МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ ГРИ ІЗ ЗАСТОСУВАННЯМ ПЛАТФОРМИ UNITY»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Б.В. Криштальов
(ініціали та прізвище)
Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Криштальову Богдану Вікторовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інтелектуальної гри із застосуванням платформи Unity

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Створення інтелектуальної гри за допомогою Unity.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	02.02.2024	
3.	Обробка методичних та теоретичних	23.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	25.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2024	

Студент _____ Б.В. Криштальов
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інтелектуальної гри із застосуванням платформи Unity»: 89 с., 18 рис., 17 джерел, 1 додаток.

ГЕЙМДИЗАЙН, ГРАФІЧНИЙ ІНТЕРФЕЙС, ПРОГРАМУВАННЯ, КОДУВАННЯ НА C#, ПРОТОТИПУВАННЯ, РОЗРОБКА ІГОР, ТЕСТУВАННЯ ГРИ, УПРАВЛІННЯ ПРОЄКТАМИ, ФІЗИКА ІГОР, UNITY.

Об’єкт дослідження – процес розробки інтелектуальної гри-пазлу на базі класичного Тетрису за допомогою платформи Unity.

Мета роботи: можливість створення гри-пазлу на основі Тетрису з використанням можливостей платформи Unity, в стилістиці 2D.

Метод дослідження – системний аналіз та моделювання, методи прототипування, методи тестування ігор.

SUMMARY

Bachelor's qualifying paper "Development of an Intelligent Game Using the Unity Platform": 89 pages, 18 figures, 17 references, 1 supplement.

GAME DESIGN, GUI, PROGRAMMING, CODING IN C#, PROTOTYPING, GAME DEVELOPMENT, GAME TESTING, PROJECT MANAGEMENT, GAME PHYSICS, UNITY.

The object of the study is the process of developing a puzzle game based on classic Tetris using the Unity platform.

The aim of the study is to possibility was created for a puzzle game based on Tetris using the unique capabilities of the Unity platform, in 2D style.

The methods of research are system analysis and modeling, prototyping methods, game testing methods.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасна індустрія відеоігор постійно розвивається, залучаючи все більше гравців та розробників. Ігрові двигуни, такі як Unity, RPGMaker, Unreal Engine, CryEngine, Source 2 стали невід'ємною частиною цього процесу, надаючи розробникам широкий спектр інструментів для створення різноманітних ігрових проєктів.

В умовах зростаючої конкуренції на ринку відеоігор, оптимізація процесу розробки та підвищення якості ігор є надзвичайно важливими аспектами, хоча деякі компанії беруть кількістю, а не якістю. Дослідження можливостей платформи Unity для створення інтелектуальної гри-пазлу на основі класичного Тетрису є актуальним та сприяє поглибленню в базові знання у галузі розробки ігор.

Мета дослідження. Створити та проаналізувати етапи розробки інтелектуальної гри-пазлу, натхненої класичним Тетрисом, з використанням можливостей гейм-енджину Unity. В реалізацію задачі входить створення інтерактивних елементів для покращення враження від геймплею.

Завдання дослідження. Для реалізації заданої мети виділено такі завдання:

- проаналізувати основні особливості та переваги ігрового двигуна, Unity;
- дослідити компанії, що використовують та розробляють ігри на ядрі Unity;
- сформувати найбільш підходящий ігрові матеріали для розробки інтелектуальної гри-пазлу;
- розробити прототип гри на основі класичного Тетрису;
- провести тестування гри для оцінки її придатності.

Об'єкт дослідження. Форуми Unity, вебсайти, вебсервіси.

Предмет дослідження. Актуальні інструменти та методи розробки, що використовуються в платформі Unity для створення сучасних інтелектуальних ігор-пазлів.

Методи дослідження. Системний аналіз та синтез, прототипування, порівняння, експеримент, тестування програмного забезпечення

Для виконання поставлених задач використовувалися електронні ресурси, розробка класичного тетрісу відбувалася в середовищі розробки Visual Studio Code та програмному рушії для ігор Unity.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань та одного додатку.

1 ІНДУСТРІЯ ІГОР: ОГЛЯД ТА ІСТОРІЯ

1.1 Проблеми сучасних компаній

Сердце будь якої сучасної компанії по розробці видео ігр та інді-розробника [2] є правильно обране та реалізоване ігрове ядро. Від цієї основи залежить майбутнє розробників їх можливості чи є подальший розвиток, перехід в кросплатформеність, вдала оптимізація. Основний тренд задають гіганти індустрії, через великий штат співробітників. Створюються підрозділи орієнтовані на вузьку спеціальність, через це створення ігор стає автоматизованим та утворюється конвеєр.

Яскравим прикладом компанії яка робить ігри заради прибутку та ставить їх на рейки автоматизму **Electronic Arts** майже всі їх ігри мають в собі систему **Pay To Win**. Ця система монетизації дозволяє користувачам купувати ігрові переваги за реальні гроші. Чим більше грошей користувач витратить, тим більше переваг він отримає у грі. При цьому немає жодних обмежень на суму грошей, яку можна вкласти в гру. Приклад таких ігор від видавця.

FIFA. EA випускає нову версію футбольного симулятора FIFA щороку. Хоча у кожній новій версії оновлюються склади команд і іноді незначно покращуються геймплей та графіка, ці зміни є незначними та погано впливають на оптимізацію проєкту, через це після релізу доводиться дороблювати все в поспіху через це з’являються все нові і нові баги. Така ситуація триває до виходу нової частини, а вид старої відмовляються.

Need for Speed. Гоночна серія має безліч випусків, і хоча не кожна гра виходить щорічно, частота релізів і механіки, що повторюються, призводять до критики за відсутність значних поліпшень та інновацій.

NHL. EA також випускає щорічні версії хокейного симулятора NHL (National Hockey League). Аналогічно FIFA та Madden NFL, ці ігри часто

отримують незначні зміни, зосереджені на оновленні складів та невеликих покращеннях у геймплей.

