрована підтримка Microsoft Visual Studio, яка спрощує процес розробки та тестування скриптів.

MICROSOFT VISUAL STUDIO тісно інтегрується з ігровими движками, як, наприклад, Unity, що дозволяє зручно та ефективно розробляти ігри, писати скрипти для поведінки об’єктів та відразу тестувати їх у рушії. Це реалізується за допомогою спеціалізованого розширення Visual Studio Tools for Unity, завдяки чому є можливість синхронізувати структуру проєкту. Середовище розробки автоматично розпізнає специфічні типи даних Unity та надає контекстну документацію до них безпосередньо під час написання коду.

Ефективність кодування забезпечується інтелектуальним редактором, ядром якого є технологія автодоповнення IntelliSense, яка аналізує поточний контекст програми, пропонує найбільш ймовірні варіанти завершення функцій, методів та змінних, що мінімізує кількість синтаксичних помилок та прискорює швидкість розробки. Редактор підтримує багаторівневе підсвічування синтаксису, автоматичне форматування структури коду згідно зі стандартами мови C#, проводить статичний аналіз коду в реальному часі. Будь-які невідповідності типам даних, пропущені символи або потенційні логічні

помилки підкреслюються компілятором ще до моменту запуску програми, що дозволяє усувати дефекти на ранніх етапах.

MS Visual Studio має вбудовані засоби покрокового налагодження коду (debugging), після розробник отримує можливість встановлювати точки зупину (breakpoints) у будь-якому місці скрипту, що дозволяє зупинити виконання гри в конкретний кадр і детально дослідити поточні значення всіх змінних, стан фізичних шарів, логічні прапорці колізій та стек виклику функцій. Наявність інструментів покрокового виконання дає змогу детально проаналізувати роботу складних алгоритмів.

Для забезпечення гнучкості та масштабованості проєкту середовище MS Visual Studio надає підтримку системи .NET та вбудованого менеджера пакетів NuGet, що дозволяє легко інтегрувати в додаток сторонні бібліотеки, плагіни та розширення для оптимізації математичних розрахунків або обробки даних.

Окрім цього, у даному програмному забезпеченні реалізовано повноцінну інтеграцію із системами контролю версій (наприклад, Git). Інструментарій дозволяє керувати гілками розробки, фіксувати зміни, створювати резервні копії стабільних збірок коду та синхронізувати роботу з віддаленими репозиторіями (GitHub або GitLab); дана система гарантує безпеку зберігання вихідного коду та забезпечує прозору командну розробку програмного продукту.

Мова програмування C# (C-Sharp). Об’єктно-орієнтована мова програмування високого рівня, яка фактично є стандартом для розробки інтерактивних додатків на базі ігрового рушія Unity [29-34].

C# володіє архітектурними перевагами та високою продуктивністю в розробці ігрових механік, гармонійно поєднує в собі чіткість синтаксису мови Java та потужність мови C++, забезпечуючи строгу типізацію та модульність коду, що дозволяє мінімізувати кількість помилок на етапі компіляції додатка; має в структурі механізм автоматичного керування пам’яттю за допомогою збирача сміття – такий підхід усуває ризик виникнення витоків пам’яті, які є поширеною проблемою у мовах з ручним керуванням пам’яттю (як у C++). Для мобільних

або десктопних ігор це забезпечує стабільну частоту кадрів та запобігає раптовим критичним збоям під час тривалих ігрових сесій.

Щодо інтеграції з Unity, то C# надає повний спектр засобів для взаємодії з апаратним забезпеченням та графічною підсистемою через API рушія, де архітектура компонентно-орієнтованого програмування в Unity ідеально накладається на об'єктні структури C#, що дозволяє розробнику зручно оперувати класами-компонентами для управління фізичними властивостями та логікою гри. Сучасні версії C# мають потужний інструментарій для оптимізації коду, наприклад, технологію LINQ (Language Integrated Query) для ефективної вибірки даних та механізми асинхронного програмування. У розробці ігор-головоломок це спрощує побудову алгоритмів пошуку шляху, збереження ігрового прогресу у файли та обробку подій користувацького інтерфейсу без блокування головного ігрового потоку.

2.2 Вимоги користувачів і аналіз Sokoban-подібної гри-головоломки

Як відомо з підрозділу 1.3, гра-головоломка типу Sokoban належить до жанру двовимірних фізичних комп'ютерних ігор-головоломок. Основною особливістю ігрового процесу є необхідність логічного планування дій для проходження рівня – гравець керує персонажем, який взаємодіє з навколишніми об'єктами, і головною задачею є переміщення коробок у визначені цілі на ігровому полі. Гра реалізується у двовимірному просторі з використанням піксельної графіки – такий стиль оформлення дозволяє створити простий та зрозумілий інтерфейс, зменшити навантаження на систему та забезпечити чітке відображення елементів гри.

Ігровий процес побудований на проходженні рівнів, складність яких поступово зростає за рахунок зміни розташування стін, коробок та цільових точок. Основна особливість головоломки – необхідність аналізу ситуації та планування послідовності дій. Це означає, що в ігровому процесі відсутні випадкові події, тому результат залежить тільки від рішень користувача, який

повинен враховувати розташування об’єктів та можливі наслідки кожного переміщення, тому що неправильна дія може призвести до блокування коробок та неможливості завершення рівня.

Ігровий процес гри включає в собі переміщення персонажа по ігровому полю, взаємодію з коробками, перевірку зіткнення зі стінами, систему перевірки проходження рівня, перехід між рівнями та роботу головного меню, а також інтеграцію зворотного таймеру.

Концептуальна модель гри описує взаємодію між основними елементами системи, а саме: головне меню, ігрові сцени, персонажі, коробки, цільові точки, стіни, систему переходу між рівнями. Після запуску гри користувач потрапляє до головного меню, де може розпочати проходження або завершити роботу програми. Після натискання кнопки «Play» відкривається перший рівень гри. У межах ігрової сцени користувач керує персонажем за допомогою клавіатури, а саме клавіш WASD. Персонаж може переміщуватися у чотирьох напрямках: вгору, вниз, ліворуч та праворуч. Якщо перед персонажем знаходиться коробка, він може штовхнути її лише за умови, що наступна клітинка є вільною. Проходження рівня вважається успішним після переміщення всіх коробок у визначені цільові позиції до закінчення роботи таймеру. Після завершення рівня користувачу надається можливість перейти до наступного рівня.

Архітектура гри побудована відповідно до архітектурних компонентів, які використовуються в середовищі Unity, де усі об’єкти сцени складаються з окремих компонентів, кожен з яких відповідає за встановлену йому функцію.

Основними даними гри є позиції стін, координати коробок, розташування цільових точок, позиція персонажа, параметри рівня, елементи інтерфейсу.

Основними компонентами архітектури є графічні та фізичні компоненти, скрипти логіки коду та елементи інтерфейсу користувача. Для реалізації графічної частини використовуються спрайти (графічні ресурси гри, організовані у папках проєкту) та Tilemap, який застосовується для створення стін та підлоги ігрового поля, що дозволяє швидко та легко створювати рівні та редагувати їх структуру, та зберігає структуру рівня у вигляді набору тайлів.

Фізична взаємодія між об’єктами реалізована за допомогою компонентів Collider2D (визначення меж об’єктів та перевірку зіткнень між ними) та Rigidbody2D (забезпечення фізичної взаємодії об’єктів у двовимірному просторі).

Логіка гри реалізована за допомогою скриптів мовою програмування C#. Основними скриптами є скрипт керування персонажем MovementPlayer.cs, менеджер контролю GameManager.cs, який має в своєму повному коді методи перевірки цілей, перемоги та керування таймером; скрипт активації цілей при штовханні на них коробок TargetCheck.cs, скрипт переходу між рівнями NextLevel.cs, скрипт роботи меню MainMenu.cs, скрипт повернення до головного меню ExitGame.cs. Система сцен використовується для організації рівнів гри, де кожен рівень реалізований як окрема сцена Unity.

Розроблена комп’ютерна гра-головоломка орієнтована на широке коло користувачів та не потребує спеціальних навичок або підготовки, інтерфейс гри створений максимально простим та зрозумілим. Для взаємодії з грою користувач повинен мати базові навички роботи з персональним комп’ютером та клавіатурою. Для запуску гри необхідно використовувати персональний комп’ютер або ноутбук із підтримкою сучасних операційних систем. Гра не потребує високої продуктивності обладнання, оскільки використовує двовимірну графіку та мінімальну кількість ресурсів.

Для формалізації логіки функціонування розробленого програмного забезпечення деталізовано основний алгоритм ігрового процесу. Основний цикл гри (від завантаження рівня до перевірки умов перемоги) виконується за таким алгоритмом:

1. Ініціалізація рівня: завантаження поточної ігрової сцени в середовищі Unity; зчитування початкових координат об’єктів стін та підлоги із матриці Tilemap; генерація ігрових об’єктів у координатах сцени: персонажа (Player), коробок (Box) та цільових точок (Target); встановлення лічильника заповнених цілей у значення 0.

