Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Огаркова Віктора Юрійовича*
(ПІБ)

академічної групи *121-20з-1*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка ігрового додатка на платформі Unity з використанням шаблонів і патернів проєктування*

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	*проф. Алексєєв М.О.*			
розділів:				
спеціальний	*проф. Алексєєв М.О.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-20з-1* (група) *Огаркова Віктора Юрійовича* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка ігрового додатка на платформі Unity з використанням шаблонів і патернів проєктування*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *470-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***15.05.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПО й тривалості його розробки***	***1.06.2024 р***

Завдання видав _____ *проф. Алексєєв М.О.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняла до виконання _____ *Огарков В.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*

Термін подання кваліфікаційної роботи до ЕК: _____

РЕФЕРАТ

Пояснювальна записка: 107 с., 15 рис., 3 дод., 26 джерел.

Об'єкт розробки: ігровий додаток на платформі Unity за допомогою C#

Мета кваліфікаційної роботи: розробка ігрового додатка з використанням шаблонів і патернів проєктування, а також сучасних методів написання програмного коду на основі принципів ООП і SOLID. Кінцева мета - створення готового ігрового застосунку, який буде задовольняти споживачів, розвивати їх аналітичні здібності та навички стратегічного мислення, а також забезпечувати динамічну і веселу взаємодію з грою. Гра буде стимулювати та вдосконалювати аналітичні здібності гравців та впливати на зменшення стресу під час проводження часу за вирішенням ігрових задач.

У вступі розглядається аналіз проблеми, вказується мета кваліфікаційної роботи і галузь її застосування, наводиться обґрунтування актуальності предмета, пояснюється формулювання завдання.

У першому розділі міститься огляд предметної області і доступних рішень, актуальність завдань і мета розробки, а також формулювання завдань.

Другий розділ включає вибір платформи розробки, проєктування та розробку додатка, опис алгоритму та функціональної структури додатка. Описуються технічні засоби, що використовуються для виконання завдання.

В економічній частині визначається вартість розробки програмного забезпечення, тобто розраховується вартість створення програми і оцінюється час, необхідний для її створення.

Практичне значення полягає у створенні ігрового додатку, який на етапі готового продукту буде задовольняти усім потребам користувача, і в подальшому забезпечуватиме структурований і масштабований код, що дасть змогу скоротити час розроблення нової ігрової логіки, підвищити якість коду та полегшити його підтримку.

Актуальність розробки обумовлена популярністю ігор як засобу розваги, що широко використовується людьми різного віку. З розвитком технологій та появою мобільних пристроїв ігри стають ще більш доступними та популярними, створюючи попит на нові та захоплюючі ігрові додатки.

Список ключових слів: UNITY, C#, ООП, SOLID, ПАТЕРНИ, ІГРОВИЙ ДОДАТОК, ІГРОВИЙ РУШІЙ, ПРОГРАМА, ПРОЄКТУВАННЯ.

ABSTRACT

Explanatory note: 64 p., 15 figures, 3 appendix, 26 sources.

Development object: game application on the Unity platform using C#

The purpose of the qualification work: development of a game application using design templates and patterns, as well as modern methods of writing program code based on the principles of OOP and SOLID. The ultimate goal is to create a ready-made game application that will satisfy consumers, develop their analytical and strategic thinking skills, and provide dynamic and fun interaction with the game. The game will stimulate and improve the analytical skills of players and help reduce stress while spending time solving game problems.

The introduction discusses the analysis of the problem, indicates the purpose of the qualification work and its field of application, provides a justification for the relevance of the subject, and explains the task formulation.

The first section provides an overview of the subject area and available solutions, the relevance of the tasks and the purpose of the development, as well as the task formulation.

The second section includes the selection of the development platform, design and development of the application, description of the algorithm and functional structure of the application. The technical means used to perform the task are described.

The economic part determines the cost of software development, i.e. calculates the cost of creating an application and estimates the time required to create it.

The practical significance lies in the creation of a game application that, at the stage of the finished product, will satisfy all user needs and, in the future, will provide structured and scalable code, which will reduce the time for developing new game logic, improve the quality of the code and facilitate its maintenance.

The relevance of the development is driven by the popularity of games as a means of entertainment that is widely used by people of all ages. With the development of technology and the emergence of mobile devices, games are becoming even more accessible and popular, creating demand for new and exciting gaming applications.

List of keywords: UNITY, C#, OOP, SOLID, PATTERNS, GAME APPLICATION, GAME ENGINE, PROGRAMME, DESIGN.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GUI – Graphical User Interface;

IT – Information Technology;

JSON – JavaScript Object Notation;

MVC – Model-View-Controller;

SOLID – (Принципи дизайну об'єктно-орієнтованого програмування);

UI – User Interface ;

ООП – Об'єктно-орієнтоване програмування;

ОС – Операційна система;

ПЗ – Програмне забезпечення.

ЗМІСТ

ВСТУП

Ігрова індустрія - це не лише форма розваги, але й значна галузь економіки. Вона представляє собою великий бізнес, який зростає та розвивається з кожним роком. В Україні ігрова індустрія також має величезний вплив і приносить значний прибуток. Тому на сьогодення розробка і покращення ігрових застосунків є релевантною задачею для програмістів. IT галузь набрала дуже швидких темпів розвитку технологій, неминаючі і геймдев. Великим значенням актуальності теми також є використання геймдев технологій у інших сферах: освіта, медицина, військова справа, бізнес та наукові дослідження.

Значна частина процесів у виробництві ігрового додатку пов'язана із розробкою та застосуванням функціоналу, який забезпечить гравцеві приємний ігровий досвід і відповідатиме побажанням користувача. Процес розробки включає в себе проєктування архітектури майбутнього програмного коду, створення різних ігрових механік, включаючи інтерфейс та звуковий супровід. Окрім цього, важлива оптимізація продукту для різних платформ, тестування. Таким чином, постає завдання досягнути ідеального балансу між потребами кінцевого користувача і використання доступних ресурсів.

Для оптимальної витрати часу та коштів, які є обмеженими ресурсами, першим і головним завданням є розробка дизайну проєкта. Це важливий етап, на якому визначаються основні аспекти майбутнього гри, що включають в себе ігрову механіку, системи GUI та UI, архітектуру програмного забезпечення, додаткові фреймворки та бібліотеки, і багато іншого. Перш за все це дає розуміння рамок проєкту і можливість сконцентруватися на виконанні певних завдань, а також дозволяє уявити, як гра буде працювати, взаємодіяти з користувачем і які додаткові засоби знадобляться під час розробки. Ці етапи дозволяють розподілити всі аспекти розробки гри перед початком, що зменшує ризики помилок та дозволяє ефективніше використовувати ресурси.

Мета цієї кваліфікаційної роботи є подвійною: демонстрація принципів ООП, SOLID та патернів проєктування, що застосовуються при розробці ігор в Unity, доведення їхньої ефективності для покращення архітектури ігор, масштабованості та супроводжуваності коду, а також розробка готового ігрового додатку, який надає можливість отримання від проводження у грі гарного користувацького досвіду, зниження рівня стресу та сприяння когнітивному розвитку, включаючи покращення швидкості прийняття рішень.

Результатом виконання даної роботи є ігровий додаток, який дозволить користувачеві зняти стрес, зосередитися на розробці стратегії та тактичних здібностях, а також знайти найкращий спосіб виконання поставлених цілей. Завдяки використанню шаблонів і патернів проєктування, додаток матиме змогу для легкого розширення із застосуванням малих ресурсів для підтримки зацікавленості користувачів до ігрового процесу.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

Відеоігри сягають своїм корінням від примітивних текстових пригод, в яких гравці, використовуючи лише свою уяву та письмові команди, досліджували віртуальний світ. Однак, з часом індустрія перетворилася на гіганта, який охоплює велике різноманіття жанрів, занурюючи гравців у нові світи наповнені приголомшливою графікою, заплутаними сюжетами та безмежними можливостями. Сьогодні люди грають не лише заради розваги. Новітнє покоління геймерів - це не просто нація чи навіть континент, а глобальне явище. Еволюція історії ігор - це задумливий вираз людської праці, творчості та невпинної потреби в інноваціях. Від простих часів ігрових автоматів до складної ери домашніх консолей та появи багатокористувацьких онлайн-ігор - це короткий опис вічного прагнення людства до досконалості.

У сучасному цифровому світі індустрія відеоігор є досить великою та мінливою галуззю, де розробники повинні постійно пристосовуватися до нових технологій та вподобань геймерів. У цьому контексті застосування об'єктно-орієнтованого програмування (ООП), принципів SOLID, а також патернів проєктування виявляється ключовим фактором для розробки надійних, розширюваних і впорядкованих відеоігор.

Внесок розробки відеоігор не обмежується лише розвагами, а охоплює ширший спектр сфер, які включають освіту і симуляцію реальних робочих процесів. Ігри, як експериментальні процеси, покликані давати складні завдання, які захоплюють, залучають, а також розважають гравців. Щоб реалізувати цей потенціал, розробники повинні використовувати структурований підхід та досвід.

Ігрова індустрія тісно пов'язана з платформою розробки Unity, і вона не може бути нічим іншим, як частиною її історії. Основою платформи є пробна версія ігрового рушія, яка була створена у 2005 році. З моменту свого створення Unity зазнала прогресивних оновлень та оптимізації, що свідчить про її надійність, гнучкість та зручне середовище для розробників ігор. Те, що починалося як невеликий проєкт, зараз є одним з найбільш зв'язаних і багатоцільових фреймворків розробки, що використовуються для створення ігор. Основна мова програмування, яка використовується в Unity, є C#.

Unity відомий своєю гнучкістю та ефективністю, зараз є найкращим вибором для розробників ігор у всьому світі. Його комплексний набір інструментів у поєднанні з передовими функціями, такими як ООП та шаблони проєктування, дозволяє розробникам безперешкодно розробляти та розгортати свої ігри на різних платформах.

Дотримання таких підходів програмування дає розробникам можливість розбити ігровий функціонал на модульні компоненти, які можна повторно використовувати, щоб мати змогу без проблем повторювати та модифікувати код.

Застосування патернів проєктування гарантує використання перевірених методів для вирішення поширених дизайнерських головоломок, що допомагають побудувати концепцію фундаментальної та стабільної ігрової архітектури. З часом стало очевидним, що архітектура відеоігрових додатків використовує принципи ООП, SOLID і патерни дизайну, щоб створити якісні та довгострокові ігрові продукти.

1.2 Призначення розробки та галузь застосування

Розвиток мобільних технологій призвів до значного зростання популярності мобільних ігор. Зокрема, головоломки, такі як п'ятнашки,

залишаються затребуваними серед користувачів різного віку завдяки їх простоті та здатності тренувати логічне мислення. Розробка такої гри вимагає чіткого розуміння архітектури системи, алгоритмів функціонування та оптимізації роботи на мобільних пристроях.

Основною метою ігрового застосунку “Puzzle & Play” є створення захоплюючого ігрового досвіду для гравців. Гра призначена для зняття стресу, отримання насолоди від ігрового процесу та відпочинку від щоденних турбот.

Крім того, “Puzzle & Play” спрямована на розвиток когнітивних здібностей гравців. Головна мета гри – вирішення головоломки за відведений час. Гра включає три рівні складності, що дає змогу поступово розвиватися, а логіка гри розроблена таким чином, що навіть з часом можна отримувати насолоду від нескінченного процесу.

1.3 Підстави для розробки

Підставою для розробки є вимоги освітньої програми, відповідно до яких у кінці навчання студент зобов'язаний виконати кваліфікаційну роботу відповідно до навчального плану та графіків навчального процесу.

Тема роботи узгоджується з керівником проєкту, випускаючою кафедрою,

та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

− освітня програма 121 «Інженерія програмного забезпечення»;

− навчальний план та графік навчального процесу;

− наказ ректора Національного технічного університету «Дніпровська

політехніка» № 470-с від 23.05.2024 р;

− завдання на кваліфікаційну роботу на тему “Розробка ігрового додатку на платформі Unity з використанням шаблонів і патернів проєктування».

1.4 Постановка завдання

“Puzzle & Play” – це класична головоломка «П’ятнашки», адаптована для мобільних пристроїв. Гравець має пересувати плитки на ігровому полі розміром 3x3 (або іншого розміру) таким чином, щоб досягти впорядкованого стану, де всі плитки розташовані за зразком. Остання клітинка на полі залишається пустою, що дозволяє пересувати плитки.

Гра пропонує кілька рівнів складності. Вона розроблена з урахуванням зручності використання на сенсорних екранах і оптимізована для різних розмірів екранів мобільних пристроїв.

Головні функції гравця:

- налаштування гри;
- розпочинання вибраної гри;
- фунція паузи;
- дострокове закінчення гри та вихід у головне меню.

Ігровий процес:

- основною метою гравця є вирішення головоломки за відведений час.

анімації та звукові ефекти:

- в грі присутні звукові ефекти та фонова музика, яка змінюється в залежності від стану гри;
- в грі застосовується анімації, які правильно налаштовані і додають більшу інтерактивність.

Цільова платформа:

- Android

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Головною метою будь-якої розробки є забезпечення високої якості вихідного продукту. Це можливо лише за умов дотримання вимог функціональних характеристик, а саме:

- гра повинна працювати без будь-яких системних збоїв та помилок;
- потрібне доцільне використання програмних та системних ресурсів для забезпечення плавної гри;
- звуковий супровід повинен бути якісним і підходити до атмосфери гри, підсилюючи враження від геймплею;
- у грі повинен бути правильно налаштований баланс, що підтримуватиме інтерес гравця;
- інтерфейс повинен бути інтуїтивно зрозумілим і зручним для користувача;

- в грі повинна бути автоматична система збереження прогресу гравця;
- готовий продукт повинен підтримувати різні платформи і відповідати їх можливостям.

1.5.2. Вимоги до інформаційної безпеки

Інформаційна безпека є важливим аспектом в розробці будь-якого програмного застосунку, і є основні вимогу, які повинні виконуватися:

1. ігровий застосунок повинен забезпечувати надійне зберігання та захист будь-якою особистої інформації користувачів, а також бути захищеним від несанкціонованого доступу та витоку даних;
2. всі ігрові дані повинні бути зашифровані, для того, щоб у разі перехоплення або викрадення, захистити чутливу інформацію користувача;
3. ігровий застосунок повинен бути спроєктований таким чином, щоб мінімізувати вплив шкідливих програм і вірусів;
4. доцільним буде збереження цілісності даних у випадку непередбачуваного завершення застосунку. Всі чутливі дані повинні зберігатися автоматично.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для ігрового додатку передбачені певні мінімальні технічні характеристики та засоби.

- ОС: Android 5.1 або вище;
- Оперативна пам’ять: 2 ГБ.

1.5.4. Вимоги до інформаційної та програмної сумісності

Гра повинна бути сумісною з платформами, на яких вона буде запущена. Також вона повинна бути оптимізована для працездатності на різних типах апаратних пристроїв.

Незалежно від характеристик пристроїв, відображення та ігровий досвід повинен бути однаковим.