Така система працює завдяки фанатам які будуть грати лише із за назви та можливість зібрати свою улюблену команду з карток. В таких умовах втрачається креативність та інновації, відділи робітників компанії починають викладатися не на повну спостерігаючи лише за монетизацією проєкту в шкоду якості. Такі фірми не вносять нічого нового до світу ігор в них з'являються конкуренти, які адаптуються під ринок та тримають чіткий курс щоб внести інновації. Через це починають проявлятися всі проблеми головні з них конкуренція всередині компанії, проблеми з розробкою та запуском великих проєктів, тяжкість адаптації під нові реалії ринку.

1.2 Реалізація перших ігор

Ера розважальним аркадним автоматам, а саме пінбол та музичні автомати були предками на основі яких був розроблений перший електромеханічний комп'ютер. Завдяки розвитку технологій та ентузіастам був розроблений перший електромеханічний комп'ютер для ігор "Nimatron".

Він був пращуром сучасних відеоігор. Вперше представлений публіці на Всесвітній виставці 1939-1940 років, був створений Едвардом Кондоном. Наступним значним кроком стало створення та патентування «Розважального пристрою на основі електронно-променевої трубки» Томасом Голдсмітом і Естлом Манном у 1947 році. Воно вважається першим спеціально призначеним для гри пристроєм, що виводить зображення на екран.

На початку 1950 року був розроблений комп'ютер для гри HIM, Bertie the Brain та хрестики-нулики. Запрограмований на два види гри, людина з комп'ютером та автоматична гра (комп'ютер з комп'ютером).

У 1948–1950 роках Алан Тюрінг і Девід Чемперноун розробили

алгоритм шахової гри Turochamp, проте комп'ютери того часу були багатокористувацькими недостатньо потужними, щоб реалізувати цей алгоритм.

Nimrod. Модель зроблена компанією Ferranti та вперше представлена у 1951 році. Прототипом для створення Nimrod був на той час серйозного Ferranti Mark I, розробка кінцевого варіанту йшла майже рік з 1 грудня 1950 року по 12 вересня 1951 року.

1.2.1 Перша мультиплеєрна гра

Tennis for Two [7] (теніс для двох) – одна з перших комп'ютерних ігор з графічним інтерфейсом та перша розрахована на багато користувачів гра. Вона була розроблена фізиком Вільямом Хігінботамом у 1958 році для розваги відвідувачів Брукхейвенської національної лабораторії.

До 1960-х років розвиток обчислювальної техніки – від вакуумних ламп до транзисторів, а від транзисторів до інтегральних схем – зробив комп'ютери набагато потужнішими та доступнішими.

У 1961 році група студентів з клубу Tech Model Railroad Club (TMRC) при Массачусетському технологічному інституті використала новітній на той момент комп'ютер DEC PDP-1 для створення однієї з перших комп'ютерних ігор – космічного симулятора Spacewar! Ця гра стала знаковою подією в історії відеоігор. Її творці активно поширювали код гри серед інших користувачів комп'ютерів PDP-1 в університетах та лабораторіях, а компанія DEC використовувала Spacewar! для демонстрації можливостей свого комп'ютера потенційним покупцям. Хоча PDP-1 був дуже дорогим комп'ютером, вартістю 120 тисяч доларів США, і було продано лише кілька десятків таких машин, Spacewar! мала значний вплив на розвиток комп'ютерних ігор.

1.2.2 Система PLATO

Повна назва програмний алгоритм для автоматизованих операцій викладання. Перша система в світі для електронного навчання. Система була розроблена в університеті Іллінойсу була в експлуатації 40 років та нараховувала декілька тисяч терміналів по всьому світу, більш 10 мейнфреймів які підключені до однієї мережі. Головна задача виконання простих курсових робіт студентами університету. Під час використання PLATO [4] були введені такі базові поняття, як вебфорум, онлайн-тестування, електронна пошта, чат, миттєве повідомлення, віддалений робочий стіл та розрахована на багато користувачів гра. Саме для комп'ютерів PLATO у 1973 році було створено першу гру з тривимірною графікою Spasim, а також вперше адаптовано такі відомі ігри, як Маджонг та Солітер.

1.2.3 Перші ігрові консолі

У 1966 році інженер Ральф Бер за власною ініціативою почав роботу над прототипом телевізійного пристрою для ігор кодова назва Channel LP. Перші спроби знайти партнера серед великих компаній по виробництву телевізорів закінчилися невдало. У 1971 році Ральф разом з колегами вдалося підписати контракт з Magnavox. Прототип консолі під назвою Brown Box, перейшов на комерційний продукт та отримав назву Magnavox Odyssey.

Перша демонстрація консолі широкій публіці відбулася в 1972 році та надійшла у продаж в межах американського ринку у серпні 1972 року. На той час приставка при собі мала 13 ігор та відомі нам зараз симулятори футболу і тенісу. До комплекту консолі також входили різні аксесуари деякі з них паперові гроші та накладки на екран. Не зважаючи на велику ціну в 100 доларів консоль здобула визнання до 1975 року було продано понад 350 тисяч пристроїв.

1.2.4 Аркадні автомати

Індустрія аркадних автоматів [5] отримала нову хвилю популярності завдяки появі складних електромеханічних ігор. Випускаючи автомат Perisope у 1966 році японська компанія Sony прирекла себе на успіх. Він став популярним в у самій Японії так в США. Породивши безліч наслідувань. Це технологічне ускладнення і зростання інтересу публіки до аркадних автоматів підготували ґрунт для появи електронних аркадних автоматів у 1970-х роках.

Перший електронний аркадний автомат мав назву Galaxy Game, розроблений у 1971 році в Стенфордському університеті на базі комп'ютера PDP-11, для гри Spacewar.

Автомат Computer Space розроблений Ноланом Бушнеллом став першим серійно виробленим продуктом, хоча й не досяг великого успіху. Через складність гри компанія Nutting Associates не змогла підвищити планку продаж в 1500, оскільки для звичайних барів, де встановлювалися пінбольні та музичні автомати він не підходив.