2. Очікування та обробка дій користувача (Ігровий цикл): система безперервно опитує пристрої введення (клавіши клавіатури); при фіксації натискання клавіші визначається вектор бажаного напрямку руху персонажа.

3. Перевірка можливості ходу: Обчислюється координата наступної клітинки за напрямком руху $P_{next} = P_{player} + vector\ D$. Перевірка наявності стіни у позиції P_{next}: якщо в позиції стіна, тоді хід блокується (повернення до 2); якщо позиція вільна від стін та коробок, тоді персонаж переміщується в P_{next} (повернення до 2). Перевірка наявності коробки у позиції P_{next}: отримується координата клітинки за коробкою $P_{boxnext} = P_{next} + vector\ D$. Якщо в $P_{boxnext}$ знаходиться стіна або інша коробка, тоді хід блокується, оскільки коробку не можна зрушити (повернення до 2); якщо клітинка $P_{boxnext}$ вільна, тоді виконується одночасне переміщення коробки в позицію $P_{boxnext}$, а ігрового персонажа в позицію P_{next} (перехід до 4).

4. Система активації цілей та перевірки умов перемоги: Після переміщення коробки викликається скрипт TargetCheck.cs, який перевіряє статус клітинки $P_{boxnext}$: якщо коробка стала на цільову точку, тоді прапорець активації точки змінюється на true, лічильник успішно розміщених коробок збільшується на 1; якщо коробка зійшла з цільової точки, тоді прапорець змінюється на false, лічильник зменшується на 1. Перевірка умови завершення рівня: якщо поточна кількість активованих точок дорівнює загальній кількості цілей на рівні, то фіксується перемога. Після цього скрипт GameManager.cs зупиняє ігровий таймер, блокує введення користувача та виводить на екран слово «Перемога» (якщо інакше – повернення до кроку 2).

Графічну інтерпретацію описаної логіки у вигляді блок-схеми алгоритму обробки ходу та взаємодії об’єктів наведено на рис 2.2.

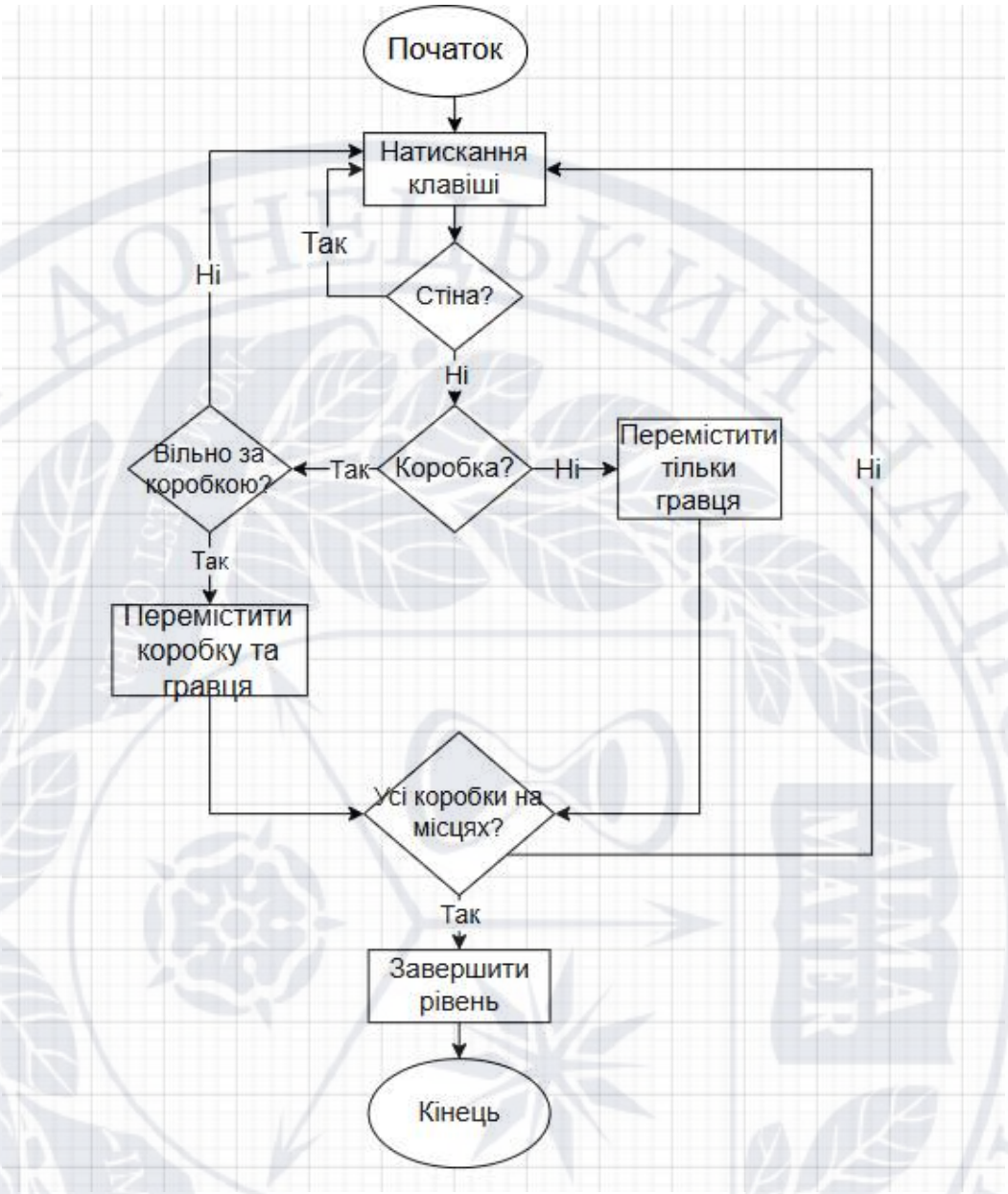

Рисунок 2.2 – Блок-схема ігрового процесу комп’ютерної гри-головоломки

2.3 Візуалізація комп’ютерної гри-головоломки

Для реалізації комп’ютерної гри-головоломки обрано 2D-графіку в стилі піксель-арт [35-39]. Гра виконується в 2D-анімації [17, 40].

Двовимірна графіка (**2D-графіка**) – це вид комп’ютерної графіки, у якому об’єкти зображуються у двовимірному просторі та описуються двома координатами: ширина (X) та висота (Y), завдяки чому 2D-графіка є відносно

простою у реалізації та не потребує значних обчислювальних ресурсів. Двовимірні графічні моделі можуть включати різні типи даних, наприклад: геометричні об’єкти (векторна графіка), растрові зображення (рис. 2.3), текстові елементи з визначеними параметрами (шрифт, розмір, колір, розташування та орієнтація). Усі ці складові можуть змінюватися за допомогою базових геометричних перетворень, таких як переміщення, обертання та масштабування.

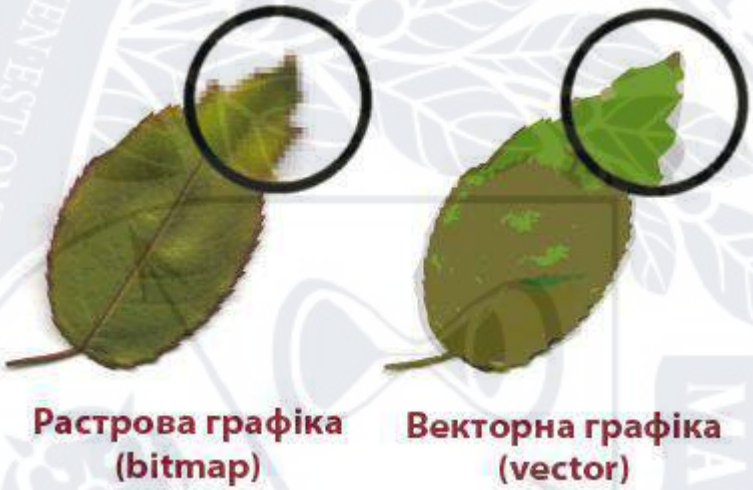

Рисунок 2.3 – Порівняння зображень з растровою та векторною графікою

У двовимірній графіці всі елементи можна уявити як окремі об’єкти (векторні лінії або пікселі). Кожен із цих об’єктів має свої властивості та може самостійно відображатися на екрані. Під час відображення (рендерингу) програма визначає, якого кольору має бути окрема точка зображення. Складні зображення створюються шляхом поєднання простіших елементів. У евклідовій геометрії паралельне перенесення означає переміщення всіх точок фігури на однакову відстань у певному напрямку (рис. 2.4). Таке перетворення не змінює форму і розміри об’єкта, тому його можна розглядати як рух твердого тіла, подібно до обертання або віддзеркалення. Паралельне перенесення також можна описати як додавання одного й того ж вектора до кожної точки або як зміну положення початку координат.

Рисунок 2.4 – Приклад паралельного перенесення в графічному дизайні

Створення складного зображення зазвичай починається з порожнього растрового поля, яке заповнюється однорідним кольором. Після цього на нього послідовно наносяться окремі елементи – прості фігури, зображення або текст, що формують кінцеву картинку. Деякі програми безпосередньо змінюють колір окремих пікселів – найчастіше використовуються спеціальні графічні бібліотеки або можливості відеокарти, які забезпечують виконання основних операцій, наприклад вставку зображень у потрібну позицію, відображення тексту із заданими параметрами, побудову геометричних фігур, малювання ліній і кривих із заданими характеристиками.