Незважаючи на версію ОС Android, ігровий додаток повинен бути сумісним з будь-якою версією, яка відповідає мінімальним характеристикам застосунку.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення інформаційної системи

Ця кваліфікаційна робота має на меті створення гри "Puzzle & Play", яка є віртуальною версією класичної головоломки, подібного до гри "П'ятнашки". Основне призначення цієї гри - надати гравцям можливість розважитися та відпочити, граючи у захоплюючі головоломки. Основні елементи гри та їх функціональне призначення включають:

- система пазлів - це специфічна система взаємодії між розташованими на полі гри фішками, яка створює геймплей, характерний для ігор у жанрі пазлів;
- механіка переміщення фішок дозволяє гравцеві переміщувати фішки по полю гри, щоб розташувати їх у вірному порядку, подібно до гри "П'ятнашки";
- розроблений графічний інтерфейс надає гравцеві можливість легко взаємодіяти з грою, включаючи можливість переміщати фішки та переглядати їх поточне розташування;
- система управління грою дозволяє гравцеві легко перетягувати фішки за допомогою дотику на екрані сенсорного пристрою, щоб керувати грою;
- система відліку часу у грі, яка обмежує кількість часу для вирішення головоломки.

Крім основних елементів гри, також присутні інші можливості. Гравця зустрічає головне меню, в якому він може налаштувати гру під свої потреби, включаючи зміну параметрів звуку, музики та тактильного відгуку, а також вибрати складність пазлу.

2.2. Опис застосованих математичних методів

При розробці цієї ігрової програми використовувались лише прості арифметичні методи та стандартна математична бібліотека Unity Mathf.

З використаних методів бібліотеки Mathf можна відзначити дві функції: Clamp та FloorToInt.

Clamp – це функція, яка обмежує значення числа в певному діапазоні. Вона приймає три аргументи: значення, яке потрібно обмежити, мінімальне значення діапазону та максимальне значення діапазону. Якщо передане значення менше мінімального, воно буде замінено на мінімальне значення. Якщо передане значення більше максимального, воно буде замінено на максимальне значення. Якщо значення знаходиться в межах діапазону, воно залишиться без змін. Цей метод дуже корисний для обмеження значень під час роботи з рухомими об'єктами, параметрами користувача та іншими значеннями в ігровому процесі.

Метод FloorToInt в Unity виконує округлення числа до найближчого меншого цілого числа і повертає його у вигляді цілого числа. Цей метод корисний для ситуацій, коли потрібно отримати ціле число, менше або рівне вхідному значенню, зокрема, при роботі з позиціями об'єктів в ігровому просторі або при роботі зі змінними, що потребують цілих значень.

2.3. Опис використаних технологій та мов програмування

Для розробки додатку використовується рушій Unity.

Unity є одним з найпопулярніших ігрових двигунів, що використовується для створення 2D та 3D ігор. Завдяки своїй потужності та гнучкості, Unity здобув

визнання серед розробників усього світу. В цьому тексті розглянемо ключові переваги та недоліки Unity, а також його вплив на процес розробки ігор.

Для Unity характерні такі переваги:

- Unity підтримує створення ігор для різних платформ, включаючи ПК, мобільні пристрої (iOS, Android), консолі (PlayStation, Xbox), веб-браузери та інші. Це дозволяє розробникам одночасно охоплювати широку аудиторію без необхідності значних зусиль на портування;
- Unity має інтуїтивний інтерфейс та відносно низький поріг входу для новачків. Це забезпечується великим обсягом документації, навчальних матеріалів та активною спільнотою, яка готова допомогти у разі виникнення питань;
- Unity надає розробникам доступ до потужних інструментів для роботи з графікою, фізикою, анімацією та штучним інтелектом. Це дозволяє створювати високоякісні ігрові продукти, не вдаючись до сторонніх бібліотек;
- Unity підходить як для невеликих інді-проєктів, так і для великих комерційних ігор. Це дозволяє команді розробників зростати та розширювати свої можливості разом з проєктом.

Проте як й інші продукти, Unity має свої недоліки:

- Unity може бути досить вимогливим до апаратних ресурсів, особливо при роботі з великими проєктами та високоякісною графікою. Це може призводити до зниження продуктивності, особливо на старіших чи менш потужних пристроях;

- хоча Unity пропонує безкоштовну версію для особистого використання та невеликих проєктів, для комерційних цілей та більших проєктів необхідна платна ліцензія, яка може бути досить дорогою;

- незважаючи на потужні інструменти, доступні в Unity, процес оптимізації гри для досягнення високої продуктивності може бути складним і трудомістким. Це особливо актуально для мобільних платформ та VR;
- Unity добре підходить для 2D-ігор, деякі розробники вважають, що спеціалізовані 2D-двигуни, такі як Godot або Construct, можуть надавати більш ефективні рішення та інструменти для створення 2D-контенту.

Основною мовою програмування в середовище Unity є C#.

Мова програмування C# - це потужна об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона була вперше представлена у 2000 році як частина ініціативи Microsoft .NET Framework. C# став важливою мовою для розробки різних програмних продуктів, включаючи веб-додатки, мобільні додатки та ігри.

Серед переваг C# можна відзначити:

- синтаксис C# є досить зрозумілим та легким для вивчення, що робить його відмінним вибором для початківців та досвідчених розробників;
- C# підтримує об'єктно-орієнтовану парадигму програмування, що сприяє розробці програмного забезпечення, яке є більш структурованим, легким у розумінні та підтримці;
- C# застосовується в багатьох галузях розробки програмного забезпечення, включаючи веб-розробку, мобільну розробку, розробку ігор, десктопні додатки та багато іншого;
- мова C# має глибоку інтеграцію з іншими продуктами та технологіями Microsoft, такими як .NET Framework, ASP.NET, WPF, Xamarin та інші.

Для створення системи збережень був використаний JavaScript Object Notation.

JSON є ефективним та простим у використанні форматом обміну даними, який пропонує багато переваг, таких як читабельність, міжмовна сумісність та компактність. Проте, він також має свої недоліки, включаючи відсутність вбудованих схем, обмежену підтримку типів даних та питання безпеки. Незважаючи на ці недоліки, JSON все ще залишається одним з найпопулярніших форматів обміну даними завдяки своїй гнучкості та широкій підтримці різних мов програмування.

Одна з основнихособливостей Unity є використання компонентної парадигми, яка відрізняється від традиційної об'єктно-орієнтованої парадигми (ООП). Проте, обидві парадигми можуть ефективно співіснувати, підсилюючи розробку та створення складних ігрових проєктів.

В основі Unity лежить компонентна архітектура, яка дозволяє створювати складні об'єкти шляхом комбінування простих, незалежних компонентів. У компонентній парадигмі основною одиницею є "компонент", який надає певну функціональність об'єкту. Це окремі модулі, які додаються до ігрових об'єктів. Кожен компонент виконує конкретну функцію, наприклад, управління фізикою, анімацією або взаємодією з користувачем. ігровий об’єкт в Unity - це контейнер, до якого можуть бути додані різні компоненти. Ігрові об'єкти без компонентів не мають поведінки або властивостей.

Компонентна парадигма дозволяє легше створювати та змінювати об'єкти, оскільки функціональність можна додавати або видаляти через додавання або видалення компонентів. Це підвищує гнучкість та повторне використання коду.

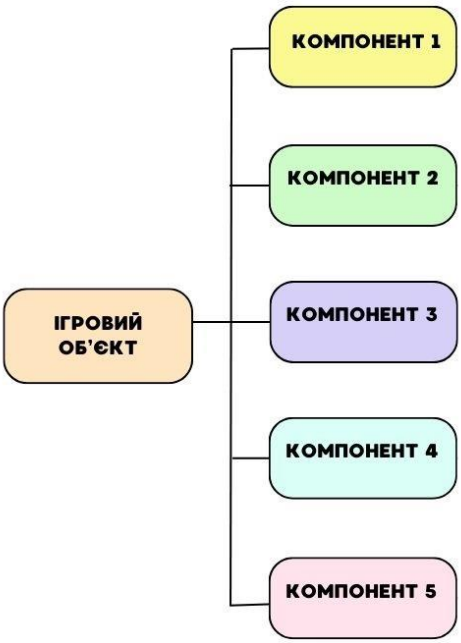

Рис. 2.1. Схема компонентної архітектури Unity

Хоча Unity значною мірою базується на компонентній парадигмі, об'єктно-орієнтоване програмування (ООП) також відіграє важливу роль у розробці ігор. ООП дозволяє структурувати код, використовуючи класи та об'єкти, що сприяє його підтримуваності та масштабованості.

Комбінування компонентної парадигми та ООП дозволяє отримати переваги обох підходів. Компонентна система забезпечує гнучкість та модульність, тоді як ООП сприяє структурованості та повторному використанню коду.

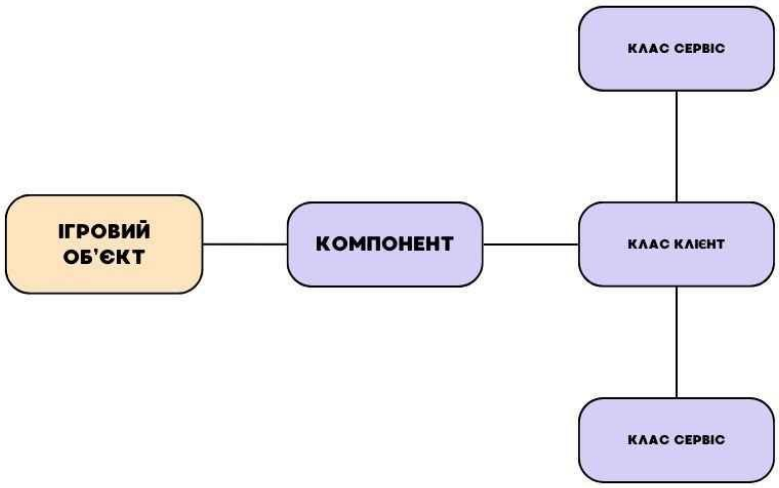

Рис. 2.2. Схема взаємодії класів і компонентів в Unity

В процесі розробки архітектури ігрового застосунку були використанні шаблони і патерни проєктування, а саме Singleton, MVC, Factory, Strategy, Observer та Iterator.

Патерн Singleton забезпечує існування лише одного екземпляра класу та надає глобальну точку доступу до цього екземпляра. Цей патерн часто використовується для управління ресурсами, які повинні бути унікальними в межах програми, наприклад, підключення до бази даних або конфігураційні налаштування. Використання Singleton гарантує, що доступ до певного ресурсу буде скоординованим і контрольованим.

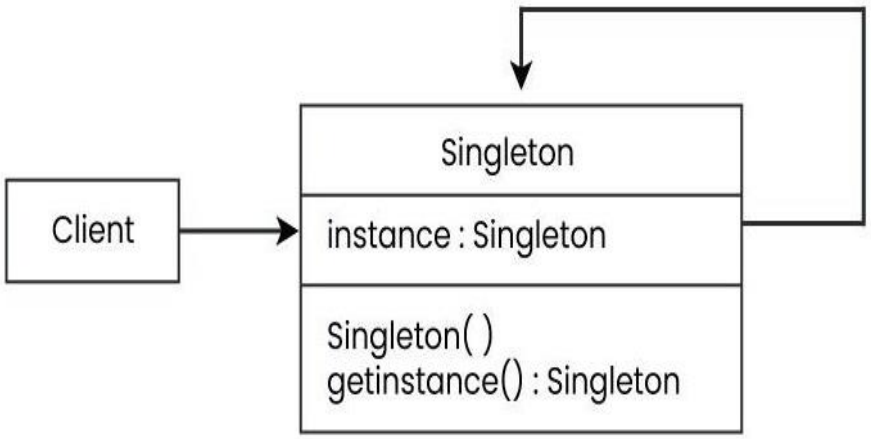

Рис. 2.3. Діаграма класу Singleton

Патерн MVC (Model – View – Controller) розділяє додаток на три основні компоненти: модель, представлення та контролер. Це дозволяє розділити логіку додатка від його користувацького інтерфейсу, що спрощує розробку, тестування та підтримку коду.

- Model: Управляє даними і логікою додатка, зберігає стан.
- View: надає користувачеві інтерфейс для взаємодії, відповідальний за відображення даних.
- Controller: Приймає та обробляє введені користувачем дані і передає дані моделі та представленню, забезпечуючи зв'язок між ними.

Переваги MVC включають покращену організацію коду та легкість внесення змін до окремих компонентів без впливу на інші.

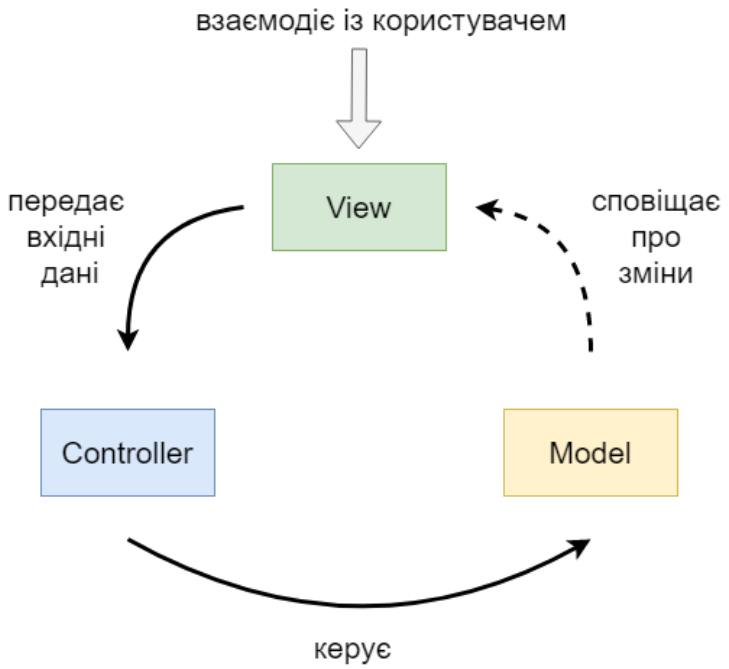

Рис. 2.4. Діаграма взаємодії MVC компонентів

Патерн Factory використовується для створення об'єктів без вказання точної класу створюваного об'єкта. Це дозволяє створювати об'єкти в залежності від ситуації, що спрощує процес створення нових об'єктів і підвищує гнучкість коду. Основна ідея полягає в тому, щоб використовувати загальний інтерфейс для створення об'єктів, що дозволяє змінювати тип створюваних об'єктів без змін в клієнтському коді.

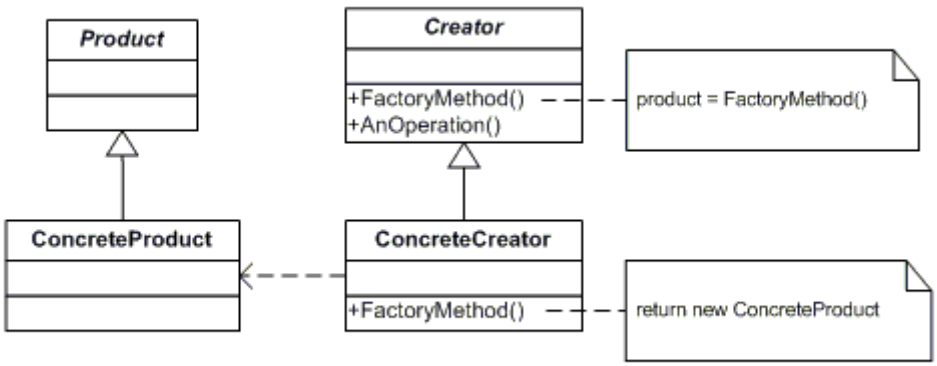

Рис. 2.5. Діаграма класу Factory

Патерн Strategy дозволяє визначати сімейство алгоритмів, інкапсулювати кожний з них і робити їх взаємозамінними. Strategy дозволяє змінювати алгоритми незалежно від клієнтів, які ними користуються. Це особливо корисно, коли є кілька варіантів поведінки і необхідно вибрати між ними під час виконання програми.