Після цієї невдачі Бушнелл у 1972 році заснував власну компанію з виробництва аркадних автоматів Atari.

У 1972 році Atari випустила аркадний автомат Pong, розроблений інженером Алланом Алкорном. Ця система, набагато простіше та дешевша . Цим вчинком Atari запустив ланцюгову реакцію щодо створення власних грального автоматів.

Під час гонки з конкурентами та розробкою ігрового автомату Atari експеринтував з іншими типами ігор, а саме гоночний симулятор Gran Trak 10, Tank та космічна гра Space Race.

У 1975 році компанія Taito Corporation розробила ігровий автомат Gun Fight, був випущений в Японії та допрацьований Midway Games був відправлений до Америки як перший аркадним автоматом, у якому використовувався мікропроцесор.

1.3 Pac-Man

Pac-Man розроблена у 1980 році компанією Namco, кодував її Тору Іватані як гру для аркадного автомату. Незважаючи на свою легкість гра мала захоплюючий геймплей та завоювала популярність серед геймерів та звичайних людей.

У грі гравець управляє персонажем Pac-Man, який повинен збирати монетки у лабіринті, уникаючи зіткнення з ворожими привидами. Великі монети дають можливість налякати привидів та з'їсти, після цього вони потраплять у пастку на деякий час. Після закінчення дії здібності привиди знов стануть ворогами.

Після того як гравець не залишить жодної монетки на рівні, пакман переходить на інший із збереженням набраних очків, гра нескінченна, поразка в ній настає лише при втраті всіх додаткових життів.

В наш час досі є ті хто змагається в списку лідерів та проводять турніри з призами.

1.3.1 Змійка

Історія «Змійки» почалася на ранніх персональних комп'ютерах і аркадних автоматах, де гравець керував «змійкою», яка намагається з'їсти їжу (зазвичай представлену як яблука), намагаючись уникати зіткнення зі своїм хвостом або стінами ігрового поля. З кожним з'їденим яблуком змійка збільшувалася у розмірах, що робило гру дедалі складніше.

У 1997 році Nokia додала «Змійку» у свої мобільні телефони, починаючи з моделі Nokia 6110. Це був вибуховий момент для гри, оскільки вона стала доступною мільйонам людей по всьому світу і була вбудована в більшість телефонів Nokia в наступні роки.

Однією з особливостей «Змійки» було те, що вона не вимагала потужного заліза чи складних контролів. Грати в неї було просто і захоплююче, що зробило її популярною серед людей різного віку та категорій.

Незабаром «Змійка» стала іконою мобільних ігор, уособлюючи простоту та веселощі, які можна отримати від гри на мобільному пристрої.

З часом «Змійка» пережила безліч оновлень та варіацій, але її оригінальний формат залишається класикою та продовжує залучати гравців по всьому світу.

1.3.2 Тетрис

Тетріс [6] було створено у червні 1984 року радянським програмістом Олексієм Пажитновим у Москві. Пажитнов працював у Обчислювальному центрі Академії наук СРСР та розробив гру на комп'ютері Електроніка 60. Його мета була створити гру, яка була б захоплюючою та допомагала розвивати логічне мислення.

Назва тетріс походить від грецького слова «тетра», що означає «чотири», оскільки всі фігури в грі складаються з чотирьох сегментів, і слова теніс, улюбленої гри Пажитнова.

Тетріс потрапив на західний ринок завдяки угорським програмістам, які створили версію гри для IBM PC. Потім британська компанія Andromeda Software набула права на поширення гри у західних країнах. 1986 року Тетріс був представлений на виставці Consumer Electronics Show у Лас-Вегасі, де його помітила компанія Spectrum HoloByte, яка згодом випустила версію для IBM PC у США.

Гра швидко стала популярною серед співробітників обчислювального центру, а згодом і за його межами. Пажитнов та його колеги розробили версії гри для IBM PC та інших платформ.

Невдовзі гра привернула увагу поза СРСР.

1.4 Ігрові двигуни

Перший ігровий двигун у розумінні сучасної людини з’явився в 1983 році на ньому була написана гра в жанрі шутер з виглядом від першої особи. Через десять років побачить світ майбутня серія ігор DOOM. Програмна частина представляла з дитини набір розумних рішень щодо поділу навантаження компонентами гри таких як зіткнення об’єктів, відтворення звуків та графічних ресурсів. Після цього почалася активна розробка двигунів, вони поділялися на два види, які зроблені тільки для ігор і нічого більше не можуть, другий вигляд який може робити прорахунки як малювати дугу тощо. В наш час більшість двигунів є багатозадачними за рахунок відкритого вихідного коду. Тепер на кожному движку можна написати будь-який жанр гри, адже у відкритому доступі є можливість зміни бібліотек. За рахунок цього на просторах інтернету є різні модифікації коду, які додають щось нове при розробці. Така політика добре впливає на доступність двигуна і його легкість в освоєнні. Цим користуються невеликі компанії та розробники одинаки. При актуальності та розвиненості движка, він збирає основу фанатів і середовище розробки починає обростати різними модифікаціями, найчастіше це аддони для швидкого написання коду допомоги в орієнтуванні за кодом і візуальна оптимізація для зручної роботи. Але найголовнішою модифікацією є додавання візуального програмування. Замість звичайного програмування розробник реалізує механіки та логіку гри блоками, такий спосіб дозволяє створити гру людям не мають досвіду в програмуванні, також немає сенсу залишатися на такому середовищі розробки, реалізація любові коду через блоки буде значно довше і більше в розмірах так що для становлення повноцінним розробником потрібно хоча б базове знання мови C #, C ++. Візуальне програмування існує в межах двигуна для ознайомлення та створення перших легких проєктів. При використанні open source движка інді-розробник або компанія приймає користувальницьку угоду, в якій присутні пункт про комерційну діяльність та умови, які необхідно виконати при

розробці гри. Цей пункт передбачає, що розробник повинен сплатити зазначену суму. Види оплати використання є стандартним при досягненні певної планки продажів потрібного виплату у процентному від загальної суми прибутку або купівля підписки для становлення партнером, рідше всього використовується плата відразу за публікацію гри. Але цей пункт обійшов один двигун. Godot Engine використовує власну мову програмування при розробці ігор на ньому розробник сам вирішує потримати проєкт чи ні, тому що середовище розробки існує через систему донатів. Щоб уникнути такої долі і не переживати за зміну умов користування, великі компанії такі як Valve, RockStar, Electronic Arts, Activision / Blizzard, Ubisoft розробляють власні двигуни для реалізації своїх потреб у проєктах. Доступ до таких движків мають тільки співробітники компанії їх немає в загальному доступі, при влаштуванні в компанію в трудовому договорі цей момент обговорюється детально.