Текст, геометричні фігури та лінії відображаються з використанням кольорів, які задає користувач. Для створення більш складних візуальних ефектів застосовуються колірні градієнти, що дозволяють формувати плавні переходи між кольорами. Колір пікселів може визначатися за допомогою текстур.

Під час відображення новий колір пікселя зазвичай замінює попередній, проте багато графічних систем підтримують прозорість і напівпрозорість, що дає змогу частково поєднувати новий і попередній кольори. Можливе комбінування кольорів різними способами за допомогою деяких операцій (інверсія кольору), що часто використовується для підсвічування об’єктів, тому що повторне застосування тієї ж операції дозволяє відновити початковий вигляд зображення.

Моделі, що використовуються у двовимірній комп'ютерній графіці, зазвичай не враховують тривимірні об'єкти та оптичні ефекти (освітлення, тіні, відбиття). Для ускладнення зображення застосовується підхід із використанням кількох шарів, які розташовуються один над одним у певному порядку. Кожен шар має свою «глибину», що визначає його положення відносно глядача. Такі багаторівневі моделі іноді називають *2,5D-графікою*. Основною перевагою такої графіки є можливість редагування окремих шарів без впливу на інші.

Формування фінального зображення відбувається шляхом послідовного накладання шарів на віртуальне полотно відповідно до їх порядку. Кожен шар обробляється окремо, після чого його пікселі накладаються на загальне зображення, за винятком прозорих ділянок. У сучасних іграх двовимірна графіка реалізується за допомогою спрайтів – окремих растрових зображень, які використовуються для відображення персонажів, об'єктів і елементів інтерфейсу. Спрайти можуть об'єднуватися в набори (спрайтові атласи), що дозволяє оптимізувати продуктивність гри. Для створення та обробки графічних ресурсів застосовуються спеціалізовані редактори (Adobe Photoshop) і безкоштовні альтернативи (Ibis Paint X).

Піксельна графіка є видом цифрового зображення, яке створюється за допомогою растрових графічних редакторів. Зображення формується та редагується на рівні окремих пікселів, а його роздільна здатність зазвичай є настільки низькою, що кожен піксель чітко помітний. За своєю структурою піксельна графіка нагадує традиційні види декоративного мистецтва: вишивка хрестиком, мозаїка, бісероплетіння – тому що зображення формується з великої кількості дрібних кольорових елементів, подібних до пікселів на екрані дисплея. Піксельна графіка є видом двовимірної цифрової графіки, у якій зображення формується з окремих кольорових точок – пікселів. Основною особливістю цього стилю є чітко виражена «піксельна сітка», де кожен елемент зображення має фіксоване положення та колір (рис. 2.5). Завдяки цьому піксельна графіка має характерний ретро-стиль і часто використовується в інді-іграх та проектах з обмеженими ресурсами.

Рисунок 2.5 – Зображення в піксельній графіці

Низькі вимоги до пам’яті завдяки використанню форматів із невеликою кількістю кольорів – навіть при обмеженій передачі кольору піксельні зображення зберігають свою виразність і впізнаваний стиль. Піксельна графіка добре виглядає на дисплеях із чіткою структурою пікселів, але водночас на сучасних потужних екранах із високою роздільною здатністю та підтримкою складних графічних ефектів інші стилі можуть виглядати більш реалістично та привабливо. Піксельна графіка погано переносить автоматичне масштабування: при зміні розміру зображення виникає необхідність його переробки або оптимізації, інакше воно може втрачати якість або чіткість. На деяких типах дисплеїв можливі візуальні спотворення, наприклад мерехтіння або ефект «сітки», що негативно впливає на сприйняття зображення.

2D-анімація – вид комп’ютерної графіки, у якому рух об’єктів створюється в двовимірному просторі шляхом послідовної зміни зображень або параметрів спрайтів. Широко використовується у відеоіграх, мультимедійних застосунках та інтерфейсах користувача. Головною метою 2D-анімації є створення ілюзії руху за рахунок швидкої зміни кадрів або трансформації графічних елементів. У розробці ігор 2D-анімація найчастіше реалізується за допомогою спрайтової

анімації, де послідовність кадрів відтворюється з фіксованою швидкістю. Інший спосіб створення рухів полягає у процедурній анімації, коли зміни об’єкта відбуваються шляхом математичних перетворень – позиція, обертання, масштабування об’єкта, що дозволяє створювати плавний рух без необхідності великої кількості окремих зображень. В ігрових рушіях типу Unity 2D-анімація реалізується за допомогою системи Animator, Animation Controller та Animation Clips, що допомагає створювати різні стани об’єктів (наприклад рух, стояння, взаємодія) та керувати переходами між ними – це значно спрощує розробку складної поведінки персонажів та ігрових об’єктів.

Отже, у другому розділі детально проаналізовано обраний інструментарій для розробки, розроблено концепцію комп’ютерної 2D гри-головоломки, визначено її структуру, основні механіки та принципи взаємодії користувача з ігровим середовищем, спроектовано алгоритм ігрового процесу, що дозволило сформувати основу для подальшої реалізації гри у середовищі Unity та забезпечило цілісність структури програмного продукту.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ КОМП'ЮТЕРНОЇ ГРИ В UNITY

3.1 Інтерфейс розробленої комп'ютерної гри-головоломки

Розроблена комп'ютерна гра-головоломка типу Sokoban має 3 головні ігрові сцени: Меню, Рівень 1, Рівень 2.

Меню має дві UI кнопки – Play та Quit (рис. 3.1). Після натискання кнопки Play відкривається перший рівень гри. Після натискання кнопки Quit вікно гри закривається. Як графічні елементи були використані графічні ресурси (UI-спрайти), заздалегідь завантажені з Asset Store Unity.

Рисунок 3.1 – Вигляд головного меню розробленої гри

Рівень 1 має в своїй структурі вигляд закритої кімнати, яка обмежена стінами та вистелена підлогою. В кімнаті знаходиться ігровий персонаж Player, заздалегідь визначена кількість коробок і стільки ж цілей (рис. 3.2).

Рисунок 3.2 – Рівень 1

На рівні 1 реалізовано як UI-елемент кнопку, що дозволяє перейти на наступний рівень 2 (рис. 3.3).

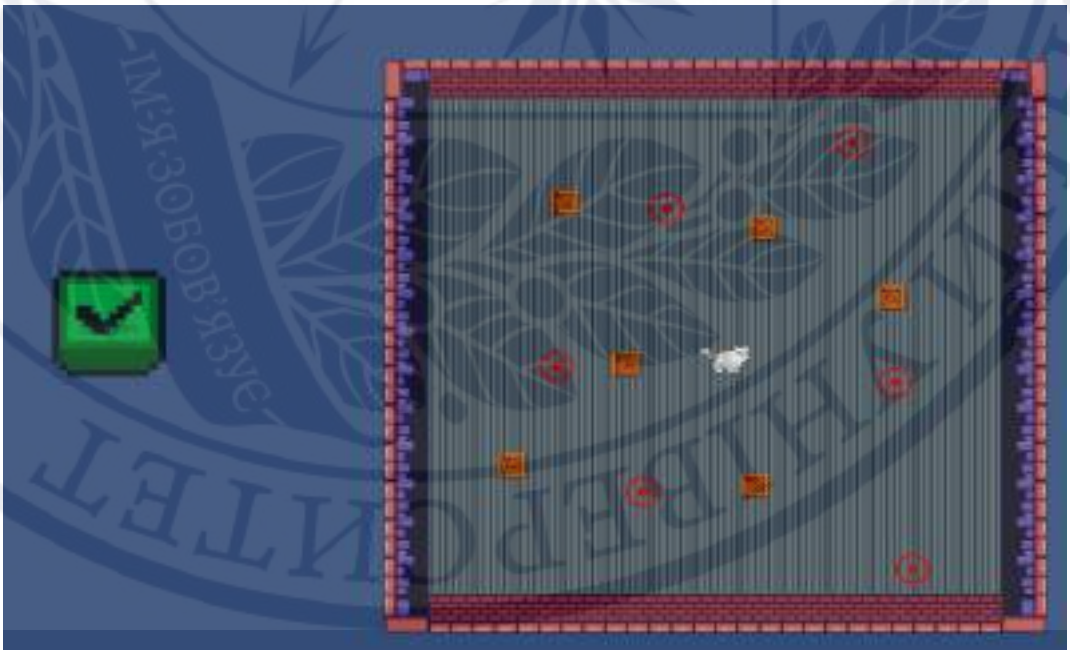

Рисунок 3.3 – Рівень 1 та кнопка Наступний рівень (зелена)

Рівень 2 має складнішу структуру організації рівня: ускладнення відбувається за рахунок додавання стін для зміни вигляду кімнати, та зменшення кількості коробок та цілей (рис. 3.4).

Рисунок 3.4 – Рівень 2

На рівні 2 реалізовано кнопку, після натискання якої користувач повертається в головне меню і має можливість або почати спочатку, або вийти з гри (рис. 3.5).