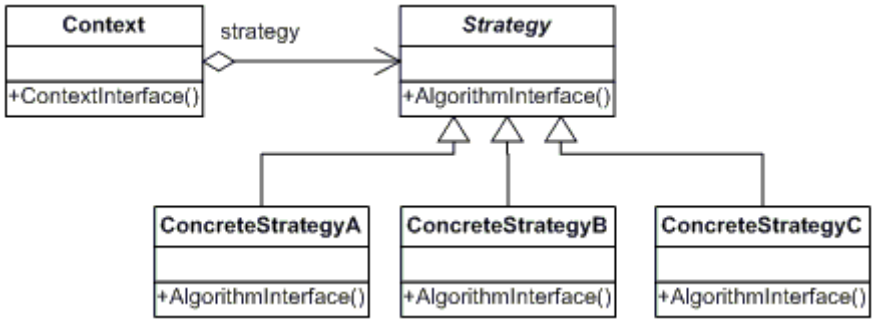

Рис. 2.6. Діаграма класу Strategy

Патерн Observer визначає взаємозв'язки "один до багатьох" між об'єктами так, що при зміні стану одного об'єкта всі залежні від нього об'єкти повідомляються та автоматично оновлюються. Це дозволяє розробити систему з динамічними залежностями, що дозволяє автоматично враховувати зміни в одному модулі іншими модулями.

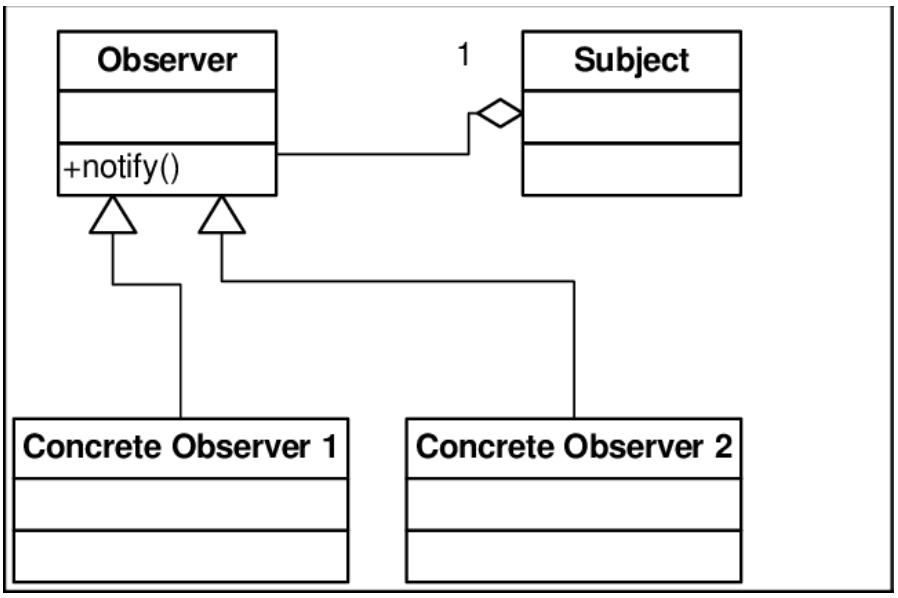

Рис. 2.7. Діаграма класу Observer

Патерн Iterator надає спосіб послідовного доступу до елементів агрегованого об'єкта без розкриття його внутрішньої структури. Використання ітераторів дозволяє абстрагувати процес перебору елементів, що забезпечує більш гнучкий і узгоджений спосіб роботи з різними колекціями об'єктів.

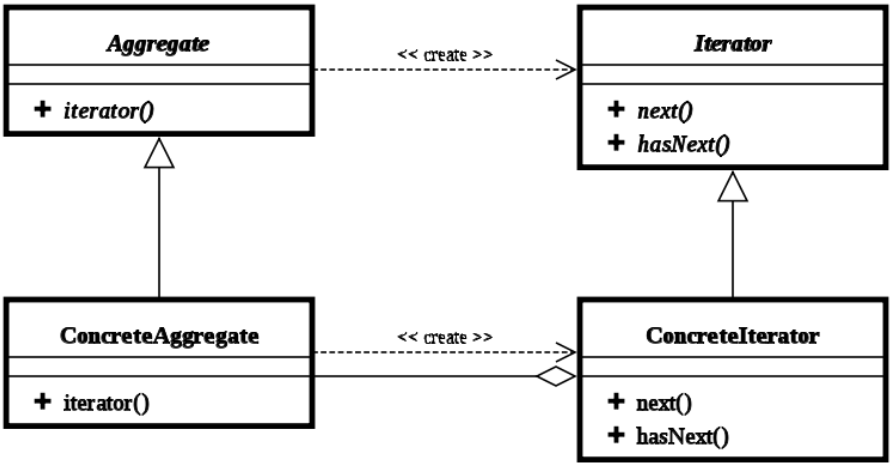

Рис. 2.8. Діаграма класу Iterator

Використання цих патернів проєктування допомагає створювати більш структурований, зрозумілий та підтримуваний код.

Крім того, в Unity використовуються додаткові сторонні технології, такі як DOTween та TextMeshPro, що дозволяють ще більше розширити функціональність ігрового рушія.

DOTween є потужною бібліотекою для анімації в Unity, яка дозволяє легко створювати анімаційні ефекти для ігрових об'єктів. Вона підтримує плавні анімації для позицій, обертання, масштабу та інших властивостей об'єктів.

DOTween пропонує простий та інтуїтивно зрозумілий синтаксис для створення анімацій. Це дозволяє швидко реалізовувати складні анімаційні ефекти без необхідності писати великий обсяг коду.

Бібліотека оптимізована для високої продуктивності, що дозволяє використовувати її навіть у проєктах з високими вимогами до швидкості виконання.

DOTween підтримує широкий спектр функцій, включаючи послідовності анімацій, паралельні анімації, циклічні анімації та багато іншого.

TextMeshPro є розширенням для роботи з текстом у Unity, що надає розширені можливості форматування і відображення тексту. Він забезпечує високу якість відображення тексту завдяки використанню векторної графіки, що дозволяє уникнути пікселізації при масштабуванні. Також пропонує широкий спектр налаштувань для тексту, включаючи шрифти, стилі, вирівнювання, інтервал між символами та інше. Розширення оптимізоване для високої продуктивності, що дозволяє використовувати його в іграх з великою кількістю тексту без значного впливу на продуктивність.

2.4. Опис структури системи та алгоритмів її функціонування

Unity є потужною платформою для розробки ігор, яка забезпечує чітку структуру проєкту, що допомагає ефективно управляти ресурсами та налаштуваннями.

Ключовою папкою в структурі проєкту є Assets. Вона зберігає в собі всі ресурси, які використовуються під час розробки. В папці можна знайти сцени, скрипти, текстури, моделі, аудіофайли, анімації та інші ресурси, необхідні для гри.

Папка Library є однією з найважливіших у проєкті Unity. Вона містить згенеровані Unity файли, які використовуються для оптимізації та швидкого доступу до ресурсів проєкту. Вона зберігає кешовані імпортовані ресурси (текстури, моделі, аудіофайли тощо), включає дані про конфігурації та налаштування проєкту, згенеровані під час його роботи та забезпечує швидке завантаження проєкту, оскільки дозволяє уникнути повторного імпортування ресурсів при кожному відкритті проєкту.

Папка Logs містить файли журналів, які зберігають інформацію про роботу Unity та його компонентів. Вона зберігає інформацію про помилки, попередження та інші повідомлення, що виникають під час роботи Unity, допомагає розробникам відстежувати проблеми та діагностувати помилки в проєкті, а також містить файли логів як редактора Unity, так і збірок гри.

Папка Obj використовується Unity для зберігання згенерованих файлів під час компіляції скриптів та збірки проєкту. Папка містить тимчасові файли, які створюються під час компіляції скриптів, що забезпечує взаємодію між Unity та середовищем розробки (наприклад, Visual Studio) та полегшує процес відладки та налагодження коду.

Папка Packages містить інформацію про пакети, які використовуються у проєкті, включаючи як стандартні, так і сторонні пакети. В папці зберігаються файли конфігурації пакетів та метадані. Також вона управляє версіями пакетів, використаних у проєкті та забезпечує легкий доступ до бібліотек та інструментів, які розширюють функціонал Unity.

Папка ProjectSettings містить налаштування проєкту, які визначають поведінку та конфігурацію Unity для конкретного проєкту. В ній зберігаються файли налаштувань, такі як налаштування графіки, фізики, вводу, аудіо та інші, включаючі файли конфігурації для платформи збірки (iOS, Android, Windows тощо), що дозволяє централізовано керувати всіма аспектами налаштувань проєкту.

Папка UserSettings містить налаштування користувача, які специфічні для конкретного користувача або розробника, який працює над проєктом. Вона зберігає особисті налаштування редактора Unity, такі як макети вікон, налаштування теми, гарячі клавіші та інші персональні параметри, що дозволяє кожному розробнику налаштовувати середовище розробки під свої потреби без впливу на загальні налаштування проєкту та забезпечити збереження індивідуальних налаштувань між сеансами роботи з Unity.

Файл Diploma.sln є рішенням для інтегрованого середовища розробки Visual Studio. Цей файл створюється автоматично під час створення проєкту і містить інформацію про структуру та налаштування проєкту.

Файл Diploma.sln є важливим компонентом проєкту, який використовується для організації, управління та налаштування розробки в середовищі Visual Studio.

Організація проєкту в Unity передбачає наявність багатьох спеціалізованих папок, кожна з яких виконує важливу роль у забезпеченні ефективної розробки та оптимізації гри.

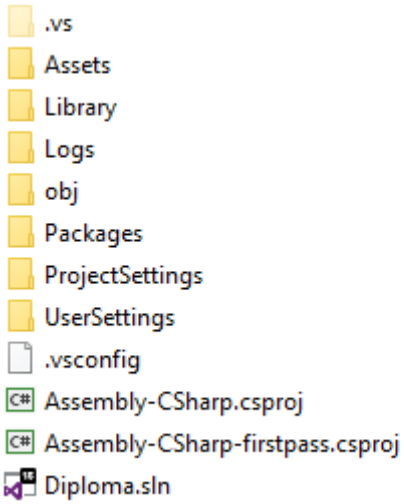

Рис. 2.9. Коренева система проєкту Unity

Основною частину ігрового застосунку є скріпти, які собою уявлять, або складаються в окремі компоненти, котрі приєднуються до ігрових об’єктів, і тим самим виконують свій функціонал.

В “Puzzle & Play” скріпти можно поділити на основні, які відповідають за ігрову логіку та допоміжні, котрі використовуються для налаштування правильної роботи користувацького інтерфейсу, і пов’язують всі алгоритми в одну єдину програму.

Класи, що реалізовані за допомогою патерну проєктування MVC (модель, вигляд, контролер), відповідають за ігрову логіку головоломки в грі.

Клас `PuzzleModel` керує моделлю головоломки, управляючи ініціалізацією, переміщенням елементів і перевіркою на вирішення головоломки. Він ініціалізує матриці для цільового та поточного стану головоломки, забезпечує анімацію та звукові ефекти під час переміщення елементів, а також сповіщає про вирішення головоломки через подію OnPuzzleSolved.

Клас PuzzleView відповідає за графічне відображення головоломки, керуючи створенням та налаштуванням полів для гравця та цільового зразка. Він здійснює анімацію переміщення елементів за допомогою бібліотеки DOTween, що дозволяє плавно анімувати рух клітинок на полі головоломки, та відображає символи у клітинках відповідно до поточного стану головоломки.

Клас PuzzleController відповідає за взаємодію користувача з графічною головоломкою. Він обробляє кліки користувача на полі головоломки, перевіряє можливість взаємодії з головоломкою залежно від поточного стану і відсутності анімаційного переміщення елементів. Клас використовує Unity для отримання координат кліків і їх перетворення на рядок і стовпчик у матриці головоломки, передаючи ці дані до моделі для виконання відповідних дій.

Клас PuzzleUtils містить статичний метод IsAdjacentToEmptyCell, який визначає, чи є клітина у матриці головоломки поруч з порожньою клітиною. Метод повертає “true”, якщо така порожня клітина знаходиться поруч з вказаною клітиною, і записує її координати у вихідні параметри. Якщо порожня клітина не знайдена, метод повертає “false”.

Клас DefaultShuffleStrategy реалізує інтерфейс IShuffleStrategy і забезпечує стратегію випадкового перемішування елементів у головоломці. Він автоматизує переміщення елементів за задану кількість кроків з певним інтервалом часу для забезпечення плавної анімації. Клас використовує рандомізацію напрямків переміщення і уникнення повторних ходів в тому ж напрямку.

Клас PuzzleMatrixIterator реалізує інтерфейси IEnumerator<int> і IEnumerator для ітерації по всіх елементах двовимірного масиву. Це забезпечує можливість послідовного доступу до кожного елемента матриці без прямого доступу до її індексів.

Клас PuzzleMatrixFactory створює матриці для головоломки. Він має методи для створення цільової матриці головоломки і матриці гравця, використовуючи рандомізацію і доступні елементи для генерації початкових станів.

Клас PuzzleTimer відповідає за логіку ведення та відображення таймера у грі. Він зменшує час і сповіщає про наближення до завершення таймера через відповідні події.

Клас GameController відповідає за керування грою і взаємодію з іншими компонентами, що включають модель головоломки, таймер і спливаючі вікна.

Він відповідає за такі функції:

- встановлення обробників подій;
- викликає відповідні функції для початку гри;
- відповідає за стан гри.

Цей клас відіграє важливу роль у керуванні ігровим процесом і взаємодії з графічним ігровим інтерфейсом та логікою головоломки.

Застосування патерну MVC дозволяє чітко розділити логіку гри на компоненти, що мають чіткі відповідальності. Модель управляє даними і бізнес-логікою, вигляд забезпечує їх візуалізацію і взаємодію з користувачем, а контролер обробляє введення користувача і керує потоком даних між моделлю та виглядом.

Такий підхід спрощує розробку, підтримку і розширення ігрового додатка, оскільки кожен компонент відповідає за конкретний аспект функціоналу і може бути модифікований або замінений без впливу на інші частини системи.

Також варто згадати про додаткові класи, але не менш важливі, такі як SoundController та DataStorageSystem.

Клас SoundController управляє аудіофункціоналом. Він ініціалізує та керує AudioSource для фонової музики та звукових ефектів. Методи цього класу дозволяють встановлювати, відтворювати, призупиняти, вимикати і включати звук, а також відтворювати звукові ефекти з опціональним відтворенням тактильного зворотного зв'язку.

DataStorageSystem це система збереження даних, яка складається з декількох взаємопов’язаних класів.

Клас JSONDataStorageSystem<T> є загальним інструментом для зберігання даних, який використовує серіалізацію у форматі JSON та шифрування в середовищі. Він реалізує інтерфейс IDataStorageSystem<T>, який визначає методи для зберігання та завантаження даних типу T.

Клас DataManager відповідає за управління даними гри через реалізацію інтерфейсу IDataStorageSystem. Він забезпечує зберігання та завантаження даних гри у вигляді об'єкту StorageData за допомогою JSONDataStorageSystem.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Гра “Play & Puzzle” передбачає інтерактивну взаємодію користувача з графічним інтерфейсом, а також управління логікою гри, збереженням стану гри та зворотним зв'язком для користувача. Вхідні дані включають дії

користувача, наприклад, кліки на клітинки, а вихідні дані - це зміни стану гри, візуальні оновлення та звукові ефекти.