Кожна велика компанія мала причину розробки саме свого движка. Історія створення двигунів: компанія Rockstar при старті свого шляху використовувала двигун RenderWare, але через зміну власника погіршилася політика ліцензування. Тоді почалася розробка нині відомого двигуна Rage. він використовує систему динамічного розподілу ресурсів пам’яті, як це працює при русі двигун підвантажує те що знаходиться в межах видимості гравця і видаляє те чого в полі зору немає, тому двигун добре працює з безшовним відкритим світом, адже саме для цього він і створювався, також в двигун входить анімація людей використовуючи інструмент euphoria, який в режимі реального часу працює над анімацією нпс. Також нововведенням euphoria є створення анімацій за допомогою декількох ключових кадрів, інструмент сам зробить недостатню кількість кадрів. На движку Rage розроблені ігри: серія Grand Theft Auto, Red Dead Redemption та Red Dead Redemption 2.

Спочатку компанія Valve використовували Quake engine, маючи вихідний код вони переробляють на той момент найкращий двигун від Quake,

зміни були настільки значними що двигун отримав назву Source. Історія розробки двигуна починається в 1998 році. Завершуючи роботу над своєю першою грою, Half-Life, розробники зрозуміли, що мають ідеї для поліпшення її движка. Однак, щоб не ризикувати з майже готовим проєктом, вони вирішили поділити код движка. В результаті новий двигун отримав назву Source, а попередній став відомий як GoldSource. Через 11 років двигун отримав масштабне оновлення та перейшов на новий рівень отримавши в кінці 2. Source 2 має більший функціонал ніж попередник, на ньому були випущені Counter Strike 2 і перенесена Dota 2 ця подія отримала назву Reborn, ігри на Source серія Half-Life, Garry's Mod, Team Fortress 2, Portal.

Ігрові двигуни, які у вільному доступі, мають велику популярність. Відомими серед них є Unity, Unreal Engine, Godot, Construct, CryEngine, GameMaker: Studio, RPG Maker. Кожен із цих двигунів має свої особливості, переваги та недоліки, але їхнім найбільшим плюсом є наявність документації та великої кількості платних та безкоштовних курсів, що дозволяє новачкам створити власну гру, ознайомившись із основами. Завдяки цим інструментам були створені такі проєкти, як Hollow Knight, Kingdoms of the Dump, PlayerUnknown's Battlegrounds, Prey, Undertale. перерахований вище список ігор лише мала частина створених ігрових продуктів, індустрія ігор майорить досить стрімко і щодня движки розвиваються випускаючи нові версії. Короткий опис кожного двигуна наведемо далі.

Construct двигун для створення 2D-ігор, розроблений компанією Scirra. Construct дозволяє користувачам створювати ігри без необхідності писати код, використовуючи натомість візуальну систему подій та дій. Існує кілька версій Construct, включаючи Construct 2 та Construct 3, які працюють у браузері та забезпечують кросплатформенну підтримку. Двигун використовує систему блоків для створення логіки гри, що робить його доступним як для початківців, так і для досвідчених розробників. Construct дозволяє експортувати ігри на різні платформи, включаючи HTML5, Android, iOS, Windows, MacOS та інші.

RPG Maker [8] серія програмних продуктів для створення рольових ігор

(RPG), розроблена японською компанією Enterbrain. Програма дозволяє користувачам без глибоких знань програмування створювати власні ігри. Існує безліч версій RPG Maker, починаючи з RPG Tsukūru Dante 98 і закінчуючи сучасними RPG Maker MV та RPG Maker MZ. Основна особливість RPG Maker полягає в його простоті використання та доступності для розробників-початківців. Інтерфейс програми інтуїтивно зрозумілий та надає різноманітні інструменти для створення карт, подій, персонажів та бойових систем. Користувачі можуть створювати свої ресурси або використовувати готові поставки. RPG Maker надає відмінні можливості для розробників-початківців, дозволяючи їм створювати захоплюючі і глибокі RPG ігри з мінімальними знаннями в галузі програмування. Завдяки простоті використання та широкому набору інструментів, це відмінний вибір для тих, хто хоче втілити свої ідеї в реальність.

GameMaker: Studio [10] інтегроване середовище розробки (IDE), створене компанією YoYo Games, яке дозволяє створювати 2D-ігри для різних платформ. Двигун надає візуальний інтерфейс для розробки ігор, підтримує скриптову мову GML (GameMaker Language), а також дозволяє використовувати Drag and Drop (D&D) функції для тих, хто не знайомий із програмуванням. GameMaker: Studio включає безліч інструментів для розробки ігор, таких як редактори спрайтів, тайлів, звуків, анімації та інших ресурсів. Також у ньому є вбудований відладчик та можливість експорту гри на різні платформи, включаючи Windows, macOS, Android, iOS та інші.

Двигун демонструє широкі можливості GameMaker: Studio у створенні різноманітних та захоплюючих ігор. Двигун дозволяє розробникам реалізовувати свої ідеї, створюючи ігри з унікальними механіками, візуальним стилем та атмосферою.