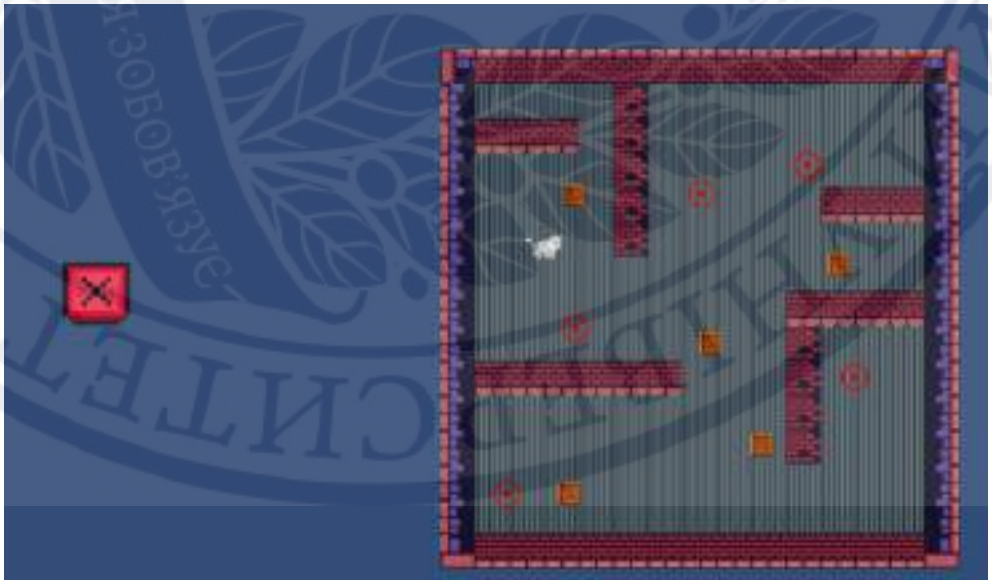

Рисунок 3.5 – Кнопка Повернутись в головне меню (червона)

Дані сцени створювались на об’єкті Tilemap, який використовується для розміщення графічних елементів рівня. Без графічних об’єктів пустий Tilemap виглядає як порожня, сірого кольору сцена без жодних об’єктів (рис. 3.6):

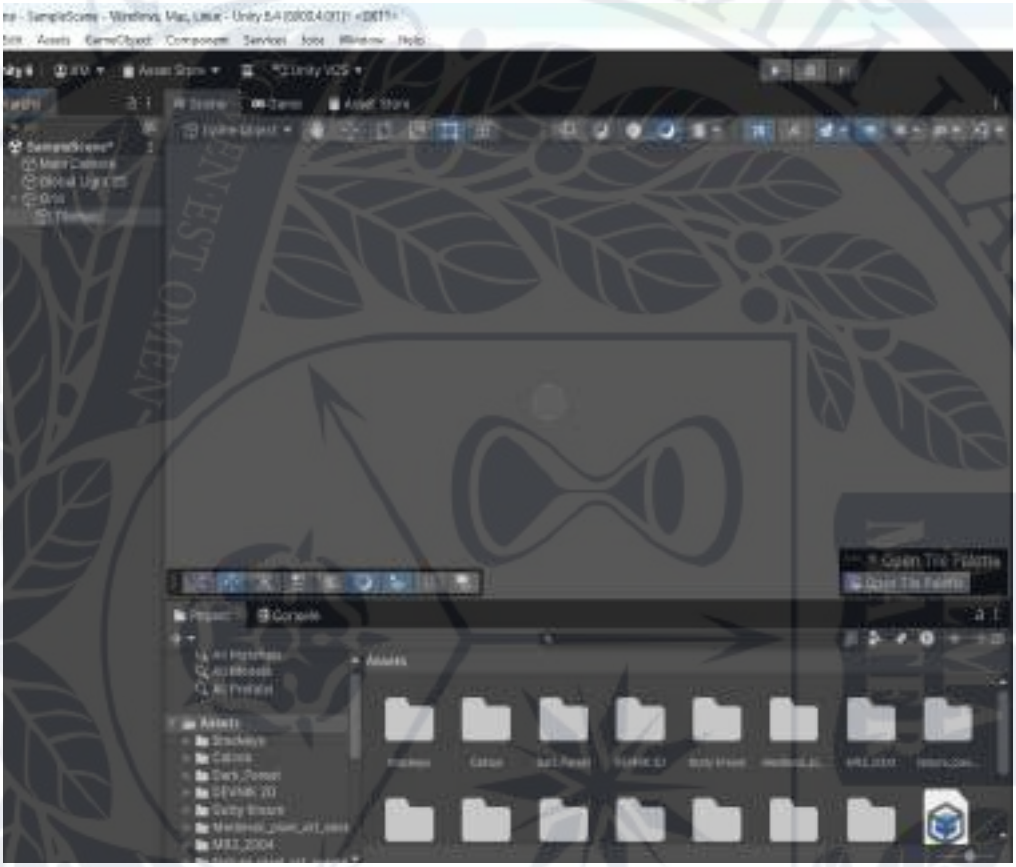

Рисунок 3.6 – Пустий Tilemap

Додавання графічних елементів на Tilemap відбувається за допомогою Tile Palette – інструмент, у якому формується набір тайлів (окремих графічних елементів), що застосовуються для побудови рівня, додаються готові спрайти і які потім використовуються (рис. 3.7). Можна використовувати готові тайтли, додані програмою автоматично на основі завантажених ассетів, а можна створити свою палетку і додати те, що вважається потрібним та скомбінувати різні графічні елементи.

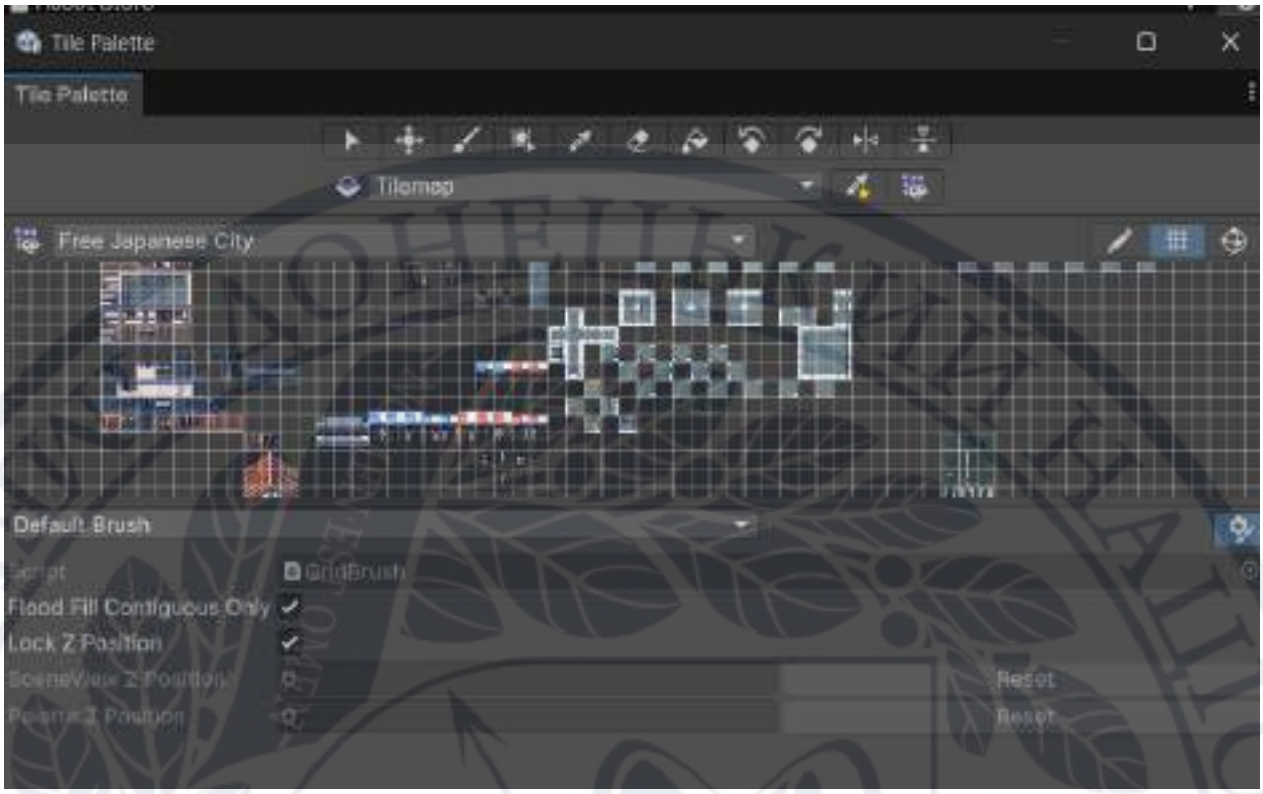

Рисунок 3.7 – Інструмент Tile Palette

Таким чином з цього вікна перетягуються обрані спрайти на сцену і будується рівень (рис. 3.8).