Основними джерелами вхідних даних є дії користувача, які включають:

- Кліки на клітинки: Користувач може клікати на клітинки, щоб пересувати їх. Ці дії виловлюються за допомогою Unity Event System, яка обробляє події введення миші або дотиків на мобільних пристроях. Кожне клікання передається контролеру гри, який визначає, чи можливий рух плитки.
- Команди меню: Користувач може використовувати меню для запуску нової гри, налаштування, або вибору складності. Ці дії також обробляються через Unity UI Event System.

Вихідні дані включають:

- після кожного ходу користувача стан матриці оновлюється і відображається на екрані. Клас PuzzleView відповідає за оновлення графічного інтерфейсу, забезпечуючи коректне відображення поточного стану головоломки;
- для підвищення візуальної привабливості гри, переміщення плиток анімується за допомогою бібліотеки DOTween. Анімації забезпечують плавний перехід плиток на нові місця після кожного ходу;
- для створення більш захоплюючого досвіду гра відтворює звукові ефекти при переміщенні плиток або завершенні гри. Звукові ефекти обробляються класом SoundController, який відтворює відповідні звуки на основі дій користувача;
- після завершення гри користувач отримує сповіщення про перемогу або поразку через повідомлення.

Таким чином, вхідні та вихідні дані організовані таким чином, щоб забезпечити інтерактивність та зручність гри для користувача. Правильна організація вхідних та вихідних даних є критично важливою для створення

захоплюючого ігрового досвіду, а також для забезпечення стабільної роботи програми.

2.6 Опис розробленої системи

2.6.1 Використані технічні засоби

Розробка ігрового застосунку “Puzzle & Play” проводилась із застосуванням персонального комп’ютера с такими характеристиками:

- ОС: Windows 10 Pro x64;
- процесор: AMD Ryzen 3 1200 Quad-Core Processor;
- графічний процесор: ASUS GeForce GTX 1050 Ti;
- оперативна пам’ять: 32 гб;
- накопичувач: SATA SSD 256 GB.

Також додаткові переферійні пристрої:

- клавіатура;
- миша;
- монітор з роздільною здатністю 1366x768.

Проведення розробки на вказаних технічних засобах забезпечило стабільну та ефективну роботу програмного забезпечення, що дозволило досягти високої продуктивності та якості кінцевого продукту. Завдяки потужним компонентам комп'ютера та додатковим периферійним пристроям, розробка ігрового застосунку здійснювалась без затримок і збоїв, що сприяло зручності в процесі програмування та тестування. Таким чином, обрані технічні

засоби забезпечили оптимальні умови для реалізації всіх функціональних вимог проєкту та досягнення запланованих цілей.

2.6.2 Використані програмні засоби

При розробці гри типу “Puzzle & Play” були застосовані програмні засоби, що забезпечили ефективну реалізацію проєкту та високу якість кінцевого продукту.

Основною платформою для розробки стала Unity, один із найпопулярніших ігрових рушіїв, що підтримує кросплатформенну розробку. Unity надає широкий спектр інструментів для створення ігрової логіки, роботи з графікою та анімаціями, а також інтеграції аудіо. Використання Unity дозволило створити гру, яка може бути легко адаптована для різних платформ, таких як Windows, macOS, Android та iOS.

Для програмування логіки гри була використана мова програмування C#. C# забезпечує високу продуктивність та зручність у використанні завдяки своєму простому синтаксису та потужним можливостям. Крім того, C# інтегрується з Unity, що дозволяє використовувати всі переваги рушія для створення ігрових механік.

Для роботи з анімаціями використовувався інструментарій DOTween, потужна бібліотека для створення анімацій в Unity. DOTween дозволяє легко і швидко створювати плавні анімації руху, масштабу та інших параметрів, що значно покращує візуальне сприйняття гри.

Система збереження та завантаження даних була реалізована за допомогою формату JSON, що дозволяє зручно серіалізувати і десеріалізувати

дані. Бібліотека Newtonsoft.Json забезпечує простоту і надійність при збереженні та відновленні стану гри.

Для забезпечення безпеки збережених даних використовувалася бібліотека System.Security.Cryptography. Це дозволило шифрувати дані перед збереженням і розшифровувати їх при завантаженні, що гарантує високий рівень захисту даних за допомогою алгоритму AES.

Для обробки звукових ефектів та фонової музики застосовувався компонент AudioSource, який є частиною Unity. Цей компонент дозволяє відтворювати, зупиняти, ставити на паузу та регулювати гучність звукових ефектів та музики, забезпечуючи динамічний ігровий досвід.

Крім того, для зручності розробки і тестування використовувалися такі інструменти, як Unity Editor, що дозволяє інтерактивно налаштовувати ігрові об'єкти, і Visual Studio, яке забезпечує потужне середовище для написання коду та відлагодження.

Використання цих технічних засобів дозволило створити високоякісну гру з привабливою графікою, плавними анімаціями та надійною системою збереження даних, що забезпечує захоплюючий та безпечний ігровий досвід для користувачів.

2.6.3 Виклик та завантаження програми

Для запуску мобільного застосунку "Puzzle & Play" користувачеві необхідно завантажити файл інсталятора у форматі APK за посиланням або носія даних. Після завантаження APK файлу користувач повинен дозволити

встановлення додатків із невідомих джерел у налаштуваннях свого пристрою, якщо цей параметр ще не увімкнено.

Процес інсталяції через APK файл складається з кількох кроків:

1. Завантаження APK файлу: Користувач завантажує файл APK на свій мобільний пристрій.
2. Налаштування безпеки: У налаштуваннях мобільного пристрою необхідно дозволити встановлення додатків із невідомих джерел. Це можна зробити в розділі "Безпека" або "Приватність".
3. Запуск інсталяції: Користувач відкриває завантажений APK файл за допомогою будь-якого файлового менеджера та натискає на нього для запуску процесу інсталяції. Відкриється вікно з запитом на підтвердження встановлення додатку, де потрібно натиснути "Встановити".
4. Завершення інсталяції: Після завершення процесу інсталяції на головному екрані пристрою з'явиться іконка додатку "Puzzle & Play", за допомогою якої можна буде запустити гру.

Завантаження і встановлення мобільного додатку "Puzzle & Play" через APK файл дозволяє забезпечити зручний та швидкий спосіб розповсюдження програмного забезпечення. Такий метод інсталяції особливо корисний для тестування додатків на різних пристроях під час розробки, а також для розповсюдження бета-версій серед обмеженої аудиторії користувачів. Це підвищує гнучкість процесу тестування та забезпечує можливість оперативного внесення змін і виправлення помилок на основі зворотного зв’язку від користувачів.

2.6.4 Опис інтерфейсу користувача

Інтерфейс користувача для гри "Puzzle & Play" був розроблений з урахуванням принципів простоти, інтуїтивної зрозумілості та зручності використання. Він складається з кількох основних елементів, які забезпечують легкий доступ до основних функцій гри і полегшують взаємодію користувача з додатком.

Головний екран є відправною точкою для користувача і містить основні елементи навігації, такі як кнопки для вибору гри, початку гри та налаштувань.

Рис. 2.10. Зображення головного екрану гри "Puzzle & Play"

На головному екрані розташовані:

- Навігаційні кнопки вибору гри
- Кнопка «Почати гру»
- Кнопка налаштувань

Екран налаштувань дозволяє користувачеві налаштувати параметри гри, такі як звук, музика та вібрації мобільного пристрою.

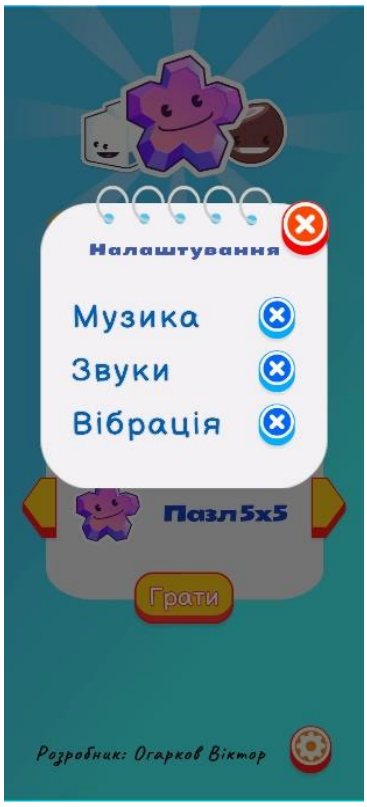

Рис. 2.11. Зображення налаштувань мобільного застосунку

Перед початком гри, гравець має змогу вибрати рівень складності, використовуючі кнопки навігації серед запропонованих варіантів, а саме гра з полем 3x3, 4x4 та 5x5.

Рис. 2.12. Зображення головного меню з режимом гри 4x4

Ігровий екран є основним робочим простором, де користувач взаємодіє з головоломкою. Він включає в себе поле головоломки, завдання, таймер і кнопки керування грою.

Рис. 2.13. Зображення ігрового екрану

На ігровому екрані розташовані:

- Поле головоломки: візуалізує плитки, які гравець повинен перемістити для вирішення головоломки.
- Поле «Місія»: візуалізує кінцевий вигляд головоломки.
- Таймер: відображає залишковий час на виконання головоломки.
- Кнопка "Пауза": дозволяє призупинити гру та відкрити меню паузи.
- Кнопка «Інформація»: відображає завдання для вирішення головоломки.
- Кнопка «Налаштування»: відкриває налаштування гри.

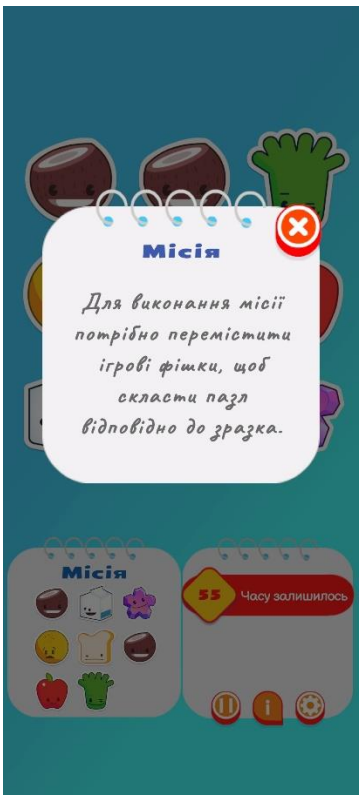

Рис. 2.14. Зображення вікна «Місія»

Після завершення пазлу або закінчення часу, гравець отримує повідомлення, де може вибрати повернутися до головного меню чи спробувати зіграти ще раз.

Рис. 2.15. Зображення повідомлення після завершення гри

Інтерфейс користувача гри "Puzzle & Play" розроблений з метою забезпечення максимальної зручності та інтуїтивної зрозумілості для користувачів. Використання чітких іконок, зручних кнопок та зрозумілих меню дозволяє гравцям легко взаємодіяти з грою та насолоджуватися процесом вирішення головоломок.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи

Вхідні дані:

− передбачуване число операторів – 1091;

− коефіцієнт корекції програми в ході її розробки – 0,1;

− коефіцієнт складності програми – 1,2;

− годинна заробітна плата програміста – 506,83 грн/год;

− коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,2;

− коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

− вартість машино-години ЕОМ – 1.74 грн/год;

Згідно з інформацією від Qubit Labs, початкова заробітна плата молодших розробників програмного забезпечення в Україні складає від 10 до 15 доларів США за годину. Це підтверджується також даними від Mobilunity, які вказують на середню місячну заробітну плату молодших розробників Unity на рівні 1500 доларів США, що при 160 годинах роботи на місяць також відповідає приблизно 10-15 доларів США за годину.

Станом на червень 2024 року курс обміну долара США до гривні становить приблизно 40.53 гривень за 1 долар. Виходячи з цього курсу, середня погодинна ставка для Unity junior програміста в Україні становить 506,83 грн.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, що розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p), \quad (3.2)$$

де q – передбачуване число операторів (1811);

C – коефіцієнт складності програми (1,2);

p – коефіцієнт корекції програми в ході її розробки (0,1).

$$Q = 1091 \cdot 1{,}2 \cdot (1 + 0{,}1) = 1440{,}12.$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k}, \quad (3.3)$$

де В - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,2);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи до 2 років він складає 0,8.

Згідно даних отримуємо такий результат за формулою:

$$t_u = \frac{1440{,}12 \cdot 1{,}2}{85 \cdot 0.8} = 25{,}41 \text{ людино} - \text{годин}.$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20..25) \cdot k}, \quad (3.4)$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу [3.4], людино-годин:

$$t_a = \frac{1440{,}12}{25 \cdot 0{,}8} = 72 \text{ людино-годин}.$$

Витрати на складання програми по готовій схемі:

$$t_n = \frac{Q}{(20..25) \cdot k}; \quad (3.5)$$

$$t_n = \frac{1440{,}12}{25 \cdot 0{,}8} = 72 \text{ людино-годин}.$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)\cdot k};$$ (3.6)

$$t_{\text{отл}} = \frac{1440,12}{5\cdot 0,8} = 360,03 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}^{\text{к}}} = 1.2 \cdot t_{\text{отл}};$$ (3.7)

$$t_{\text{отл}^{\text{к}}} = 1.2 \cdot 360.03 = 432,04 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}};$$ (3.8)

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)\cdot k};$$ (3.9)

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\text{до}} = 0,75 \cdot t_{\text{др}};$$ (3.10)

Підставляючи відповідні значення, отримаємо:

$$t_{\text{др}} = \frac{1440,12}{20\cdot 0,8} = 90 \text{ людино} - \text{годин;}$$

$$t_{\text{до}} = 0,75 \cdot 90 = 67,5 \text{ людино} - \text{годин;}$$

$$t_{\text{д}} = 90 + 67,5 = 157,5 \text{ людино} - \text{годин.}$$

Повертаючись до формули [3.1], отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 24,41 + 72 + 72 + 360,03 + 157,5 = 578,44 \text{ людино – годин.}$$

3.2. Розрахунок витрат на створення програми

Вартість створення програмного забезпечення $К_{по}$ включає вартість заробітної плати розробника програми $З_{ЗП}$ та вартість машинного часу, необхідного для налагодження програми на комп'ютері.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн}, \quad (3.11)$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн}, \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 506,83 грн/год, то отримаємо:

$$З_{ЗП} = 578{,}44 \cdot 506{,}83 = 293170{,}75, \text{ грн},$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot С_{Мч}, \text{ грн}, \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$С_{Мч}$ – вартість машино-години ЕОМ, грн/год.

Підставивши в формулу відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$З_{МВ} = 360,03 \cdot 1,74 = 626,45$ грн.

Звідси, за формулою (3.11), витрати на створення програмного продукту:

$К_{ПО} = 293170,75 + 626,45 = 293797,2$, грн.

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k \cdot F_p}, \text{міс}, \quad (3.14)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 160 годин).

За формулою 3.14 очікуваний період створення програмного забезпечення:

$$T = \frac{578,44}{1 \cdot 160} = 3,6, \text{міс}$$

Висновок: Ігровий додаток “Puzzle & Play” був розроблений для надання гравцям можливості скоротити час та розважитися під час дозвілля. Розрахунки трудомісткості та вартості розробки цього програмного забезпечення показують, що загальна трудомісткість проєкту становить 578,44 людино-годин, що еквівалентно близько 3,6 місяців при стандартному робочому графіку. Загальна вартість створення програмного продукту складає 293797,2 гривень. Ці витрати включають заробітну плату виконавців та вартість машинного часу, необхідного для налагодження програми.