Godot [3] безкоштовний та відкритий двигун для розробки ігор, який завоював популярність завдяки своїй доступності та широкому набору інструментів. Він підтримує створення 2D і 3D ігор, надаючи розробникам зручний та інтуїтивно зрозумілий інтерфейс, а також гнучкий та потужний

мову програмування GDScript, схожий на Python. Крім GDScript, Godot можна використовувати C#, C++ і VisualScript, що робить його придатним для розробників з різними рівнями навичок і уподобаннями. Однією з ключових переваг Godot є його система сцен, що дозволяє модульно будувати ігри. Сцени може бути ієрархічно пов'язані і перевикористовуватися, що полегшує управління складними проєктами. Двигун також пропонує безліч інструментів для анімації, фізики, інтерфейсу користувача та мережевої гри, що дозволяє створювати високоякісні проєкти без необхідності використовувати сторонні плагіни. Godot активно використовується для створення безлічі різноманітних ігор.

CryEngine двигун розроблений спеціально для шутера Far Cry, під час свого виходу випередив усіх у технічному та візуальному плані. Через 5 років вийшов CryEngine 2. На базі цього ядра був розроблений Crysis, але дана версія програвала в якості картинки минулої версії. На CryEngine 4 була зроблена Ryse: Son of Rome. У 2016 році компанія змінила свою політику, випустивши CryEngine 5. Його розбудова відбувалася безкоштовно для залучення нових розробників. CryEngine 5 підтримує розробку VR проєктів дозволяє оптимізувати рендеринг та просатки продуктивності.

Unreal Engine ігровий двигун розроблений Epic Games. Перша гра створена на ньому називалася unreal випущена у 1998 році. Як і Frostbite був розроблений для шутерів від першої особи але пізніше він став універсальним та використовувався у жанрах файтингу та мморпг. Unreal Engine підтримує всі популярні ігрові платформи, включаючи консолі та мобільні пристрої. Серед ігор зроблених на ньому Borderlands 3, Dead Island 2, The Sinking City, Fortnite, серія ігор метро. Також у ньому є свій редактор, заточений під професійне створення анімації. У двигуні передбачено закріплення логічних зв'язків між об'єктами, фотореалістична глибина різкості, процедурна генерація рослинності, інструменти для перекладу написів на текстурах.

Unity [1] двигун на якому випущені ігри найрізноманітніших жанрів. Має безкоштовну базову версію, також має обхват у 24 платформи включаючи

в себе VR, консолі, мобільні пристрої, вебплатформи та комп’ютери. Unity має в своєму розпорядженні інді розробників за рахунок меншої праці і грошової витратності. На ньому були зроблені такі ігри Pathologic 2, Cuphead, Hearthstone, Pillars of Eternity Двигун порівняно простий для розробника-початківця але при цьому не відстає в технічних новинках. Він має візуальну середу розробки та конструктора для створення ігор. У юніті можна подивитися на результат роботи у реальному часі. Він підтримує написання шейдерів і має всі компоненти для створення мультиплеера, Також двигун отримує багато популярних форматів. Моделі, звуки, скрипти, текстури можна запаковувати у формат Unity assets та передавати іншим розробникам. Цей же формат використовується у внутрішньому магазині Unity Asset Store. У якому користувачі викладають свої напрацювання безкоштовно або за гроші. Також юніті погано працює із сторонніми бібліотеками. Перша версія Unity з’явилася в 2005 році для макбуків, але через час стала доступна підтримка Windows. Згодом були додані нові платформи та посилювалася деталізація та посилювалася деталізація та додавали нові функції на даний момент це один з швидко розвиваються і доступних двигунів.

1.5 Види графіки в іграх

Графічний стиль гри вибирають виходячи з лора, жанру та технічних особливостей, але так було назавжди. На початку створення ігор розробники мали обмеження в обсязі пам’яті пристроїв та їх розрахункової потужності. Протягом усього часу існування відеоігор використовувалися різноманітні види відображення графіки. Певні методи та стилі графіки змінилися з часом, чи припинили своє існування через відсутність в них потреби. За час модернізації потужності комп’ютерів, а саме центральних та графічних процесорів, графіка стрімко почала розвиватися обмеженість в ресурсах все менше впливала на розробку гри. Завдяки чому почали експериментувати та

створювати нові види графіки. За всю історію ігор використовувалися наступні стилі pixel art, flat design, 2d та 3d стилі, Вексельні рушії, 3d cartoon, low poly, вигляд від першої особи та третьої особи, cel-shading.

Pixel Art Мистецтво створення зображень, персонажів та навколишнього світу у вигляді квадратних або прямокутних пікселів. В ігровій промисловості воно часто використовується для візуального оформлення ігор, особливо в жанрах, де важлива ностальгійна атмосфера або обмеження технічних можливостей.При розробці для збільшення чіткості та більш детальної обробки, використовують наступні формати 8, 16 32, 64, 128, 256 бітів. Завдяки не великому розміру файлів розробник може зробити більше анімації. рівнів. Для створення спрайтів до гри використовують Aseprite, Paint, Photoshop. SAI.

Ігри розроблені з такою графікою Stardew valley, Terraria, Dave the diver, Balatro.

2D style стиль графіки в іграх є візуальним підходом, в якому всі елементи гри відображаються в двовимірній площині перспектива зверху вниз, без використання тривимірної графіки або ефектів. Цей стиль може змінюватись від піксель-артових ігор з ретро-видом до більш сучасних та деталізованих 2D ігор з деталізованими спрайтами та анімацією. Така візуалізація дозволяє оптимізувати проєкти для більш широкого спектру гаджетів до них входить мобільні пристрої та більш старих комп'ютерів. Приклади ігор з 2D стилем графіки Super Mario Bros, Hollow Knight.