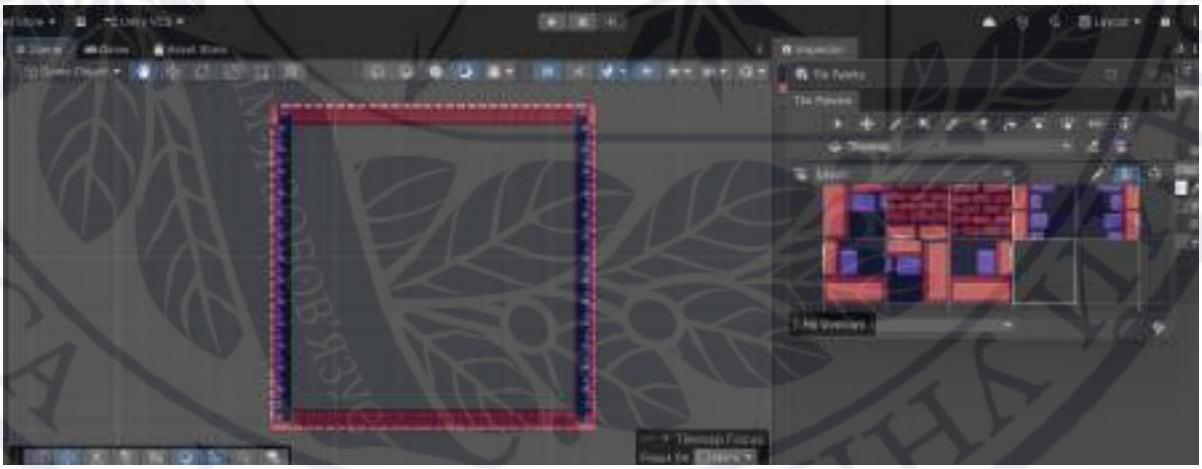

Рисунок 3.8 – Побудова рівня

Для кожного елементу рівня, стін чи підлоги створено окремі Tilemap, щоб фізика цих об’єктів була різною та гра працювала коректно, наприклад стінам встановлено статичну фізику, а підлога залишена як фон (рис. 3.9).

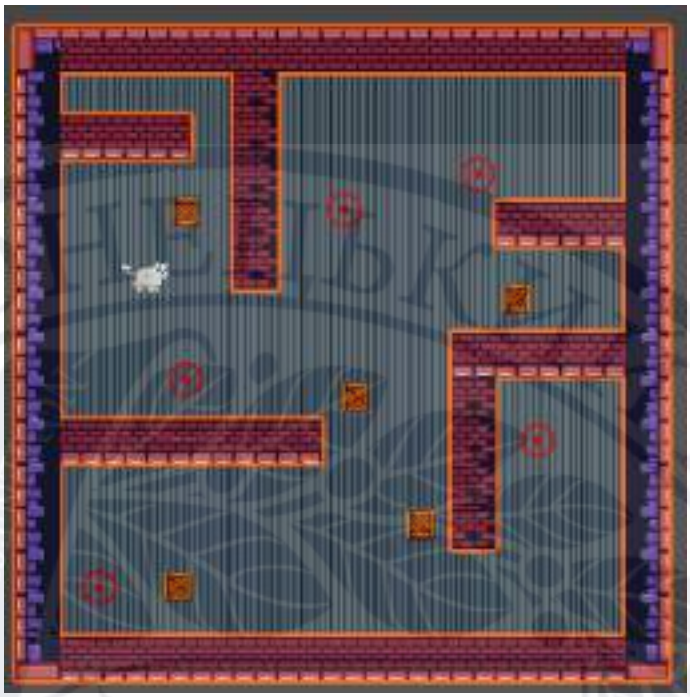

Рисунок 3.9 – Помаранчевим кольором виділено Tilemap тільки зі стінами

Об’єкти гри Player, Box, Target спочатку створено як звичайні фігури (коло, квадрат); в процесі розробки їх замінено на обрані завантажені спрайти (рис. 3.10).

Рисунок 3.10 – Жовтий квадрат – Box, білий – Player (до заміни на спрайти)

Усі графічні ресурси або спрайти, що використано в даній комп’ютерній грі-головоломці, завантажені та встановлені з Asset Store – магазина ресурсів Unity (рис. 3.11). Даний спосіб використання ресурсів надзвичайно економний, зручний і допомагає одназу використовувати готові картинки для створення гри, без їх малювання з нуля в Photoshop. Альтернативний сайт для завантаження спрайтів – Kenney, з якого можна завантажити найрізноманітніші ассети як дво-, так і тривимірної графіки.

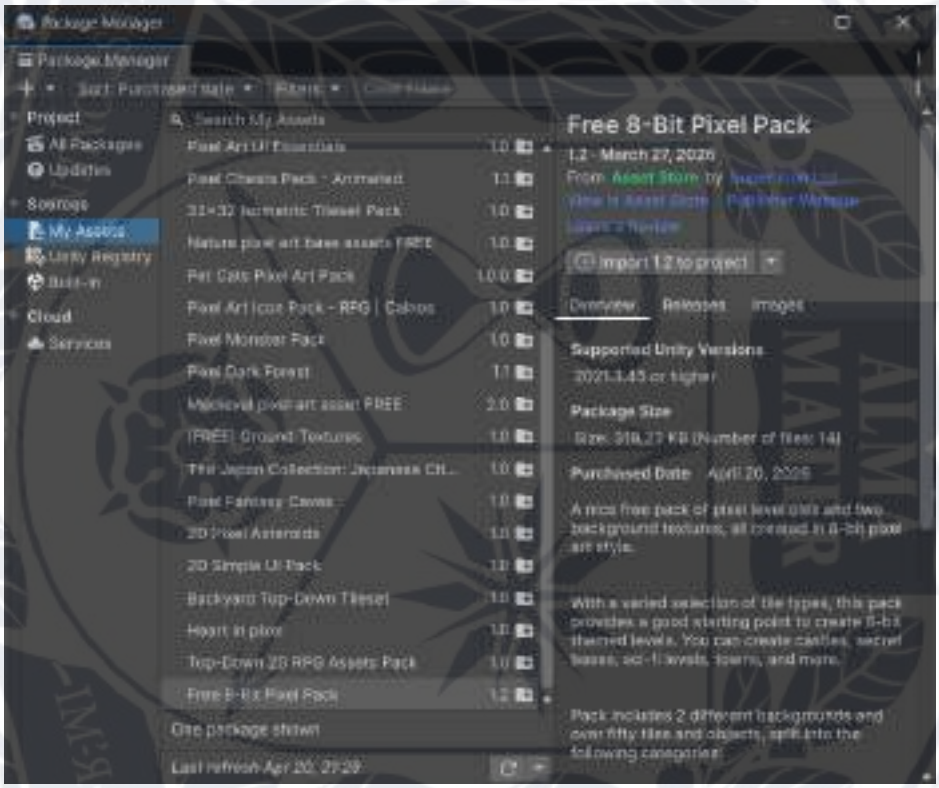

Рисунок 3.11 – Завантажені спрайти з Asset Store

Режим користувача. Під час запуску гри користувач потрапляє в головне меню і має можливість почати гру або вийти з неї. Після натискання кнопки Play користувач потрапляє на перший рівень гри, з якою він взаємодіє за допомогою клавіатури. Для переміщення персонажа використовуються клавіші W, A, S, D. На рівні є 60-секундний таймер. Гравець повинен перемістити всі коробки у визначені цільові точки до закінчення таймеру для завершення рівня, після чого

на екран показується повідомлення «ПЕРЕМОГА» (рис. 3.12). Якщо необхідно, гравець може перезапустити рівень, натиснувши клавішу R на клавіатурі.

Рисунок 3.12 – Екран перемоги

Користувач може перейти на наступний рівень за допомогою відповідної кнопки, а на наступному рівні повернутись у головне меню. Рівень вважається непройденим, якщо користувач: 1) переставив коробку дуже близько до стіни, тим самим не маючи можливості передвинути її назад в ігрове поле; 2) застряг у стіні і не має можливості з неї вибратись; 3) не встиг доставити коробки до цілей до закінчення таймеру.

3.2 Процесорна частина комп’ютерної гри-головоломки

Програмна логіка гри реалізована мовою програмування C# в середовищі Unity та MS Visual Studio, які пов’язані між собою. В Unity в папці Assets створюється пустий скрипт, і, натиснувши на нього два рази, відкривається MS Visual Studio, де здійснюється введення коду. При цьому в MS Visual Studio одразу створюється середовище гри .sln, яке пов’язане з Unity та проектом гри та в якому зберігаються усі скрипти проекту.