При визначенні вартості враховувалися такі фактори, як коефіцієнт корекції програми, коефіцієнт складності програми, кваліфікація програміста і вартість комп'ютерного машинного часу. Середня погодинна оплата праці

програміста становить 506,83 грн./година, що виходить з урахуванням поточного обмінного курсу долара США по відношенню до гривні.

Отримані результати підтверджують, що розробка даного програмного забезпечення є економічно доцільною і відповідає сучасним вимогам ринку праці та технологій. Високий рівень деталізації розрахунків дозволяє оцінити реальні витрати та ресурси, необхідні для успішного виконання проєкту, що забезпечує його конкурентоспроможність на ринку програмного забезпечення.

ВИСНОВКИ

Метою даної кваліфікаційної роботи була розробка ігрового застосунку "Puzzle & Play", який надає користувачам захоплюючий ігровий процес та сприяє розвитку логічного мислення і уваги. Цей проєкт був виконаний з використанням сучасних технологій і методів розробки мобільних ігор, зокрема, ігрового двигуна Unity та мови програмування C#.

Процес розробки передбачав кілька етапів: аналіз вимог, проєктування, програмування, тестування та впровадження. При аналізі вимог було враховано потреби цільової аудиторії, зокрема, необхідність у доступному та зрозумілому інтерфейсі користувача, а також висока якість графіки та звуку. Під час проєктування була розроблена архітектура додатку, що забезпечує модульність і гнучкість у подальшій розробці та підтримці.

Програмування здійснювалось з використанням мови C#, яка дозволяє ефективно маніпулювати ресурсами та забезпечує високу продуктивність додатку. Використання Unity Engine значно спростило процес розробки, надаючи готові інструменти та бібліотеки для створення ігрових об'єктів, анімацій, фізичних ефектів та інших важливих компонентів гри.

Під час тестування особлива увага приділялась стабільності роботи додатку на різних пристроях, оптимізації продуктивності та виправленню можливих помилок. Це дозволило забезпечити високий рівень якості кінцевого продукту та його відповідність очікуванням користувачів.

Інсталяція додатку "Puzzle & Play" проводиться через APK-файл, що дозволяє зручно встановлювати програму на мобільні пристрої з операційною системою Android. Це забезпечує широке охоплення цільової аудиторії та легкість доступу до гри.

У "Економічному розділі" було проведено детальний розрахунок трудомісткості та вартості розробки програмного забезпечення. Згідно з отриманими даними, загальна трудомісткість розробки склала 578,44 людино-годин. Основними складовими трудомісткості були витрати на підготовку та опис поставленої задачі, дослідження алгоритму рішення, розробку блок-схеми алгоритму, програмування, налагодження програми на ЕОМ та підготовку документації.

Загальна вартість створення програмного забезпечення становить 293797,2 грн, з яких основну частину складає заробітна плата виконавців. Середня заробітна плата програміста за одну годину становила 506,83 грн/год, що відповідає середнім показникам на ринку праці для молодших розробників Unity в Україні.

Період створення програмного застосунку склав 3,6 місяці, що є оптимальним терміном для проєктів такого типу. Це враховує стандартний робочий графік та забезпечує достатній час для виконання всіх етапів розробки.

Розроблений ігровий додаток "Puzzle & Play" успішно виконує поставлені цілі, надаючи користувачам цікаву та корисну розвагу. Він демонструє ефективність використаних технологій та підходів у процесі створення мобільних ігор. Проєкт є конкурентоспроможним на ринку мобільних додатків та має потенціал для подальшого розвитку та вдосконалення.

Загалом, даний проєкт є прикладом успішної реалізації ідеї мобільного ігрового додатку, що поєднує в собі розвагу та розвиток когнітивних здібностей користувачів. Це підтверджує важливість застосування сучасних технологій та методів у процесі розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Биков В.В. Інформаційні системи: підручник. Київ: Видавничий дім "Слово", 2015.
2. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).
3. Браун Джеймс. Розробка додатків для мобільних пристроїв. Київ: Видавництво "Техніка", 2021. 370 с.
4. Войцеховський О.В., Петренко В.В. Основи програмування на мові C#. Київ: Видавництво Київського університету, 2018.
5. Гамма Еріх, Хелм Річард, Джонсон Ральф, Влісідес Джон. Патерни проектування. Київ: Видавництво "КУДІК", 2018. 395 с.
6. Гуменюк М.О. Розробка інтерфейсів користувача. Львів: ЛНУ імені Івана Франка, 2021.
7. Джонсон Е. Ефективний розробник. Київ: Видавництво "КУДІК", 2019. 380 с.
8. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.
9. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. — Київ : Держстандарт України, 1994. – 88 с.
10. Керріган Джордж. Кодування з нуля: Основи програмування для початківців. Київ: Видавництво "Техноцентр", 2019. 300 с.
11. Кормен, Томас Г. Вступ до алгоритмів : Переклад з англійської третього видання : [укр.] = Introduction to Algorithms : Third Edition : [пер. з англ.] /

Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест, Кліфорд Стайн, — К. : К. І. С., 2019. — 1288 с.

12. Ковальчук М.О. Проектування і розробка баз даних. Київ: НТУУ "КПІ", 2018.
13. Ляшенко В.М., Петрова О.В. Програмування на мові C#. Одеса: ОНУ імені Мечникова, 2019.
14. Майерс Грег. Тестування програмного забезпечення. Київ: Видавництво "Інтелект", 2019. 420 с.
15. Макконнелл Стів. Кодерський етос. Київ: Видавництво "КУДІК", 2019. 672 с.
16. Мартін Роберт. Чистий код: Створення, аналіз та рефакторинг. Київ: Видавництво "КУДІК", 2020. 432 с.
17. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с
18. Мельник І.О., Карпенко Ю.О. Основи розробки комп'ютерних ігор. Одеса: ОНПУ, 2019.
19. Нистром Роберт. Патерни проектування ігор. Київ: Видавництво "КУДІК", 2017. 640 с.
20. Сандерс Дональд. Алгоритми та структури даних. Київ: Видавництво "КУДІК", 2021. 600 с.
21. Сєдова І.В., Нестеренко Л.О. Програмування та проектування ігор. Харків: ХДАДМ, 2018.
22. Сіммондс Метью. Розробка програмного забезпечення: Основи та принципи. Київ: Видавництво "Наука", 2018. 350 с.
23. Спірінцев В.В., Удовик І.М., Шевцова О.С. Методичні рекомендації до виконання кваліфікаційних робіт здобувачів першого рівня вищої освіти спеціальності 122 Комп'ютерні науки. Дніпро: НТУ «Дніпровська політехніка», 2021. 65 с.

24. Тищенко С.П., Боровик І.В. Основи об'єктно-орієнтованого програмування. Львів: ЛНУ, 2019.
25. Хант Ендрю, Томас Девід. Програміст-прагматик: Шлях від подмастерья до майстра. Київ: Видавництво "КУДІК", 2020. 352 с.
26. Unity Technologies. Unity Documentation. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 1.06.2024).

ДОДАДОК А

КОД ПРОГРАМИ

Лістинг DependencyInjector.cs:

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

public static class DependencyInjector
{
	private static readonly Dictionary<Type, object> _dependencies = new Dictionary<Type, object>();

	public static void Register<T>(T implementation)
	{
		Type interfaceType = typeof(T);

		_dependencies[interfaceType] = implementation;
	}

	public static T Resolve<T>()
	{
		Type interfaceType = typeof(T);
		if (!_dependencies.ContainsKey(interfaceType))
		{
			Debug.LogError($"Dependency of type {interfaceType} is not registered.");
			return default;
		}

		return (T)_dependencies[interfaceType];
	}
}
```

Лістинг AESEncryptor.cs:

```
using System;
using System.Security.Cryptography;
using System.Text;



public static class AESEncryptor
{
	public struct AESEncryptedText
```

```
    {
        public string IV;
        public string EncryptedText;
    }

    public static AESEncryptedText Encrypt(string plainText, string password)
    {
        using (var aes = Aes.Create())
        {
            aes.GenerateIV();
            aes.Key = ConvertToKeyBytes(aes, password);

            var textBytes = Encoding.UTF8.GetBytes(plainText);

            var aesEncryptor = aes.CreateEncryptor();
            var encryptedBytes = aesEncryptor.TransformFinalBlock(textBytes, 0, textBytes.Length);

            return new AESEncryptedText
            {
                IV = Convert.ToBase64String(aes.IV),
                EncryptedText = Convert.ToBase64String(encryptedBytes)
            };
        }
    }

    public static string Decrypt(AESEncryptedText encryptedText, string password)
    {
        return Decrypt(encryptedText.EncryptedText, encryptedText.IV, password);
    }

    public static string Decrypt(string encryptedText, string iv, string password)
    {
        using (Aes aes = Aes.Create())
        {
            var ivBytes = Convert.FromBase64String(iv);
            var encryptedTextBytes = Convert.FromBase64String(encryptedText);

            var decryptor = aes.CreateDecryptor(ConvertToKeyBytes(aes, password), ivBytes);
            var decryptedBytes = decryptor.TransformFinalBlock(encryptedTextBytes, 0, encryptedTextBytes.Length);
```

```
                return Encoding.UTF8.GetString(decryptedBytes);
            }
        }

        private static byte[] ConvertToKeyBytes(SymmetricAlgorithm algorithm, string password)
        {
            algorithm.GenerateKey();

            var keyBytes = Encoding.UTF8.GetBytes(password);
            var validKeySize = algorithm.Key.Length;

            if (keyBytes.Length != validKeySize)
            {
                var newKeyBytes = new byte[validKeySize];
                Array.Copy(keyBytes, newKeyBytes, Math.Min(keyBytes.Length,
newKeyBytes.Length));
                keyBytes = newKeyBytes;
            }

            return keyBytes;
        }
}
```

Лістинг DataManager.cs:

```
using GrislyTools.Interfaces;
using UnityEngine;

namespace GrislyTools
{
    public static class DataManager
    {
        public static StorageData Data => _gameData;


        private static StorageData _gameData;
        private static string _dataStorageKey = "GameData";
        private static IDataStorageSystem<StorageData> _storageSystem;

        static DataManager()
        {
            _storageSystem = new JSONDataStorageSystem<StorageData>();
```

```
                        if (!_storageSystem.Load(_dataStorageKey, out _gameData))
                        {
                                _gameData = new StorageData();
                        }

                        _gameData.InitializeStorageData(_dataStorageKey, _storageSystem);

                        Debug.Log("DataManager created");
                }
        }
}
```

Лістинг JSONDataStorageSystem.cs:

```
using GrislyTools.Interfaces;
using Newtonsoft.Json;
using UnityEngine;
using static AESEncryptor;

namespace GrislyTools
{

        public class JSONDataStorageSystem<T> : IDataStorageSystem<T> where T : IData
        {
                private readonly string _key;

    public JSONDataStorageSystem()
    {
                        _key = System.Convert.ToBase64String(KeyManager.GetOrCreateKey());
    }

    public bool Load(string key, out T data)
                {
                        string savedData = PlayerPrefs.GetString(key, string.Empty);

                        if (savedData == string.Empty)
                        {
                                data = default;
                                return false;
                        }
                        else
                        {
```

```
                    AESEncryptedText encryptedText =
JsonConvert.DeserializeObject<AESEncryptedText>(savedData);
                    string jsonData = Decrypt(encryptedText, _key);
                    data = JsonConvert.DeserializeObject<T>(jsonData);
                    return true;
                }
            }

            public void Save(string key, T data)
            {
                string jsonData = JsonConvert.SerializeObject(data);
                AESEncryptedText closedData = Encrypt(jsonData, _key);
                string savedData = JsonConvert.SerializeObject(closedData);
                PlayerPrefs.SetString(key, savedData);
            }
        }
}
```

Лістинг KeyManager.cs:

```
using System.Security.Cryptography;
using UnityEngine;

public static class KeyManager
{
    private static readonly string encryptionKey = "encryption_key";

    public static byte[] GetOrCreateKey()
    {
        if (PlayerPrefs.HasKey(encryptionKey))
        {
            string savedKey = PlayerPrefs.GetString(encryptionKey);
            return System.Convert.FromBase64String(savedKey);
        }
        else
        {
            byte[] key = new byte[32];
            using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())
            {
                rng.GetBytes(key);
            }
            string keyString = System.Convert.ToBase64String(key);
            PlayerPrefs.SetString(encryptionKey, keyString);
```

```
                return key;
            }
        }
}
```

Лістинг StorageData.cs:

```
using GrislyTools.Interfaces;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using UnityEngine;

namespace GrislyTools
{
    [Serializable]
    public class StorageData : IData
    {
        [JsonProperty("Data")] private Dictionary<string, object> _data;

        private Dictionary<string, Action<object>> _onValueChanged;
        private static IDataStorageSystem<StorageData> _storageSystem;
        private string _storageKey;

        public void InitializeStorageData(string key, IDataStorageSystem<StorageData>
storageSystem)
        {
            _storageSystem = storageSystem;
            _storageKey = key;

            if (_data == null)
                _data = new Dictionary<string, object>();

            _onValueChanged = new Dictionary<string, Action<object>>();
        }

        public bool GetValue<T>(string key, out T data, T defaultValue = default)
        {
            if (_data.TryGetValue(key, out var storedValue))
            {
                try
                {
                    if (storedValue is T value)
```

```
                {
                    data = value;
                }
                else
                {
                    data = (T)Convert.ChangeType(storedValue, typeof(T));
                }
            }
            catch (InvalidCastException)
            {
                Debug.LogWarning($"Key {key} found, but value is not of type {typeof(T)}. Using default value.");
                data = defaultValue;
            }
            catch (FormatException)
            {
                Debug.LogWarning($"Key {key} found, but value cannot be converted to type {typeof(T)}. Using default value.");
                data = defaultValue;
            }
        }
        else
        {
            Debug.LogWarning($"No data was found with this key: {key}. A new one was created.");
            _data[key] = defaultValue;
            data = defaultValue;
        }

        return true;
    }
    public bool SetValue<T>(string key, T value)
    {
        if (_data.ContainsKey(key))
        {
            if (_data[key] is T)
            {
                _data[key] = value;
            }
            else
            {
                try
```

```
                {
                    _data[key] = (T)Convert.ChangeType(value, typeof(T));
                }
                catch (InvalidCastException)
                {
                    Debug.LogError($"Failed to set value for key {key}. Invalid cast from {value.GetType()} to {typeof(T)}.");
                    return false;
                }
            }

            if (_onValueChanged.ContainsKey(key))
                _onValueChanged[key]?.Invoke(value);
        }
        else
        {
            _data.Add(key, value);
            Debug.LogWarning($"No data was found with this key: {key}. A new one was created");
        }

        _storageSystem.Save(_storageKey, this);
        return true;
    }
    public void SubcribeOnValueChange(string key, Action<object> callback)
    {
        if (_data.ContainsKey(key) && _onValueChanged.ContainsKey(key))
        {
            _onValueChanged[key] += callback;
        }
        else if (_data.ContainsKey(key))
        {
            _onValueChanged.Add(key, callback);
        }
        else
        {
            Debug.Log($"No data was found with this key: {key}");
        }
    }

    public void UnsubcribeAll()
    {
```

```
                _onValueChanged = new Dictionary<string, Action<object>>();
            }

            public void UnsubcribeOnValueChange(string key, Action<object> callback)
            {
                if (_onValueChanged.ContainsKey(key))
                {
                    _onValueChanged[key] -= callback;
                }
            }

            public void Save()
            {
                _storageSystem.Save(_storageKey, this);
            }
        }
}
```

Лістинг IData.cs:

```
using System;

namespace GrislyTools.Interfaces
{
    public interface IData
    {
        bool SetValue<T>(string key, T value);
        bool GetValue<T>(string key,out T data, T defaultValue = default);
        void SubcribeOnValueChange(string key, Action<object> callback);
        void UnsubcribeOnValueChange(string key, Action<object> callback);
        void UnsubcribeAll();
    }
}
```

Лістинг IDataStorageSystem.cs:

```
namespace GrislyTools.Interfaces
{
    public interface IDataStorageSystem<T> where T : IData
    {
        void Save(string key, T data);

        bool Load(string key, out T data);
    }
```

```
}
```

Лістинг IPuzzleController.cs:

```
using UnityEngine.EventSystems;

public interface IPuzzleController
{
	public bool IsEnabled { get; set; }
	void OnPointerClick(PointerEventData eventData);
}
```

Лістинг IPuzzleMatrixFactory.cs:

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

public interface IPuzzleMatrixFactory
{
	int[,] CreateTargetMatrix(int puzzleSize, Sprite[] sprites, out List<int> availableChips);
	int[,] CreateRandomPlayerMatrix(int puzzleSize, List<int> availableChips);
}
```

Лістинг IPuzzleModel.cs:

```
using UnityEngine;
using System;

public interface IPuzzleModel
{
	event Action OnPuzzleSolved;
	int PuzzleSize { get; }
	Sprite[] Sprites { get; }
	Sprite EmptySprite { get; }
	void Move(int row, int col);
}
```

Лістинг IPuzzleView.cs:

```
using UnityEngine;

public interface IPuzzleView
{
	RectTransform PuzzleField { get; }
	bool IsMoving { get; }
	void CreatePlayerPuzzleField(int size);
```

```
        void AnimateMove(int fromRow, int fromCol, int toRow, int toCol, int movedValue, float animDuration);
        public void CreateAndDisplayTargetPuzzleField(int size, int[,] matrix);
        public void Display(int[,] matrix);
}
```

Лістинг IShuffleStrategy.cs:

```
using System;
using System.Collections;

public interface IShuffleStrategy
{
        IEnumerator Shuffle(int movesCount, int[,] playerMatrix, Action<int, int> move, float animDuration, Action enableController);
}
```

Лістинг ArrayExtension.cs:

```
using System;

public static class ArrayExtensions
{
        public static (int,int) IndexOf(this int[,] array, int value)
        {
                for (int i = 0; i < array.GetLength(0); i++)
                {
                        for (int j = 0; j < array.GetLength(1); j++)
                        {
                                if(array[i,j] == value)
                                        return (i,j);
                        }
                }

                throw new Exception("Value not found");
        }
}
```

Лістинг DefaultShuffleStrategy.cs:

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

```
public class DefaultShuffleStrategy : IShuffleStrategy
{
	private const int EMPTY = -1;

	public IEnumerator Shuffle(int movesCount, int[,] playerMatrix, Action<int, int> move, float animDuration, Action enableController)
	{
		float animDurationCoef = 1.3f;
		float startDelay = 0.5f;
		(int, int) lastMovedIndex = (-1, -1);

		yield return new WaitForSeconds(startDelay);

		for (int i = 0; i < movesCount; i++)
		{
			bool isMoved = false;
			while (!isMoved)
			{
				(int, int) emptyIndex = playerMatrix.IndexOf(EMPTY);

				List<(int, int)> possibleMoves = new()
				{
					(emptyIndex.Item1 + 1, emptyIndex.Item2),
					(emptyIndex.Item1 - 1, emptyIndex.Item2),
					(emptyIndex.Item1, emptyIndex.Item2 + 1),
					(emptyIndex.Item1, emptyIndex.Item2 - 1)
				};

				possibleMoves = possibleMoves
					.Where(move => move.Item1 >= 0 && move.Item1 < playerMatrix.GetLength(0) && move.Item2 >= 0 && move.Item2 < playerMatrix.GetLength(1))
					.Where(move => move != lastMovedIndex)
					.ToList();

				if (possibleMoves.Count == 0) continue;

				(int, int) randomMove = possibleMoves[UnityEngine.Random.Range(0, possibleMoves.Count)];

				move(randomMove.Item1, randomMove.Item2);
				lastMovedIndex = emptyIndex;
```

```
                    isMoved = true;
                    yield return new WaitForSeconds(animDuration * animDurationCoef);
                }
            }

            enableController?.Invoke();
        }
}
```

Лістинг GameController.cs:

```
using System.Collections;
using UnityEngine;

public class GameController : MonoBehaviour
{
    [SerializeField] private PuzzleModel _puzzleModel;
    [SerializeField] private PuzzleTimer _timer;
    [SerializeField] private PopupsGroup _popupsGroup;

    private float _animDuration = 3;
    private float _scale = 1f;

    private void Awake()
    {
        _timer.OnTimerEnd += OnTimerEndHandler;
        _puzzleModel.OnPuzzleSolved += OnPuzzleSolvedHandler;
    }

    private void Start()
    {
        StartCoroutine(StartGame());
    }
    private void OnTimerEndHandler()
    {
        // do lose popup
        _popupsGroup.LostPopup.StartPopup(_animDuration, _scale);
    }
    private void OnPuzzleSolvedHandler()
    {
        // do win popup
        _timer.IsTimerActive = false;
        _popupsGroup.WonPopup.StartPopup(_animDuration, _scale);
```

```
        }
        private void OnDestroy()
        {
                _timer.OnTimerEnd -= OnTimerEndHandler;
                _puzzleModel.OnPuzzleSolved -= OnPuzzleSolvedHandler;
        }
        private IEnumerator StartGame()
        {
                _timer.IsTimerActive = false;
                var loadingDelay = new WaitForSeconds(1f);

                yield return new WaitForEndOfFrame();
                yield return loadingDelay;

                _timer.IsTimerActive = true;
        }
}
```

Лістинг PuzzleController.cs:

```
using UnityEngine;
using UnityEngine.EventSystems;

public class PuzzleController : MonoBehaviour, IPuzzleController, IPointerClickHandler
{
        public bool IsEnabled { get; set; }

        private IPuzzleView _view;
        private IPuzzleModel _model;

        public void Init()
        {
                _view = DependencyInjector.Resolve<IPuzzleView>();
                _model = DependencyInjector.Resolve<IPuzzleModel>();
        }

        public void OnPointerClick(PointerEventData eventData)
        {
                if (_view.IsMoving || !IsEnabled)
                        return;

                Vector2 localPoint;
```

```
                if (RectTransformUtility.ScreenPointToLocalPointInRectangle(_view.PuzzleField,
eventData.position, eventData.pressEventCamera, out localPoint))
                {
                        int size = _model.PuzzleSize;
                        float cellSize = _view.PuzzleField.rect.width / size;

                        int col = Mathf.Clamp(Mathf.FloorToInt(localPoint.x / cellSize), 0, size - 1);
                        int row = Mathf.Clamp(Mathf.FloorToInt(-localPoint.y / cellSize), 0, size - 1);

                        _model.Move(row, col);
                }
        }
}
```

Лістинг PuzzleMatrixFactory.cs:

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

public class PuzzleMatrixFactory : IPuzzleMatrixFactory
{
        private const int EMPTY = -1;

        public int[,] CreateTargetMatrix(int puzzleSize, Sprite[] sprites, out List<int> availableChips)
        {
                int totalCells = puzzleSize * puzzleSize;
                List<int> spriteIndices = new();
                availableChips = new List<int>();

                for (int i = 0; i < sprites.Length; i++)
                {
                        spriteIndices.Add(i + 1);
                }

                while (spriteIndices.Count < totalCells - 1)
                {
                        int randomValue = UnityEngine.Random.Range(1, sprites.Length + 1);
                        spriteIndices.Add(randomValue);
                }

                System.Random rand = new System.Random();
                spriteIndices = ShuffleList(spriteIndices, rand);
```

```
            int[,] matrixArray = new int[puzzleSize, puzzleSize];
            int index = 0;
            for (int i = 0; i < puzzleSize; i++)
            {
                for (int j = 0; j < puzzleSize; j++)
                {
                    if (index < spriteIndices.Count)
                    {
                        matrixArray[i, j] = spriteIndices[index];
                        index++;
                    }
                }
            }

            matrixArray[puzzleSize - 1, puzzleSize - 1] = EMPTY;


            availableChips.AddRange(spriteIndices);
            return matrixArray;
        }

        public int[,] CreateRandomPlayerMatrix(int puzzleSize, List<int> availableChips)
        {
            availableChips.Add(EMPTY);

            int[,] matrixArray = new int[puzzleSize, puzzleSize];
            System.Random rand = new System.Random();
            for (int i = 0; i < puzzleSize; i++)
            {
                for (int j = 0; j < puzzleSize; j++)
                {
                    if (availableChips.Count > 0)
                    {
                        int randomIndex = rand.Next(availableChips.Count);
                        matrixArray[i, j] = availableChips[randomIndex];
                        availableChips.RemoveAt(randomIndex);
                    }
                    else
                    {
                        throw new InvalidOperationException("No more available chips to
assign.");
```

```
                }
            }
        }
        return matrixArray;
    }

    private List<int> ShuffleList(List<int> list, System.Random rand)
    {
        for (int i = list.Count - 1; i > 0; i--)
        {
            int randomIndex = rand.Next(i + 1);
            int temp = list[i];
            list[i] = list[randomIndex];
            list[randomIndex] = temp;
        }
        return list;
    }
}
```

Лістинг PuzzleMatrixIterator.cs:

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

public class PuzzleMatrixIterator : IEnumerator<int>
{
    private readonly int[,] _matrix;
    private readonly int _size;
    private int _row;
    private int _col;

    public PuzzleMatrixIterator(int[,] matrix)
    {
        _matrix = matrix;
        _size = matrix.GetLength(0);
        _row = 0;
        _col = -1;
    }
    public int Current => _matrix[_row, _col];
    object IEnumerator.Current => Current;
    public bool MoveNext()
    {
        _col++;
```

```
            if (_col >= _size)
            {
                _col = 0;
                _row++;
            }
            return _row < _size;
        }
        public void Reset()
        {
            _row = 0;
            _col = -1;
        }
        public void Dispose()
        {
        }
}
```

Лістинг PuzzleModel.cs:

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

public class PuzzleModel : MonoBehaviour, IPuzzleModel
{
        public event Action OnPuzzleSolved;

        public int PuzzleSize => _puzzleSize;
        public Sprite[] Sprites => _sprites;
        public Sprite EmptySprite => _emptySprite;

        [SerializeField] private IPuzzleView _view;
        [SerializeField] private IPuzzleController _controller;
        [SerializeField, Min(3)] private int _puzzleSize;
        [SerializeField] private Sprite[] _sprites;
        [SerializeField] private Sprite _emptySprite;
        [SerializeField] private AudioClip _socketMoveAudio;
        [SerializeField] private float _socketMoveVolume;
        [SerializeField] private AudioClip _uiClickAudio;
        [SerializeField] private float _uiClickVolume;
        [SerializeField] private AudioClip _shuffleAudio;
        [SerializeField] private float _shuffleVolume;
```

```
        private int[,] _targetMatrix;
        private int[,] _playerMatrix;
        private float _animDuration = 0.1f;
        private List<int> _availableChips = new();
        private bool _hasShuffled = false;

        private IPuzzleMatrixFactory _matrixFactory;
        private IShuffleStrategy _shuffleStrategy;

        public void Init()
        {
                _view = DependencyInjector.Resolve<IPuzzleView>();
                _controller = DependencyInjector.Resolve<IPuzzleController>();
                _matrixFactory = DependencyInjector.Resolve<IPuzzleMatrixFactory>();
                _shuffleStrategy = DependencyInjector.Resolve<IShuffleStrategy>();

                _controller.IsEnabled = false;
                _targetMatrix = _matrixFactory.CreateTargetMatrix(_puzzleSize, _sprites, out _availableChips);
                _playerMatrix = _matrixFactory.CreateRandomPlayerMatrix(_puzzleSize, _availableChips);

                _view.CreatePlayerPuzzleField(_puzzleSize);
                _view.CreateAndDisplayTargetPuzzleField(_puzzleSize, _targetMatrix);
                _view.Display(_playerMatrix);

                StartCoroutine(_shuffleStrategy.Shuffle(15, _playerMatrix, Move, _animDuration, () =>
                {
                        _animDuration = 0.2f;
                        _controller.IsEnabled = true;
                        _hasShuffled = true;
                }));
        }

        public void Move(int row, int col)
        {
                if (PuzzleUtils.IsAdjacentToEmptyCell(_playerMatrix, row, col, out int emptyRow, out int emptyCol))
                {
                        SwapTiles(row, col, emptyRow, emptyCol);

                        _view.AnimateMove(row, col, emptyRow, emptyCol, _playerMatrix[emptyRow, emptyCol], _animDuration);
```

```
                    PlayMoveSound();

                    if (IsPuzzleSolved())
                    {
                        OnPuzzleSolved?.Invoke();
                        Debug.Log("Puzzle solved!");
                    }
                }
                else
                {
                    PlayClickSound();
                }
            }
            private void SwapTiles(int row, int col, int emptyRow, int emptyCol)
            {
                int temp = _playerMatrix[row, col];
                _playerMatrix[row, col] = _playerMatrix[emptyRow, emptyCol];
                _playerMatrix[emptyRow, emptyCol] = temp;
            }
            private void PlayMoveSound()
            {
                if (!_hasShuffled)
                    GlobalSettings.SoundController.PlaySFX(_shuffleAudio, _shuffleVolume);
                else
                    GlobalSettings.SoundController.PlaySFX(_socketMoveAudio,
_socketMoveVolume, true);

            }
            private void PlayClickSound()
            {
                GlobalSettings.SoundController.PlaySFX(_uiClickAudio, _uiClickVolume, true);
            }
            private bool IsPuzzleSolved()
            {
                if (_playerMatrix.GetLength(0) != _targetMatrix.GetLength(0) || _playerMatrix.GetLength(1)
!= _targetMatrix.GetLength(1))
                {
                    return false;
                }

                for (int i = 0; i < _playerMatrix.GetLength(0); i++)
```

```
                {
                        for (int j = 0; j < _playerMatrix.GetLength(1); j++)
                        {
                                if (_playerMatrix[i, j] != _targetMatrix[i, j])
                                {
                                        return false;
                                }
                        }
                }

                return true;
        }
}
```

Лістинг PuzzleTimer.cs:

```
using TMPro;
using UnityEngine;

public class PuzzleTimer : MonoBehaviour
{
        public bool IsTimerActive = true;

        public event System.Action OnTimerEnd;
        public event System.Action OnNearToEnd;

        [SerializeField] private TextMeshProUGUI _totalTimeText;
        [SerializeField] private TextMeshProUGUI _remainingTimeText;
        [SerializeField] private int _totalTime;

        private float _timer;
        private float _nearToEndTime = 15;

        private void Awake()
        {
                _totalTimeText.text = $"{_totalTime.ToString()}c";
                _remainingTimeText.text = _totalTime.ToString();
        }
        private void Update()
        {
                DecreaseTime();
        }
        private void DecreaseTime()
```