3D style вид комп'ютерної гри, яка використовує тривимірну графіку для представлення ігрового світу та персонажів. На відміну від 2D ігор, які відображаються на двовимірній площині, 3D ігри створюють ілюзію глибини та простору, дозволяючи гравцям вільно пересуватися по тривимірному середовищу та взаємодіяти з ним. Всі об'єкти та оточення у грі створюються за допомогою тривимірних моделей, які можуть бути переглянуті з різних кутів і обертаються в просторі. Гравці мають можливість вільно переміщатися по тривимірному середовищі, досліджувати світ гри та взаємодіяти з різними

об’єктами та персонажами. 3D ігри часто прагнуть реалізувати візуальний стиль, використовуючи складні системи освітлення (RTX) і тіней для створення ефекту об’ємності і глибини. Багато 3D ігри мають реалістичні фізичні моделі, які визначають поведінку об’єктів та персонажів в ігровому світі, такі як гравітація, колізії та динамічна взаємодія. Такі ігри надають гравцю різні механіки та можливості для взаємодії гравців з навколишнім світом, такі як збирання предметів, вирішення головоломок. Приклади ігор з 3D стилем графіки Portal, The Witness, Fez, Deep Rock Galactic, The Witcher 3: Wild Hunt.

Стилізований реалізм візуальний стиль, що поєднує елементи реалізму з художньою стилізацією. У такому стилі розробники прагнуть створити візуально привабливий світ, який зберігає певний ступінь реалізму, але має унікальний художній підхід чи стилізацію. Оточення, персонажі та об’єкти можуть бути відображені з певним ступенем реалізму, з урахуванням деталей та освітлення, щоб створити відчуття присутності та імерсії. Незважаючи на реалістичність, гра має унікальний візуальний стиль чи художню стилізацію, яка відрізняє її від простого копіювання реального світу. Це може бути виражено через вибір палітри кольорів, спрощені форми або абстрактні елементи.

Cel-shading техніка відображення графіки, яка імітує стилізований художній стиль коміксів чи мультфільмів. На відміну від реалістичного відображення, де об’єкти та персонажі мають різні відтінки та тіні, при використанні cel-shading об’єкти представлені у вигляді плоских областей кольору з чіткими контурами, як це робиться в анімації. У cel-shading контури об’єктів та персонажів чіткі та певні, що створює враження, ніби вони були намальовані рукою художника. Кольори в cel-shading зазвичай яскраві та насичені, а контрасти між ними виражені сильніше, ніж у реалістичному відображенні. Cel-shading широко використовується в анімованих фільмах, відеоіграх та коміксах для створення унікального візуального стилю та підкреслення естетики.

Ігри з використанням cel-shading серія ігор Borderlands, Jet Set Radio, Okami.

3D cartoon окремий вид 3д графіки, що прагне створити веселу і доброзичливу атмосферу. У даному стилі присутні яскраві та насичені кольори незвичайні та спрощені форми персонажа та оточення. Також застосовується художній стиль для надання більшої мультяшності. Використання 3D cartoon стилю Fortnite, Ratchet & Clank.

Low poly один з видів тривимірної графіки навмисним зменшенням полігонів. Стиль був популярний у ранніх 3D-іграх і досі використовується для створення унікального візуального ефекту та покращення продуктивності. Моделі складаються з мінімального числа трикутників (полігонів), що надає їм спрощеного, незграбного вигляду. Використання низькополігональних моделей дозволяє заощаджувати обчислювальні ресурси, що особливо корисно для мобільних ігор та ігор з великими відкритими світами. Через малу кількість полігонів моделі легше обробляються графічним процесором, що покращує продуктивність гри. Не підходить для ігор з фотографічною графікою. Також анімація низькополігональних моделей може бути менш плавною та деталізованою.

Low poly ігри Totally Accurate Battle Simulator, Ravenfield, Astroneer.

Воксельна графіка використовує тривимірні пікселі (вокселі) для створення зображень. Цей стиль часто застосовується для створення ігор із ретро-естетикою або унікальним візуальним стилем. Воксельна графіка часто використовується в іграх із процедурною генерацією світу, оскільки вокселі легко обробляються та модифікуються алгоритмами. Воксельні ігри зазвичай мають спрощений, блоковий стиль, який надає їм унікального та впізнаваного вигляду. Воксельна графіка надає унікальні можливості для створення інтерактивних та процедурно згенерованих світів. Незважаючи на певні обмеження в деталізації та продуктивності, її переваги у простоті модифікації та зберіганні даних роблять її популярним вибором для ігор, де креативність та дослідження займають центральне місце.

Ігри з використанням воксельної графіки Minecraft, Trove, Brick Rigs.

1.5.1 Поглиблення в 3D моделінг

Окремою темою розмови є галузь 3д [9] як мистецтва. Паралельно з розвитком мов програмування та двигунів покращувалося та оптимізувалося середовище для створення тривимірних об’єктів. Початкове моделювання разюче відрізняється від того яким ми його бачимо зараз. Адже за весь час існування проводилася робота над помилками та можливостей зростання побільшало з розвитком технологій. За весь цей час індустрія стала доступнішою створивши велику кількість програм для роботи платних і безкоштовних знайшовши при цьому безліч відгалужень в індустрії. На даний момент актуальні такі програми Blender, ZBrush, Maya, 3ds Max, Cinema 4D, Lumin, FreeCAD. Кожна з них має своє призначення ставитися до них скульптинг, моделювання, технічне моделювання.

Скульптинг схожий на ліплення із глини. Немає обмежень на кількість полігонів, об’єкти створюються з нічого. Створення відбувається за допомогою кистей їх кількість досить велика, так ще є можливість додавання нових кастомних, вони можуть разюче відрізнятися від стандартних заданих програм, їх можна придбати на спеціалізованих сайтах. На самому початку створення моделі автор вибирає між блокінгом і скульптингом.Скульптинг це перші кроки моделі у світ її сітка не оптимізована, але за рахунок втрати якості сітки модель набуває дрібними деталями.

Після скульптингу модель отримує контур квадратами так називаються полігони в двигунах для ігор по типом Unity вони розбиваються на трикутники. У предельно правильно сформованої сітки має бути мінімальна кількість трикутників до завантаження в двигун, повна відсутність N гонів, фігури що мають більше 4 граней і сітка повинна розподілятися рівномірно. Саме цей етап має сильний вплив на фінальну реалізацію модему. Під час неправильної розробки сітки починаються проблеми з анімацією, накладенням текстур та карт нормалей.