Для керування ігровим персонажем створено скрипт MovementPlayer.cs, який дозволяє керувати персонажем за допомогою клавіш клавіатури, при цьому гравець за один клік рухається на одну клітинку чітко в заданому напрямку (наліво, направо, вгору, вниз) На рис. 3.13 наведено фрагмент лістингу (Додаток А), який реалізує стандартний системний метод «Update», що виконується кожного кадру гри та відповідає за зчитування дій користувача і визначення вектора напрямку руху персонажа у двовимірному просторі.

```
void Update()
{
    if (isMoving) return;

    Vector3 dir = Vector3.zero;

    if (Input.GetKeyDown(KeyCode.W)) dir = Vector3.up;
    if (Input.GetKeyDown(KeyCode.S)) dir = Vector3.down;
    if (Input.GetKeyDown(KeyCode.A)) dir = Vector3.left;
    if (Input.GetKeyDown(KeyCode.D)) dir = Vector3.right;

    if (dir != Vector3.zero)
        TryMove(dir);
}
```

Рисунок 3.13 – Фрагмент лістингу MovementPlayer.cs

На самому початку методу здійснюється захисна перевірка if (isMoving) return;, яка миттєво перериває подальше виконання коду, якщо гравець уже

перебуває у процесі переміщення. Далі ініціалізується нульовий вектор напрямку Vector3 dir = Vector3.zero, після чого за допомогою послідовних умовних операторів if` та вбудованого методу Input.GetKeyDown програма перевіряє факт натискання клавіш керування: клавіші W та S задають вектори руху вгору (Vector3.up) і вниз (Vector3.down), а клавіші A та D – вліво (Vector3.left) і вправо (Vector3.right) відповідно. Наприкінці виконується фінальна перевірка if (dir != Vector3.zero), яка у разі реєстрації натискання будь-якої з перелічених клавіш передає сформований вектор напрямку як аргумент у кастомний метод TryMove(dir) для подальшого прорахунку фізичних колізій із коробками чи стінами та безпосереднього переміщення ігрового об’єкта.

Скрипт TargetCheck.cs перевіряє місцезнаходження коробки на цілі (рис. 3.14). На початку роботи скрипту в системному методі Start за допомогою функції FindObjectOfType<GameManager>() відбувається пошук та ініціалізація посилання на центральний менеджер гри для подальшої координації дій.

```
using UnityEngine;

public class TargetCheck : MonoBehaviour
{
    private GameManager gameManager;
    private bool isOccupied = false;

    private void Start()
    {
        gameManager = FindObjectOfType<GameManager>();
    }

    private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.CompareTag("Box") && !isOccupied)
        {
            isOccupied = true;
            Debug.Log("Коробка встала на місце!");
            gameManager.TargetActivated();
        }
    }

    private void OnTriggerExit2D(Collider2D other)
    {
        if (other.CompareTag("Box") && isOccupied)
        {
            isOccupied = false;
            Debug.Log("Коробку прибрали з місця!");
            gameManager.TargetDeactivated();
        }
    }
}
```

Рисунок 3.14 – TargetCheck.cs

Основне функціонування класу базується на обробці подій фізичного рушія Unity через два методи зворотного виклику: OnTriggerEnter2D та OnTriggerExit2D. Метод OnTriggerEnter2D спрацьовує при входженні стороннього колайдера в зону тригера; він виконує перевірку за тегом other.CompareTag("Box") та поточним станом зайнятості !isOccupied, після чого змінює прапорець isOccupied на true, надсилає інформаційне повідомлення в консоль розробника та викликає метод TargetActivated() об'єкта gameManager для збільшення загального лічильника прогресу. Зворотний метод OnTriggerExit2D фіксує момент, коли гравець зміщує коробку з цільової точки, скидає логічний прапорець isOccupied у значення false, реєструє це в консолі та через метод TargetDeactivated() дає команду центральному менеджеру зменшити лічильник активованих зон.

Для того, щоб після місцезнаходження усіх коробок на цілях на екран виводилось слово «Перемога», реалізовано скрипт GameManager.cs (Додаток Б), який порівнює активовані цілі з усіма наявними на рівні; якщо так, то він активує текст «Перемога», який завчасно зроблено неактивним (рис. 3.15).

На фрагменті коду у класі оголошено дві змінні для збереження прогресу: публічне ціле число totalTargets, що визначає фіксовану кількість необхідних для перемоги об’єктів (за замовчуванням 6), приватне ціле число targetsAchieved для відстеження поточної кількості успішно активованих цілей, публічне посилання на ігровий об’єкт winText, що відповідає за графічне відображення напису про перемогу. Основна логіка реалізована за допомогою двох публічних методів: TargetActivated(), який викликається при встановленні ігрового елемента у правильну зону, збільшує лічильник досягнутих цілей на одиницю та перевіряє умову if (targetsAchieved >= totalTargets), після виконання якої виводить повідомлення в консоль розробника Debug.Log і активує інтерфейс перемоги за допомогою функції winText.SetActive(true). Другий метод TargetDeactivated() реалізує зворотну логіку та декрементує значення targetsAchieved на одиницю у випадку, якщо гравець змістив об’єкт із цільової позиції, забезпечуючи динамічний перерахунок поточного стану гри.

```
using UnityEngine;

public class GameManager : MonoBehaviour
{
    public int totalTargets = 6;
    private int targetsAchieved = 0;
    public GameObject winText;

    public void TargetActivated()
    {
        targetsAchieved++;
        if (targetsAchieved >= totalTargets)
        {
            Debug.Log("ПЕРЕМОГА! Всі коробки на місцях!");
            winText.SetActive(true);
        }
    }

    public void TargetDeactivated()
    {
        targetsAchieved--;
    }
}
```

Рисунок 3.15 – Фрагмент скрипту GameManager.cs з методами перевірки усіх коробок

Скрипт MainMenu.cs реалізує роботу головного меню гри: після натисканні кнопки Play програма запускає перший рівень гри, при натисканні Exit – закривалась (рис. 3.16). Для забезпечення взаємодії з підсистемою завантаження ігрових рівнів у скрипті підключено спеціалізований простір імен using UnityEngine.SceneManagement;.

Функціонал класу структуровано за допомогою двох публічних методів, які призначені для зв'язку з графічними компонентами UI через подієву систему рушія: метод PlayGame() відповідає за ініціалізацію початку ігрового процесу шляхом виклику функції SceneManager.LoadScene(), яка приймає як аргумент рядковий параметр "SampleScene", що дозволяє вивантажити поточну сцену головного меню з оперативної пам'яті та завантажити безпосередньо ігрову сцену з головоломкою, а метод QuitGame() призначений для коректного завершення роботи додатка через звернення до системного методу Application.Quit(), який

надсилає операційній системі сигнал на закриття процесу гри та звільнення всіх задіяних апаратних ресурсів у фінальній скомпільованій збірці проєкту.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public void PlayGame()
    {
        SceneManager.LoadScene("SampleScene");
    }

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

Рисунок 3.16 – Скрипт MainMenu.cs

Для кнопки, при натисканні якої користувач переключається на наступний рівень, реалізовано скрипт NextLevel.cs, який можна побачити на рис. 3.17. Логіку класу NextLevel зосереджено в одному публічному методі LoadNextLevel(), який зазвичай прив'язується до інтерфейсних кнопок переходу або викликається автоматично після успішного виконання умов перемоги. Внутрішня структура методу використовує функцію SceneManager.LoadScene(), аргументом для якої слугує динамічно розрахований індекс, який визначається шляхом звернення до поточної активної сцени через SceneManager.GetActiveScene().buildIndex та додає до отриманого числового значення одиниці.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class NextLevel : MonoBehaviour
{
    public void LoadNextLevel()
    {
        SceneManager.LoadScene(
            SceneManager.GetActiveScene().buildIndex + 1
        );
    }
}
```

Рисунок 3.17 – Скрипт NextLevel.cs

Для кнопки, яка дозволяє користувачеві повернутись в головне меню, реалізовано скрипт ExitGame.cs, який завантажує сцену головного меню гри (рис. 3.18). Уся програмна логіка компонента зосереджена в єдиному публічному методі ExitToMenu(), розробленому для зв'язку з графічними елементами користувацького інтерфейсу. Внутрішня інструкція методу виконує виклик статичної функції SceneManager.LoadScene(), передаючи їй як єдиний аргумент рядковий ідентифікатор "MainMenu". Під час активації даної команди поточний стан ігрового рівня вивантажується з оперативної пам'яті комп'ютера, після чого рушій миттєво ініціалізує та відображає на екрані стартову сцену головного меню, забезпечуючи коректний зворотно-навігаційний цикл у структурі розроблюваного додатка.

```
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  public class ExitGame : MonoBehaviour
5  {
6      public void ExitToMenu()
7      {
8          SceneManager.LoadScene("MainMenu");
9      }
10 }
```

Рисунок 3.18 – Скрипт ExitGame.cs

В грі розроблена можливість перезапуску рівня. Якщо користувач вважає за потребу почати рівень заново і скинути прогрес, він може натиснути для перезапуску клавішу R на клавіатурі (рис. 3.19). Основна логіка компонента зосереджена всередині системного методу Update(), який автоматично викликається рушієм кожного кадру для відстеження дій гравця. Внутрішня інструкція методу містить умовний оператор if (Input.GetKeyDown(KeyCode.R)), що безперервно опитує стан клавіатури на предмет натискання гарячої клавіші R. У момент фіксації цієї події спрацьовує статична функція SceneManager.LoadScene(), аргументом для якої слугує динамічно отриманий індекс поточної локації SceneManager.GetActiveScene().buildIndex.

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class RestartLevel : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R))
        {
            SceneManager.LoadScene(
                SceneManager.GetActiveScene().buildIndex
            );
        }
    }
}
```

Рисунок 3.19 – RestartLevel.cs

В грі реалізовано 60-секундний зворотний таймер. Рівень необхідно встигнути пройти до його закінчення; якщо ні, то рівень вважається нейпройденим, а замість цифр виводиться фраза «Час вийшов!». Код роботи таймеру реалізовано додатково в скрипт GameManager.cs. Наведений фрагмент коду (рис. 3.20) реалізує приватний метод «UpdateTimerDisplay», призначений для динамічного оновлення та форматування візуального відображення ігрового таймера в інтерфейсі користувача.