```
        {
            if (IsTimerActive)
            {
                _timer += Time.deltaTime;

                if (_timer >= 1)
                {
                    _totalTime -= (int)_timer;
                    _remainingTimeText.text = _totalTime.ToString();
                    _timer = 0;
                }

                if (_totalTime == _nearToEndTime && _timer == 0)
                {
                    OnNearToEnd?.Invoke();
                }

                if (_totalTime <= 0)
                {
                    OnTimerEnd?.Invoke();
                    IsTimerActive = false;
                }
            }
        }
}
```

Лістинг PuzzleUtils.cs:

```
public static class PuzzleUtils
{
    public const int EMPTY = -1;

    public static bool IsAdjacentToEmptyCell(int[,] matrix, int row, int col, out int emptyRow, out int emptyCol)
    {
        emptyRow = -1;
        emptyCol = -1;
        int size = matrix.GetLength(0);

        if (row > 0 && matrix[row - 1, col] == EMPTY)
        {
            emptyRow = row - 1;
            emptyCol = col;
```

```
                return true;
            }

            if (row < size - 1 && matrix[row + 1, col] == EMPTY)
            {
                emptyRow = row + 1;
                emptyCol = col;
                return true;
            }

            if (col > 0 && matrix[row, col - 1] == EMPTY)
            {
                emptyRow = row;
                emptyCol = col - 1;
                return true;
            }

            if (col < size - 1 && matrix[row, col + 1] == EMPTY)
            {
                emptyRow = row;
                emptyCol = col + 1;
                return true;
            }

            return false;
        }
}
```

Лістинг PuzzleView.cs:

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using DG.Tweening;

public class PuzzleView : MonoBehaviour, IPuzzleView
{
    public RectTransform PuzzleField => _puzzleField;
    public bool IsMoving => _isMoving;

    [SerializeField] private GridLayoutGroup _playerMatrix;
    [SerializeField] private GridLayoutGroup _targetMatrix;
    [SerializeField] private RectTransform _puzzleField;
```

```
[SerializeField] private RectTransform _targetField;
[SerializeField] private int _playerOffset;
[SerializeField] private int _playerSpacing;
[SerializeField] private int _matchSpacing;
[SerializeField] private GameObject _cellPrefab;

private IPuzzleModel _model;
private List<GameObject> _cells;
private List<GameObject> _targetCells;
private Tween _tween;
private bool _isMoving = false;

public void Init()
{
	_model = DependencyInjector.Resolve<IPuzzleModel>();
}

public void CreatePlayerPuzzleField(int size)
{
	ConfigureGrid(_playerMatrix, _puzzleField, size, _playerOffset, _playerSpacing);

	_cells = new List<GameObject>();
	for (int i = 0; i < size * size; i++)
	{
		GameObject cell = Instantiate(_cellPrefab, _puzzleField);
		_cells.Add(cell);
	}
}

public void CreateAndDisplayTargetPuzzleField(int size, int[,] matrix)
{
	ConfigureGrid(_targetMatrix, _targetField, size, _matchSpacing, _matchSpacing);

	_targetCells = new List<GameObject>();
	for (int i = 0; i < size * size; i++)
	{
		GameObject cell = Instantiate(_cellPrefab, _targetField);
		_targetCells.Add(cell);
	}

	UpdateCells(matrix, _targetCells);
}
```

```
	public void Display(int[,] matrix)
	{
		UpdateCells(matrix, _cells);
	}

	private void ConfigureGrid(GridLayoutGroup grid, RectTransform field, int size, int offset, int spacing)
	{
		float maxFieldSize = Mathf.Max(field.rect.width, field.rect.height);
		float cellSize = Mathf.Floor((maxFieldSize - (offset * (size - 1))) / size);

		grid.cellSize = new Vector2(cellSize, cellSize);
		grid.spacing = new Vector2(spacing, spacing);
		grid.constraint = GridLayoutGroup.Constraint.FixedColumnCount;
		grid.constraintCount = size;
	}

	private void UpdateCells(int[,] matrix, List<GameObject> cells)
	{
		int size = matrix.GetLength(0);
		for (int i = 0; i < size; i++)
		{
			for (int j = 0; j < size; j++)
			{
				int index = i * size + j;
				Image cellImage = cells[index].GetComponent<Image>();
				int value = matrix[i, j];
				cellImage.sprite = value == -1 ? _model.EmptySprite : _model.Sprites[value
- 1];
			}
		}
	}

	public void AnimateMove(int fromRow, int fromCol, int toRow, int toCol, int movedValue, float
animDuration)
	{
		int size = _model.PuzzleSize;
		int fromIndex = fromRow * size + fromCol;
		int toIndex = toRow * size + toCol;

		GameObject fromCell = _cells[fromIndex];
		GameObject toCell = _cells[toIndex];
```

```
                Vector3 fromPosition = fromCell.transform.position;
                Vector3 toPosition = toCell.transform.position;

                _tween?.Kill();
                _tween = fromCell.transform.DOMove(toPosition, animDuration)
                        .OnStart(() => _isMoving = true)
                        .OnKill(() =>
                        {
                                Image fromImage = fromCell.GetComponent<Image>();
                                Image toImage = toCell.GetComponent<Image>();

                                fromImage.sprite = _model.EmptySprite;
                                toImage.sprite = _model.Sprites[movedValue - 1];
                                fromCell.transform.position = fromPosition;
                                _isMoving = false;
                        });
        }

        private void OnDestroy()
        {
                _tween?.Kill();
        }
}
```

Лістинг BackToLobbyButton.cs:

```
using DG.Tweening;
using GrislyTools;
using UnityEngine;
using UnityEngine.UI;

public class BackToLobbyButton : MonoBehaviour
{
        [SerializeField] private Button _button;

        private const string LOBBY_SCENE = "Lobby";

        private void Start()
        {
                _button.onClick.AddListener(BackToLobby);
        }
        private void BackToLobby()
```

```
        {
                DOTween.KillAll();
                DataManager.Data.GetValue("CurrentScene", out string scene);
                GlobalSettings.SceneLoader.UnloadScene(scene);
                GlobalSettings.SceneLoader.LoadScene(LOBBY_SCENE);
                GlobalSettings.SceneLoader.AllowSceneActivation();
        }
        private void OnDestroy()
        {
                _button.onClick?.RemoveListener(BackToLobby);
        }
}
```

Лістинг OpenPopupWindow.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class OpenPopupWindow : MonoBehaviour
{
        [SerializeField] private Transform _transform;
        [SerializeField] private Button _openButton;
        [SerializeField] private Button _closeButton;
        [SerializeField] private PuzzleTimer _timer;

        private void Awake()
        {
                _openButton.onClick.AddListener(OpenButtonHandler);
                _closeButton.onClick.AddListener(CloseButtonHandler);
                gameObject.SetActive(false);
        }

        private void OpenButtonHandler()
        {
                if (_timer != null)
                        _timer.IsTimerActive = false;

                Time.timeScale = 0;
                _transform.gameObject.SetActive(true);
        }
        private void CloseButtonHandler()
        {
                if(_timer != null)
```

```
                _timer.IsTimerActive = true;

            Time.timeScale = 1;
            _transform.gameObject.SetActive(false);
        }

        private void OnDestroy()
        {
            Time.timeScale = 1;
            _openButton.onClick.RemoveAllListeners();
            _closeButton.onClick.RemoveAllListeners();
        }
}
```

Лістинг PopupsGroup.cs:

```
using UnityEngine;

public class PopupsGroup : MonoBehaviour
{
    public PopupController WonPopup;
    public PopupController LostPopup;
}
```

Лістинг ReloadCurrentSceneButton.cs:

```
using UnityEngine;
using UnityEngine.UI;

public class ReloadCurrentSceneButton : MonoBehaviour
{
    [SerializeField] private Button _button;

    private void Start()
    {
        _button.onClick.AddListener(ReloadScene);
    }
    private void ReloadScene()
    {
        GlobalSettings.SceneLoader.ReloadScene();
    }
    private void OnDestroy()
    {
        _button.onClick.RemoveAllListeners();
```

```
	}
}
```

Лістинг GlobalSettings.cs:

```
using DG.Tweening;
using GrislyTools;
using System.Collections;
using UnityEngine;

public class GlobalSettings : MonoBehaviour
{
	#region Singleton
	public static GlobalSettings Instance
	{
		get
		{
			return _instance;
		}
	}
	private static GlobalSettings _instance;
	#endregion

	public static SoundController SoundController => Instance._soundController;
	public static SceneLoader SceneLoader => Instance._sceneLoader;

	[SerializeField] private SoundController _soundController;
	[SerializeField] private SceneLoader _sceneLoader;
	[SerializeField] private AudioClip _musicClip;
	[SerializeField] private float _volume;

	private void Awake()
	{
		if (_instance != null && _instance != this)
		{
			Destroy(gameObject);
			return;
		}

		_instance = this;
		DontDestroyOnLoad(gameObject);
	}
```

```
        private void Start()
        {
            Initialize();
        }

        private void Initialize()
        {
            Application.targetFrameRate = 60;
            DOTween.defaultAutoKill = true;

            StartCoroutine(InitAsync());
        }
        private IEnumerator InitAsync()
        {
            if (!PlayerPrefs.HasKey("Loaded"))
            {
                // settings
                yield return new WaitUntil(() => DataManager.Data.GetValue("Music", out bool m));
                yield return new WaitUntil(() => DataManager.Data.GetValue("Sound", out bool s));
                yield return new WaitUntil(() => DataManager.Data.GetValue("Taptic", out bool t));

                DataManager.Data.Save();
                PlayerPrefs.SetString("Loaded", "true");
            }

            _soundController.Initialize(10);
            _soundController.PlayMusic(_musicClip, _volume);

            _sceneLoader.LoadScene("Lobby");
            _sceneLoader.AllowSceneActivation();
        }
}
```

Лістинг SceneLoader.cs:

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using DG.Tweening;
using System.Collections;
using GrislyTools;
```

```
public class SceneLoader : MonoBehaviour
{
	[SerializeField] private Canvas _mainCanvas;
	[SerializeField] private Camera _camera;
	[SerializeField] private Slider _loadSlider;
	[SerializeField] private LoadingBar _loadingBar;
	[SerializeField] private float _minLoadDuration;
	[SerializeField] private float _maxLoadDuration;

	private Tween _tween;
	private Coroutine _coroutine;
	private AsyncOperation _loadingOperation;
	private bool _loading;

	public void LoadScene(string sceneName)
	{
		if (string.IsNullOrEmpty(sceneName))
		{
			Debug.LogError("Scene name is null or empty");
			return;
		}

		if (_loading)
			return;

		_loading = true;

		DataManager.Data.SetValue("CurrentScene", sceneName);

		_tween?.Kill();
		if (_coroutine != null)
		{
			StopCoroutine(_coroutine);
			_coroutine = null;
		}

		_loadSlider.value = 0;

		// Mute music when loading
		GlobalSettings.SoundController.PauseMusic();
```

```
            _loadingOperation = SceneManager.LoadSceneAsync(sceneName,
LoadSceneMode.Additive);
            _loadingOperation.allowSceneActivation = false;

            _mainCanvas.gameObject.SetActive(true);
            _camera.gameObject.SetActive(true);

            float loadDuration = Random.Range(_minLoadDuration, _maxLoadDuration);

            _loadingBar.StartLoadingAnim();

            _coroutine = StartCoroutine(LoadSceneRoutine());
            _tween = _loadSlider.DOValue(_loadSlider.maxValue, loadDuration).OnComplete(() =>
            {
                  _loadingBar.KillAnim();
            });
      }

      private IEnumerator LoadSceneRoutine()
      {
            while (!_loadingOperation.isDone)
            {
                  yield return null;
            }

            _camera.gameObject.SetActive(false);
            _mainCanvas.gameObject.SetActive(false);

            // Unmute music if possible
            GlobalSettings.SoundController.UnpauseMusic();

            _loading = false;
      }

      public void UnloadScene(string sceneName)
      {
            if (SceneManager.GetSceneByName(sceneName).isLoaded)
            {
                  _camera.gameObject.SetActive(true);
                  SceneManager.UnloadSceneAsync(sceneName);
            }
      }
```

```
	public void ReloadScene()
	{
		DataManager.Data.GetValue("CurrentScene", out string scene);
		StartCoroutine(UnloadAndReloadScene(scene));
	}

	public void AllowSceneActivation()
	{
		StartCoroutine(WaitForSliderAndActivateScene());
	}

	private IEnumerator WaitForSliderAndActivateScene()
	{
		// Wait until the slider reaches its maximum value
		while (_loadSlider.value != _loadSlider.maxValue)
		{
			yield return null;
		}

		// Allow scene activation
		_loadingOperation.allowSceneActivation = true;
	}

	private IEnumerator UnloadAndReloadScene(string sceneName)
	{
		if (SceneManager.GetSceneByName(sceneName).isLoaded)
		{
			AsyncOperation unloadOperation = SceneManager.UnloadSceneAsync(sceneName);

			while (!unloadOperation.isDone)
			{
				yield return null;
			}

			AsyncOperation loadOperation = SceneManager.LoadSceneAsync(sceneName, LoadSceneMode.Additive);
			_loadingOperation = loadOperation;
			_loadingOperation.allowSceneActivation = false;

			StartCoroutine(LoadSceneRoutine());
```

```
                AllowSceneActivation();
            }
        }

        private void OnDestroy()
        {
            _tween?.Kill();
        }
}
```

Лістинг ListExtensions.cs:

```
using System.Collections.Generic;
using System.Linq;

using Random = UnityEngine.Random;

public static partial class Extensions
{
    public static T GetRandom<T>(this IEnumerable<T> collection)
    {
        var count = collection.Count();
        if (count == 0)
            return default(T);

        return collection.ElementAt(Random.Range(0, count));
    }

    public static void Populate<T>(this T[] arr, T value)
    {
        for (var i = 0; i < arr.Length; i++)
            arr[i] = value;
    }

    public static string Print<T>(this IList<T> list)
    {
        var res = "[ " + string.Join(", ", list.Select(s => $"{s}")) + " ]";
        return res;
    }

    public static IList<T> Shuffle<T>(this IList<T> list)
    {
        System.Random rng = new System.Random();
```

```
            var n = list.Count;
            while (n > 1)
            {
                n--;
                var k = rng.Next(n + 1);
                T value = list[k];
                list[k] = list[n];
                list[n] = value;
            }

            return list;
        }

        public static int GetRandomIndex<T>(this List<T> list)
        {
            if (list.Count == 0)
                return -1;

            return Random.Range(0, list.Count);
        }
}
```

Лістинг LoadingBar.cs:

```
using DG.Tweening;
using TMPro;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.UI;


public class LoadingBar : MonoBehaviour
{
    [SerializeField] private Image _fillImage;
    [SerializeField] private TextMeshProUGUI _text;
    [SerializeField] private Slider _slider;

    private Tween _tween;
    private Vector3 _endPos = new Vector3(0, 0, -360F);
    private float _animDuration = 1f;

    private void Start()
```

```
	{
		_slider.onValueChanged.AddListener(OnSliderValueChanged);
	}
	private void OnSliderValueChanged(float value)
	{
		_text.text = Mathf.RoundToInt(value).ToString() + "%";

		if (value >= 99)
			_tween?.Kill();
	}
	private void OnDestroy()
	{
		KillAnim();
		_slider.onValueChanged?.RemoveListener(OnSliderValueChanged);
	}
	public void StartLoadingAnim()
	{
		_tween?.Kill();
		_tween = _fillImage.transform.DORotate(_endPos, _animDuration,
RotateMode.FastBeyond360)
			.SetLoops(-1, LoopType.Restart)
			.SetEase(Ease.Flash);
	}
	public void KillAnim()
	{
		_tween?.Kill();
	}
}
```