Для створення анімації до моделі застосовується модифікатор кістки. Логіка прив’язки кісток до моделі, як у скелета. На цьому етапі фіксується

ступінь еластичності елементів. За допомогою схеми модель, що покриває, користувач регулює значенням діапазону кольорів синій, блакитний, зелений, жовтий, червоний чим яскравіше колір тим сильніше вона реагує на рух кісток.

Перед фінальним етапом створення моделі є текстуринг, запікання тіней. Цей етап робить безбарвну модель насиченою. При налагодженні тіней відбуватиметься uv розгортка моделі, модель розділяти на частини для правильного розподілу текстур. Для ігрових моделей створюються два види моделей high та low poly, високополіоганальна модель використовується для трейлерів, катсцен та рекламної акції щоб залучити майбутніх споживачів. Другий вид моделі використовується в самій грі, модель має менше полігонів за рахунок чого нагрузка на комплектуючі комп'ютера знижується. Візуально вона не поступається high poly, так відбувається при дотриманні ряду правил, створення карти нормалей для лоу полі моделі відбувається шляхом запікання текстурами високополігоанальної моделі на низькополігоанальну. Після таких маніпуляцій відбувається візуальне поліпшення без вишнього навантаження на систему. Финалом створення моделі є її штучне старіння, для цього є спеціалізовані програми від Adobe і безкоштовні аналоги з меншою різноманітністю елементів. Старіння відбувається для кожного об'єкта по-різному для залізних додавання іржі, одязі надають зношений вигляд з додаванням об'ємних принтів і т.д. Майстерність в даній справі безпосередньо впливає на результат і відмінного від початківців відрізнити дуже легко. Ця частина потрібна для того, щоб не губилася атмосфера кінцевого продукту, адже ідеально чисті і не подерті штани в давно занедбаному будинку не синергують з оточенням.

Blender є швейцарським ножем у світі створення моделлю, а все тому, що в ньому зібрані всі види від створення моделі та її скульптингу до створення 3D/2D анімації. Розробники роблять програму з кожним оновленням простішим для освоєння. Останні велика зміна принесла повну переробку системи node, має схожість із візуальним програмуванням. По цьому середовищі розробки дуже багато навчального матеріалу. Також

розповсюдження відбувається на безоплатній основі. У 2019 році компанія Epic Games стала спонсором, виділивши на розвиток 1.2 мільйона доларів на розвиток проєкту.

1.5.2 Інструменти для створення 2D графіки

Створення 2D-графіки включає використання різних інструментів та програм, які призначені для різних завдань – від створення ілюстрацій та дизайну інтерфейсів до анімації та обробки фотографій. Список найбільш популярних та широко використовуваних програм для 2D-графіки: Adobe Illustrator, Krita, Figma.

Krita безкоштовне програмне забезпечення для створення 2D-графіки, особливо популярне серед художників, що працюють у галузі цифрового живопису та концепт-арту. Krita дозволяє настроювати робочу область під свої потреби, включаючи розташування панелей, палітр та інструментів. Можна зберігати та перемикатися між різними робочими просторами, що є корисним для різних типів завдань, таких як малювання, анімація та композитинг. Krita пропонує безліч встановлених пензлів для різних стилів малювання, а також можливість створювати та налаштовувати свої власні пензлі. Програма підтримує текстури та візерунки, які можна застосовувати до пензлів для створення різних ефектів. Масштабування, поворот, спотворення та інші трансформаційні інструменти дають змогу легко змінювати частини малюнка. Підтримка шарів дозволяє створювати складні композиції, змінюючи та коригуючі окремі елементи без шкоди для інших. Інструменти маскування дають більше контролю над процесом малювання та редагування. Krita має інструменти для імітації традиційних матеріалів, таких як акварель, масло та вугілля, а функція стабілізації пензля допомагає малювати плавні лінії та криві, також надає можливості для створення та використання різних колірних палітр та підтримує роботу в різних колірних режимах, таких як RGB, CMYK

та LAB, що важливо для підготовки зображень до друку. Програма також пропонує інструменти для створення анімації, включаючи таймлайн та ключові кадри, що дозволяє створювати прості анімаційні проєкти. Krita відмінно працює з графічними планшетами, підтримуючи чутливість до натиску та нахилу пера, що дозволяє створювати більш природні та точні спрайти. Програма ідеально підходить для цифрового живопису, дозволяючи художникам створювати деталізовані та реалістичні цифрові картини. Вона також широко використовується для створення концепт-арту для ігор, фільмів та інших медіа завдяки потужним інструментом для малювання та роботи з кольором. Krita є потужним та гнучкий інструмент для створення 2D-графіки, який підходить як для професійних художників, так і для любителів. Його основні переваги включають широкий набір інструментів для малювання, підтримку шарів та масок, а також інтеграцію із графічними планшетами. Krita продовжує активно розвиватися, пропонуючи нові і нові можливості для своїх користувачів.

Illustrator програмне забезпечення серії Adobe для створення векторної графіки, яке широко використовується в ігровій індустрії для розробки різних графічних елементів. Завдяки своїм потужним інструментам та функціональності, Illustrator дозволяє створювати високоякісні та масштабовані графічні ресурси, що робить його ідеальним для розробки ігор. Однією з ключових переваг Adobe Illustrator для розробки ігор є робота з векторною графікою. Векторні зображення складаються з математичних кривих, що дозволяє їх масштабувати до будь-якого розміру без втрати якості. Це особливо важливо для ігор, де графічні елементи можуть використовуватися в різних дозволах та різних пристроях. Векторна графіка дозволяє створювати чіткі та деталізовані зображення, що виглядають однаково добре як на маленьких екранах мобільних пристроїв, так і на великих моніторах. Робота з кривими Безьє в Illustrator дозволяє розробникам ігор створювати та редагувати складні форми та контури з високою точністю. Інструменти, такі як Pen Tool (інструмент Перо), дозволяють створювати та