Спочатку за допомогою математичної функції округлення до меншого цілого «Mathf.FloorToInt» відбувається конвертація секунд у хвилини (шляхом ділення на 60) та виокремлення залишку секунд у межах поточної хвилини (за допомогою оператора взяття остачі від ділення «%»). Якщо текстовий компонент успішно ініціалізовано, метод «string.Format» здійснює побудову текстового рядка, де специфікатори «{0:00}» та «{1:00}» автоматично додають провідні нулі до однозначних чисел, забезпечуючи класичний цифровий формат часу «XX:XX», який одразу записується у властивість «.text» графічного компонента «timerText».

```
private void UpdateTimerDisplay(float timeToDisplay)
{
    float minutes = Mathf.FloorToInt(timeToDisplay / 60);
    float seconds = Mathf.FloorToInt(timeToDisplay % 60);

    if (timerText != null)
    {
        timerText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }
}
```

Рисунок 3.20 – Фрагмент коду GameManager.cs з методом для роботу таймеру

Повний вихідний код розроблених скриптів, що забезпечують функціонування ігрових механік, взаємодію об'єктів та логіку керування інтерфейсом, наведено у Додатках А та Б.

Для скриптів ExitGame.cs, NextLevel.cs, GameManager.cs та MainMenu.cs додатково створювано пусті об’єкти Empty Object, в які ці скрипті додавались. Потім пусті об’єкти прив’язувались до події OnClick() відповідних кнопок і в параметрах присвоювалась дія, яку виконує кнопка – наприклад, перехід на наступний рівень або закриття гри (рис. 3.21). Це зроблено для того, щоб UI кнопки гри працювали і виконували розроблену для них дію. Для роботи інших скриптів було достатньо перетягнути їх на об’єкт виконання.

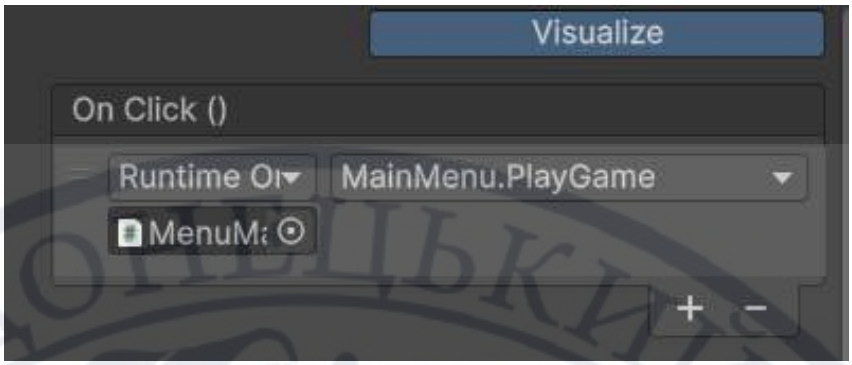

Рисунок 3.21 – Подія OnClick() для кнопки Play головного меню

Двовимірна фізика компонентів гри стін, гравця, коробок та цільових позицій, реалізована за допомогою фізичних компонентів середовища Unity. Для визначення меж взаємодії між об’єктами використовувався компонент Box Collider 2D, який забезпечує перевірку зіткнень та обмежує проходження об’єктів крізь стіни й інші елементи сцени. Для реалізації фізичних властивостей і переміщення об’єктів застосовано компонент Rigidbody 2D, який дозволяє обробляти рух об’єктів у двовимірному просторі та забезпечує коректну фізичну взаємодію між елементами гри.

Використання компонентів фізики дозволило реалізувати механіку переміщення коробок, систему зіткнень зі стінами та взаємодію гравця з ігровими об’єктами. Для коректної роботи логіки гри були налаштовані параметри колайдерів, обмеження руху та фізичні характеристики об’єктів, що забезпечило стабільність роботи основних механік гри.

3.3 Тестування комп’ютерної гри та перспективи її розвитку

Під час проведення комплексного функціонального та інтеграційного тестування гри виявлено особливість фізичної взаємодії персонажа зі стінами, за якої у деяких випадках можливе часткове блокування руху персонажа при надто близькому контакті з елементами оточення. Для мінімізації виникнення даної ситуації на етапі системного тестування оптимізовано структуру рівнів та розташування стін, що дозволило повністю уникнути критичних ситуацій під час

проходження гри. Крім того, у грі реалізовано функціонал швидкого перезапуску рівня, який дозволяє користувачу миттєво відновити ігровий процес у разі виникнення нестандартної ситуації або логічного тупика. Водночас результати фінального внутрішнього тестування показали стабільну роботу основних механік гри, коректне функціонування інтерфейсу та повну відповідність розробленого програмного продукту висунутим технічним вимогам.

Перспективи подальшого розвитку гри. Наразі розроблена комп’ютерна Sokoban-подібна гра-головоломка реалізована за допомогою базових механік та та компонентів, які визначають жанр головоломки (Sokoban) та які дозволяють зручно керувати грою (вихід з гри, перехід на наступний рівень, повернення в головне меню). Даний цифровий продукт має перспективи для розвитку в майбутньому. Розроблену комп'ютерну гру можна розвивати далі шляхом додавання:

- додаткових рівнів складності, кожний наступний з яких ускладнюється шляхом додавання перешкод у вигляді додаткових стін та зміною параметрів кімнати (наприклад, зробити її не у формі квадрата, а у формі складніших фігур з вузькими проходами всередині);
- окремої сцени зі списком усіх наявних в грі рівнів; при цьому зробити активними ті рівні, які користувач вже пройшов;
- збереження прогресу проходження рівня;
- системи підказок;
- звукового супровіду.

ВИСНОВКИ

У кваліфікаційній роботі розроблено двовимірну комп'ютерну гру-головоломку типу Sokoban у середовищі Unity. У результаті виконання роботи повністю досягнуто поставленої мети та вирішено всі визначені завдання:

Проаналізовано аналоги комп'ютерних ігор у жанрі головоломки, досліджено їхні основні механіки, принципи побудови ігрового процесу та особливості взаємодії користувача з ігровим середовищем. Особливу увагу приділено логічним іграм типу Sokoban, де основою геймплею є планування послідовності дій для переміщення об'єктів. На основі аналізу виділено ключові особливості жанру, такі як детермінованість ігрового процесу (відсутність випадкових подій) та критичність кожної дії користувача.

Сформовано концепцію комп'ютерної гри-головоломки, що включає правила проходження рівнів, покрокову механіку керування персонажем та систему логічної взаємодії з коробками, стінами й цільовими точками. Запрограмовано логіку рівнів із поступовим зростанням складності за рахунок зміни конфігурації перешкод. Розроблено простий графічний інтерфейс користувача, який містить головне меню, кнопки керування грою та систему автоматичного переходу між сценами.

Обґрунтовано вибір інструментальних засобів для реалізації програмного продукту, якими стали міжплатформне ігрове середовище розробки Unity та мова програмування C#. Використання Unity дозволило ефективно реалізувати 2D-графіку, фізику колізій (Collider2D, Rigidbody2D) та оптимізувати архітектуру ігрового простору за допомогою системи Tilemap. Мова C# забезпечила гнучку реалізацію об'єктно-орієнтованої логіки проєкту.

Розроблено та програмно реалізовано гру-головоломку з набором базових рівнів з урахуванням специфіки руху об'єктів. Створено архітектуру взаємопов'язаних скриптів (MovementPlayer.cs, GameManager.cs, TargetCheck.cs тощо), які забезпечують безпомилкове покрокове переміщення гравця та штовхання коробок лише за умови вільного простору за ними.

Оцінено роботу комп’ютерної гри та визначено її подальші перспективи розвитку. Шляхом проведення функціонального тестування перевірено коректність обробки руху персонажа, зіткнень зі стінами, активації цільових точок, роботи інтерфейсу та загальну стабільність системи. Тестування підтвердило повну працездатність та стабільність розробленого програмного забезпечення.