Лістинг ChangeImageOnClick.cs:

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class ChangeImageOnClick : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
	[SerializeField] private Image _targetImage;
	[SerializeField] private Sprite _onDownImage;
	[SerializeField] private Sprite _onUpImage;

	public void OnPointerDown(PointerEventData eventData)
	{
```

```
                _targetImage.sprite = _onDownImage;
        }

        public void OnPointerUp(PointerEventData eventData)
        {
                _targetImage.sprite = _onUpImage;
        }
}
```

Лістинг GameChoice.cs:

```
using DG.Tweening;
using GrislyTools;
using UnityEngine;
using UnityEngine.UI;

public class GameChoice : MonoBehaviour
{
        [SerializeField] private Button _prevButton;
        [SerializeField] private Button _nextButton;
        [SerializeField] private Button _playButton;
        [SerializeField] private Transform _gamesTransform;
        [SerializeField] float _transformOffset;

        private const float FIRST_GAME = 0;
        private const float LAST_GAME = 2;
        private string[] _games = {"15Puzzle3", "15Puzzle4", "15Puzzle5" };

        private Tween _tween;
        private int _currentGame = 0;
        private float _duration = 1f;

        private void Start()
        {
                _prevButton.onClick.AddListener(OnPrevClicked);
                _nextButton.onClick.AddListener(OnNextClicked);
                _playButton.onClick.AddListener(OnPlayClicked);
        }

        private void OnPrevClicked()
        {
                if (_currentGame == FIRST_GAME)
                        return;
```

```
                _currentGame--;

                _tween.Kill();
                float pos = _gamesTransform.localPosition.x + _transformOffset;
                _tween = _gamesTransform.DOLocalMoveX(pos, _duration)
                        .OnStart(() => SetButtonsInteractable(false))
                        .OnComplete(() => SetButtonsInteractable(true));
        }
        private void OnNextClicked()
        {
                if (_currentGame == LAST_GAME)
                        return;

                _currentGame++;

                _tween?.Kill();
                float pos = _gamesTransform.localPosition.x - _transformOffset;
                _tween = _gamesTransform.DOLocalMoveX(pos, _duration)
                        .OnStart(() => SetButtonsInteractable(false))
                        .OnComplete(() => SetButtonsInteractable(true));
                }
        private void SetButtonsInteractable(bool enable)
        {
                _nextButton.interactable = enable;
                _prevButton.interactable = enable;
                _playButton.interactable = enable;
        }
        private void OnPlayClicked()
        {
                DataManager.Data.GetValue("CurrentScene", out string scene);
                GlobalSettings.SceneLoader.UnloadScene(scene);
                GlobalSettings.SceneLoader.LoadScene(_games[_currentGame]);
                GlobalSettings.SceneLoader.AllowSceneActivation();
        }
        private void OnDestroy()
        {
                _prevButton.onClick.RemoveAllListeners();
                _nextButton.onClick.RemoveAllListeners();
                _playButton.onClick.RemoveAllListeners();
        }
}
```

Лістинг ScaleOnEnter.cs:

```
using DG.Tweening;
using UnityEngine;
using UnityEngine.EventSystems;

public class ScaleOnEnter : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
	[SerializeField] private Transform _targetTransform;
	[SerializeField] private float _startScale = 1f;
	[SerializeField] private float _endScale = 1.1f;
	[SerializeField] private float _duration;

	private Tween _tween;

	public void OnPointerEnter(PointerEventData eventData)
	{
		_tween?.Kill();
		_tween = _targetTransform.DOScale(_endScale, _duration)
			.SetEase(Ease.InOutQuad);
	}

	public void OnPointerExit(PointerEventData eventData)
	{
		_tween?.Kill();
		_tween = _targetTransform.DOScale(_startScale, _duration)
			.SetEase(Ease.InOutQuad);
	}
}
```

Лістинг TittleEffect.cs:

```
using DG.Tweening;
using UnityEngine;

public class TittleEffect : MonoBehaviour
{
	[SerializeField] private Transform _effectTransform;
	[SerializeField] private Transform _tittleTransform;
	[SerializeField] private Transform _firstCharacter;
	[SerializeField] private Transform _secondCharacter;
	[SerializeField] private Transform _thirdCharacter;
```

```
        private Tween _tween;
        private Tween _tween2;
        private Sequence _sequence;
        private Vector3 _fullRotate = new Vector3(0, 0, 360f);
        private float _rotateSpeed = 10f;
        private float _movePosY;
        private float _offset = 25f;
        private float _moveDuration = 1f;
        private float _endScale = 1.3f;
        private float _scaleDuration = 0.8f;

        private void Start()
        {
            _movePosY = _tittleTransform.localPosition.y + _offset;
            _sequence = DOTween.Sequence();

            KillTween();

            _tween = _effectTransform.DOLocalRotate(_fullRotate, _rotateSpeed,
RotateMode.FastBeyond360)
                .SetLoops(-1, LoopType.Restart)
                .SetEase(Ease.Linear);

            _tween2 = _tittleTransform.DOLocalMoveY(_movePosY, _moveDuration)
                .SetLoops(-1, LoopType.Yoyo);

            float normalScale = 1f;

            _sequence.Append(_firstCharacter.DOScale(_endScale, _scaleDuration));
            _sequence.Append(_firstCharacter.DOScale(normalScale, _scaleDuration));
            _sequence.Append(_secondCharacter.DOScale(_endScale, _scaleDuration));
            _sequence.Append(_secondCharacter.DOScale(normalScale, _scaleDuration));
            _sequence.Append(_thirdCharacter.DOScale(_endScale, _scaleDuration));
            _sequence.Append(_thirdCharacter.DOScale(normalScale, _scaleDuration));
            _sequence.SetLoops(-1, LoopType.Restart);
        }
        private void KillTween()
        {
            _tween?.Kill();
            _tween2?.Kill();

        }
```

```
        private void OnDestroy()
        {
                KillTween();
                _sequence?.Kill();
                _sequence = null;
        }
}
```

Лістинг PopupController.cs:

```
using DG.Tweening;
using UnityEngine;

public class PopupController : MonoBehaviour
{
   [SerializeField] private Transform _popupTransform;
   [SerializeField] private Transform _popupImage;
   [SerializeField] private AudioClip _popupClip;
   [SerializeField] private float _popupVolume;
   [SerializeField] private float _startScale = 0.2f;

   private Tween _tween;

   private void Awake()
   {
      ClosePopup();
   }

   public void StartPopup(float animDuration, float endScale)
   {
      _popupTransform.gameObject.SetActive(true);
      GlobalSettings.SoundController.PauseMusic();
      GlobalSettings.SoundController.PlaySFX(_popupClip, _popupVolume, true);

      _tween?.Kill();

      _tween = _popupImage.DOScale(endScale, animDuration);
   }
   public void ClosePopup()
   {
      GlobalSettings.SoundController.UnpauseMusic();
      _tween?.Kill();
      _tween = _popupImage.DOScale(_startScale, 0);
```

```
        _popupTransform.gameObject.SetActive(false);
    }

    private void OnDestroy()
    {
        _tween?.Kill();
        _tween = null;
    }
}
```

Лістинг PuzzleSceneController.cs:

```
using UnityEngine;

public class PuzzleSceneController : MonoBehaviour
{
        [SerializeField] private PuzzleController _puzzleController;
        [SerializeField] private PuzzleModel _puzzleModel;
        [SerializeField] private PuzzleView _puzzleView;

        private void Awake()
        {
                CreateInjectorAndDepencies();
                InitializeMVC();
        }
        private void CreateInjectorAndDepencies()
        {
                DependencyInjector.Register<IPuzzleController>(_puzzleController);
                DependencyInjector.Register<IPuzzleModel>(_puzzleModel);
                DependencyInjector.Register<IPuzzleView>(_puzzleView);
                DependencyInjector.Register<IShuffleStrategy>(new DefaultShuffleStrategy());
                DependencyInjector.Register<IPuzzleMatrixFactory>(new PuzzleMatrixFactory());
        }
        private void InitializeMVC()
        {
                _puzzleController.Init();
                _puzzleView.Init();
                _puzzleModel.Init();
        }
}
```

Лістинг SettingButtonAnimToggle.cs:

```
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;


public class SettingButtonAnimToggle : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
	[SerializeField] private Image _toggleImage;
	[SerializeField] private Sprite _onToggleSprite;
	[SerializeField] private Sprite _pressedSprite;
	[SerializeField] private Sprite _offToggleSprite;

	private bool _isActive = false;

	public void Initialize(bool enable)
	{
		_isActive = enable;
		_toggleImage.sprite = enable ? _onToggleSprite : _offToggleSprite;
	}

	public void OnPointerDown(PointerEventData eventData)
	{
		_toggleImage.sprite = _pressedSprite;
		_isActive = !_isActive;
	}

	public void OnPointerUp(PointerEventData eventData)
	{
		_toggleImage.sprite = _isActive ? _onToggleSprite : _offToggleSprite;
	}
}
```

Лістинг SettingsController.cs:

```
using UnityEngine.UI;
using UnityEngine;
using GrislyTools;

public class SettingsController : MonoBehaviour
{
	[SerializeField] private Button _openButton;
	[SerializeField] private Button _closeButton;
	[SerializeField] private PuzzleTimer _timer;
```

```
[Header("Buttons")]
[SerializeField] private Button _musicButton;
[SerializeField] private Button _soundButton;
[SerializeField] private Button _tapticButton;

[Header("Toggle anim")]
[SerializeField] private SettingButtonAnimToggle _musicToggle;
[SerializeField] private SettingButtonAnimToggle _soundToggle;
[SerializeField] private SettingButtonAnimToggle _tapticToggle;

private void Awake()
{
        Initialize();
        _openButton.onClick.AddListener(OpenButtonHandler);
        _closeButton.onClick.AddListener(CloseButtonHandler);
        gameObject.SetActive(false);
}
private void Initialize()
{
        _musicButton.onClick.AddListener(OnClickMusicHandler);
        _soundButton.onClick.AddListener(OnClickSoundHandler);
        _tapticButton.onClick.AddListener(OnClickTapticHandler);

        DataManager.Data.GetValue("Music", out bool music);
        DataManager.Data.GetValue("Sound", out bool sound);
        DataManager.Data.GetValue("Taptic", out bool taptic);

        _musicToggle.Initialize(music);
        _soundToggle.Initialize(sound);
        _tapticToggle.Initialize(taptic);

        SetMusicActivity(music);
        SetSoundActivity(sound);
}
#region HANDLERS
private void OnClickMusicHandler()
{
        DataManager.Data.GetValue("Music", out bool music);
        DataManager.Data.SetValue("Music", !music);

        SetMusicActivity(!music);
```

```
}
private void OnClickSoundHandler()
{
	DataManager.Data.GetValue("Sound", out bool sound);
	DataManager.Data.SetValue("Sound", !sound);

	SetSoundActivity(!sound);

}
private void OnClickTapticHandler()
{
	DataManager.Data.GetValue("Taptic", out bool taptic);
	DataManager.Data.SetValue("Taptic", !taptic);
}
#endregion

private void SetMusicActivity(bool enable)
{
	if (enable)
		GlobalSettings.SoundController.UnmuteMusic();
	else
		GlobalSettings.SoundController.MuteMusic();
}
private void SetSoundActivity(bool enable)
{
	if (enable)
		GlobalSettings.SoundController.UnmuteAllSFXSources();
	else
		GlobalSettings.SoundController.MuteAllSFXSources();
}
private void OpenButtonHandler()
{
	if (_timer != null)
		_timer.IsTimerActive = false;

	Time.timeScale = 0;
	gameObject.SetActive(true);
}
private void CloseButtonHandler()
{
	if (_timer != null)
		_timer.IsTimerActive = true;
```

```
                Time.timeScale = 1;
                gameObject.SetActive(false);
        }
        private void OnDestroy()
        {
                _openButton.onClick.RemoveAllListeners();
                _closeButton.onClick.RemoveAllListeners();
        }
}
```

Лістинг SoundController.cs:

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

[RequireComponent(typeof(AudioListener))]
public class SoundController : MonoBehaviour
{
        private AudioSource _backgroundSource;
        private List<AudioSource> _SFX_Sources;

        public void Initialize(int sourceCount)
        {
                _SFX_Sources = new List<AudioSource>();

                for (int i = 0; i < sourceCount; i++)
                {
                        _SFX_Sources.Add(gameObject.AddComponent<AudioSource>());
                }

                _backgroundSource = gameObject.AddComponent<AudioSource>();
        }

        public void PlayMusic(AudioClip music, float volume)
        {
                _backgroundSource.clip = music;
                _backgroundSource.volume = volume;
                _backgroundSource.loop = true;
                _backgroundSource.Play();
        }
        public void StopMusic() => _backgroundSource.Stop();
```

```
	public void PauseMusic() => _backgroundSource.Pause();
	public void UnpauseMusic() => _backgroundSource.UnPause();
	public void MuteMusic() => _backgroundSource.mute = true;
	public void UnmuteMusic() => _backgroundSource.mute = false;
	public void PlaySFX(AudioClip sfx, float volume, bool taptic = false)
	{
		var source = _SFX_Sources.FirstOrDefault(source => !source.isPlaying);

		if (source == null)
			Debug.Log("No available AudioSource to play SFX");

		source.clip = sfx;
		source.volume = volume;
		source.Play();

		if (taptic)
			Taptic.Light();
	}
	public void MuteAllSFXSources()
	{
		foreach (var source in _SFX_Sources)
		{
			source.mute = true;
		}
	}
	public void UnmuteAllSFXSources()
	{
		foreach (var source in _SFX_Sources)
		{
			source.mute = false;
		}
	}
}
```

Лістинг Taptic.cs:

```
using CandyCoded.HapticFeedback;
using GrislyTools;


public static class Taptic
{
	public static void Light()
```

```
        {
            DataManager.Data.GetValue("Taptic", out bool taptic, false);

            if (taptic)
                HapticFeedback.LightFeedback();
        }
}
```

Лістинг UIClickSound.cs:

```
using UnityEngine;
using UnityEngine.UI;


public class UIClickSound : MonoBehaviour
{
    [SerializeField] private Button _button;
    [SerializeField] private AudioClip _audioClip;
    [SerializeField] private float _volume;

    private void Awake()
    {
        _button.onClick.AddListener(PlaySoundOnClick);
    }

    private void PlaySoundOnClick()
    {
        GlobalSettings.SoundController.PlaySFX(_audioClip, _volume, true);
    }

    private void OnDestroy()
    {
        _button?.onClick.RemoveListener(PlaySoundOnClick);
    }
}
```

ДОДАТОК Б

ВІДГУК

на кваліфікаційну роботу бакалавра

на тему:

" Розробка ігрового додатка на платформі Unity з використанням шаблонів і патернів проектування"

студента групи 121-20з-1 Огаркова Віктора Юрійовича

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	Опис
Пояснювальні документи	
Огарков_Диплом.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Огарков_Диплом.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.zip	Архів. Містить коди програми і скомпільовану програму
Презентація	
Огарков_Презентація.ppt	Презентація кваліфікаційної роботи