Практична цінність результатів полягає у можливості використання гри як готового шаблону 2D-проєктів на Unity. Перспективами розвитку проєкту є інтеграція системи динамічного збереження прогресу, додавання звукового супроводу, розширення бази рівнів та розробка процедурного генератора лабіринтів.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Salen K., Zimmerman E. Rules of Play: Game Design Fundamentals. MIT Press, 2003. 688 p.
2. Brathwaite B., Schreiber I. Challenges for Game Designers. Course Technology, 2009. 317 p.
3. Hocking J. Unity in Action: Multiplatform Game Development in C#. Manning Publications, 2018. 352 p. URL: https://www.manning.com/books/unity-in-action
4. Bond J. Introduction to Game Design, Prototyping, and Development. Addison-Wesley, 2021. 1296 p.
5. How Do Video Games Work? Capstone Classroom, 2013. 48 p.
6. Johnson L. L., DeBoeser E. Review of Inside, Playdead. Journal of Adolescent & Adult Literacy. 2017. P. 340—341.
7. Професія «Геймдизайнер»: хто це, чим займається і як ним стати. URL: https://vokigames.com/ua/profesiya-gejmdyzajner-hto-cze-chym-zajmayetsya-i-yak-nym-staty/
8. Bates B. Game Design. Thomson Course Technology, 2004. 350 p.
9. Moore M. E., Novak J. Game Industry Career Guide. Delmar: Cengage Learning, 2010. 320 p. URL: https://archive.org/details/isbn_2901428376471
10. Evans R., Rabin S. AI Game Programming Wisdom. Charles River Media, 2002. 672 p.
11. Хто такий tester. URL: https://qalight.ua/baza-znaniy/hto-takyj-tester/
12. What Is Puzzle Games? A Complete Guide to This Popular Gaming Genre. URL: https://cone-ing.com/what-is-puzzle-games/
13. Класифікація головоломок. URL: https://cikavamatematuka.blogspot.com/p/page_16.html
14. Puzzle Game Categories Guide: Word, Logic, Jigsaw & More. URL: https://puzzle-game.io/puzzle-game-categories-guide

15. Schell J. The Art of Game Design: A Book of Lenses. 2nd ed. CRC Press, 2014. 600 p. URL: https://www.taylorfrancis.com/books/mono/10.1201/b17723/art-game-design-jesse-schell
16. Adams E. Fundamentals of Game Design. New Riders, 2014. 560 p. URL: https://www.academia.edu/81247135/Fundamentals_of_game_design
17. Millington I. Game Physics Engine Development. Morgan Kaufmann, 2007. 480 p. URL: https://www.sciencedirect.com/book/monograph/9780123694713/game-physics-engine-development
18. Nystrom R. Game Programming Patterns. Genever Benning, 2014. 354 p. URL: https://gameprogrammingpatterns.com/
19. Culberson J., Schaeffer J. Sokoban: Improving the Search with Relevance Cuts. *Theoretical Computer Science*. 2001. Vol. 252, No. 1–2. P. 151–175. URL: https://doi.org/10.1016/S0304-3975(00)00080-3
20. Junghanns A., Schaeffer J. Sokoban: Enhancing General Single-Agent Search Methods Using Domain Knowledge. Artificial Intelligence Journal. 2001. Vol. 129, No. 1–2. P. 219–251. URL: https://doi.org/10.1016/S0004-3702(01)00109-6
21. Culberson J. Sokoban is PSPACE-complete. University of Alberta, 1997. 13 p. URL: https://doi.org/10.7939/R3JM23K33
22. Bento D. S., Pereira A. G., Lelis L. H. S. Procedural Generation of Initial States of Sokoban. 2019. P. 4651-4657. URL: https://doi.org/10.24963/ijcai.2019/646
23. Top similar games like Sokoban (Boxman) Classic. URL: https://steampeek.hu/?from=36&appid=1406570
24. Що таке Unity? URL: https://www.tecnoloblog.com/uk/
25. Ward T. Using Unity to Help Solve Intelligence. 2020. URL: https://arxiv.org/abs/2011.09294
26. Goldstone W. Unity Game Development Essentials. Packt Publishing, 2013. 316 p.

27. Price M. C# 10 and .NET 6 – Modern Cross-Platform Development. Packt Publishing, 2021. 826 p.
28. Liberty J., MacDonald B. Learning C# by Developing Games with Unity. O'Reilly Media, 2021. 466 p.
29. Unity Technologies. Unity User Manual: Scripting Reference. San Francisco: Unity Technologies. URL: https://docs.unity3d.com/Manual/ScriptingSection.html
30. Troelsen A., Japikse P. Pro C# 9 with .NET 5: Foundational Principles and Practices. 10th ed. Berkeley, CA: Apress, 2021. 1011 p.
31. Skeet J. C# in Depth. 4th ed. Shelter Island, NY: Manning Publications, 2019. 528 p.
32. Gibson J. Introduction to Game Design, Prototyping, and Development with Unity and C#. 3rd ed. Boston, MA: Addison-Wesley Professional, 2022. 1216 p.
33. Albahari J. C# 9.0 in a Nutshell: The Definitive Reference. Sebastopol, CA: O'Reilly Media, 2021. 1056 p.
34. Pile John Jr. 2D Graphics Programming for Games. New York: CRC Press, 2010. 320 p.
35. Основи 2D-графіки. URL: https://gdansk.itstep.pl/basics-of-twod-graphics-tools-and-techniques-for-beginners
36. Rogers S. Level Up! The Guide to Great Video Game Design. Wiley, 2014. 560 p.
37. Novak J. Game Development Essentials: An Introduction. Delmar Cengage Learning, 2011. 510 p.
38. Azzi D. Pixel Art for Game Developers. CRC Press, 2020. 256 p.
39. Wolf M. J. P. The Video Game Explosion: A History from PONG to PlayStation and Beyond. Greenwood Press, 2008. 410 p. URL: https://archive.org/details/videogameexplosi0000unse
40. Williams R. The Animator's Survival Kit. London: Faber & Faber, 2001. 342 p.

ДОДАТКИ

ДОДАТОК А

Скрипт MovementPlayer.cs

```
using UnityEngine;
public class MovementPlayer : MonoBehaviour
{
    public float moveDistance = 1f;
    public LayerMask wallLayer;
    public LayerMask boxLayer;
    private bool isMoving = false;
    private Vector3 targetPosition;
    void Update()
    {
        if (isMoving) return;
        Vector3 dir = Vector3.zero;
        if (Input.GetKeyDown(KeyCode.W)) dir = Vector3.up;
        if (Input.GetKeyDown(KeyCode.S)) dir = Vector3.down;
        if (Input.GetKeyDown(KeyCode.A)) dir = Vector3.left;
        if (Input.GetKeyDown(KeyCode.D)) dir = Vector3.right;
        if (dir != Vector3.zero)
            TryMove(dir);
    }
    void TryMove(Vector3 dir)
    {
        Vector3 nextPos = transform.position + dir * moveDistance;
        if (Physics2D.OverlapCircle(nextPos, 0.2f, wallLayer))
            return;
        Collider2D box = Physics2D.OverlapCircle(nextPos, 0.2f, boxLayer);
        if (box != null)
        {
            Vector3 boxNextPos = nextPos + dir * moveDistance;
            if (Physics2D.OverlapCircle(boxNextPos, 0.2f, wallLayer) ||
                Physics2D.OverlapCircle(boxNextPos, 0.2f, boxLayer))
                return;
            box.transform.position = boxNextPos;
        }
        targetPosition = nextPos;
        StartCoroutine(Move());
    }
    System.Collections.IEnumerator Move()
    {
```

```
isMoving = true;
while ((targetPosition - transform.position).sqrMagnitude > 0.001f)
{
    transform.position = Vector3.MoveTowards(
        transform.position,
        targetPosition,
        10f * Time.deltaTime
    );
    yield return null;
}
transform.position = targetPosition;
isMoving = false;
```

ДОДАТОК Б

Скрипт GameManager.cs

```
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    [Header("Targets Settings")]
    public int totalTargets = 6;
    private int targetsAchieved = 0;

    [Header("UI Elements")]
    public GameObject winText;
    public TextMeshProUGUI timerText;

    [Header("Timer Settings")]
    public float timeRemaining = 60f;
    private bool isGameActive = true;

    private void Start()
    {
        if (winText != null) winText.SetActive(false);
    }

    private void Update()
    {
        if (isGameActive)
        {
            if (timeRemaining > 0)
            {
                timeRemaining -= Time.deltaTime;
                UpdateTimerDisplay(timeRemaining);
            }
            else
            {
                timeRemaining = 0;
                isGameActive = false;
                TimeOut();
            }
        }
    }
```

```
    public void TargetActivated()
    {
        if (!isGameActive) return;

        targetsAchieved++;
        if (targetsAchieved >= totalTargets)
        {
            isGameActive = false;
            Debug.Log("ПЕРЕМОГА! Всі коробки на місцях!");

            if (winText != null) winText.SetActive(true);
        }
    }

    public void TargetDeactivated()
    {
        if (!isGameActive) return;
        targetsAchieved--;
    }

    private void UpdateTimerDisplay(float timeToDisplay)
    {
        float minutes = Mathf.FloorToInt(timeToDisplay / 60);
        float seconds = Mathf.FloorToInt(timeToDisplay % 60);

        if (timerText != null)
        {
            timerText.text = string.Format("{0:00}:{1:00}", minutes, seconds);
        }
    }

    private void TimeOut()
    {
        Debug.Log("Час вийшов!");
        if (timerText != null)
        {
            timerText.text = "ЧАС ВИЙШОВ!";
            timerText.color = Color.red;
        }
    }
}
```