коригувати точки прив’язки та сегменти кривих, що дає повний контроль над формою та деталями графічних елементів. Це корисно для створення персонажів, об’єктів та фонів, які можуть вимагати високого ступеня деталізації та точності. Adobe Illustrator підтримує роботу з шарами, що полегшує керування складними проєктами. В ігровому дизайні це означає, що різні елементи, такі як персонажі, фонові об’єкти та інтерфейсні компоненти можуть бути організовані і згруповані в окремі шари. Спрощує редагування та дозволяє легко змінювати окремі елементи без впливу на інші частини проєкту. Колірні можливості Illustrator включають підтримку різних колірних моделей, таких як RGB, CMYK та Pantone, що дозволяє працювати з кольорами, що підходять для екранних та друкованих проєктів. В ігровому дизайні це важливо для створення яскравих графічних елементів, що привертають увагу. Інструменти для створення градієнтів, візерунків та текстур дозволяють додавати глибину та реалізм до об’єктів, що робить їх більш привабливими для гравців. Інструменти для роботи з текстом у Illustrator дозволяють створювати унікальні та складні типографічні композиції, що корисно для розробки інтерфейсів та ігрових меню. Можливості управління шрифтами, зміною їх розміру, інтервалів і вирівнювання дозволяють створювати чіткі та легкочитані текстові елементи, які можуть бути використані в різних аспектах ігрового інтерфейсу. Adobe Illustrator інтегрується з іншими продуктами Adobe Creative Cloud, такими як Adobe Photoshop та Adobe After Effects. Це дозволяє легко переносити елементи та ресурси між різними програмами, що особливо корисно для створення комплексних ігрових проєктів, що включають анімацію та ефекти. Наприклад, персонажі та об’єкти, створені в Illustrator, можуть бути анімовані в After Effects, що додає додатковий рівень інтерактивності та візуальної привабливості. Додаток підтримує плагіни та розширення, які дозволяють додавати нові функції та інструменти, розширюючи можливості програми. Розробники ігор можуть використовувати скрипти і автоматизувати завдання, що повторюються, що прискорює процес створення графіки і покращує

продуктивність. Adobe Illustrator є незамінним інструментом для розробки ігор, який дозволяє створювати високоякісні та масштабовані графічні ресурси. Його основні переваги включають точні інструменти для роботи з кривими, широкі можливості для роботи з кольором та текстом, підтримку шарів та символів, а також інтеграцію з іншими продуктами Adobe. Завдяки цим можливостям Illustrator продовжує залишатися важливим інструментом для професійних розробників ігор та художників.

1.6 Жанри ігор

Жодна сфера розваг не зрівняється з відеоіграми щодо різноманітності. Ігри з кількістю своїх жанрів обійшли кіноіндустрії, телебачення, книги. Класифікація гри по жанру відбувається через взаємодію з гравцем, механіки та цілі [13]. Кожен жанр є унікальним також присутнє комбінування для розробки глибини механик, сюжету та занурення гравця. Кожен жанр має свої особливості які приваблюють певну аудиторію, пропонуючи унікальні ігрові досліди та стилі оповідання.

Ігри діляться на жанри:

- shooter;
- RPG;
- stealth;
- MOBA;
- strategy;
- adventure;
- visual novella;
- racing Simulator;
- roguelike.

Шутер (Shooter) основний ігровий процес зосереджений на стрільбі та знищенні ворогів,формат геймплея поділяється як від першої особи та третьої

особи. Гравці керують персонажем або транспортним засобом, використовуючи різні види зброї задля досягнення цілей. Шутери можуть бути як одиночними, так і розрахованими на багато користувачів, з акцентом на швидкі рефлекси і точність. Ігри даного жанру пропонують широкий арсенал зброї та боєприпасів, а також пасток які гравці можуть використовувати для більшої ефективності знищення ворогів та отримання переваги чисельного чи позиційного. Також шутери мають свої піджанри та їх об'єднання. Самий відомий Battle Royale(Королівська гра) Мета залишитися останнім, хто вижив на карті. До ігрової особливості даного жанру відносять поступове зменшення ігрової зони та систему луту.

Рольові ігри (RPG) пропонують багатий та різноманітний ігровий досвід, поєднуючи глибокі сюжети, складні механіки та великі світи для дослідження. Завдяки різноманітності піджанрів, кожен гравець може знайти RPG, яка йому до душі, чи то лінійна JRPG із захоплюючим сюжетом, чи відкрита WRPG із величезною свободою дій. Гравці розвивають персонажів, підвищуючи їхній рівень, покращуючи навички та здібності, а також екіпіруючи їх різними предметами та зброєю. Основний метод просування за сюжетом та отримання нагород. Квести можуть бути основними (сюжетними) та побічними, іноди завдяки таким квестам відкривається нова можливість відповіді в діалозі з NPS). Ігри такого жанру часто включають складні системи діалогів, де вибір гравця може вплинути на сюжет та розвиток подій. Свобода дій дає гравцеві можливість знаходити приховані локації, досліджувати світ гри Жанр RPG ігор ділиться також на піджанри **JRPG**. Відрізняються стильною анімацією, часто включають лінійний сюжет та покрокові бої. **WRPG**.Включають відкриті світи для дослідження, нелінійні сюжети та більшу свободу дій. **Экшен-RPG**. Поєднують елементи екшен-ігор та RPG, з акцентом на динамічні бої **Тактические RPG**. Включають покрокові битви на тактичних картах, де важливі стратегія та планування, приклади проєктов XCOM, Final Fantasy Tactics. **MMORPG**. Масові розраховані на багато користувачів онлайн-рольові ігри, де тисячі гравців взаємодіють в одному світі.

Стелс (Stealth) жанр відеоігор, у яких основний акцент робиться на потайливе переміщення та уникнення виявлення ворогами. Гравці повинні використовувати тактику скритності, щоб досягати цілей, уникати прямих зіткнень та мінімізувати шум. У таких іграх є система досягнень за проходження без жодного вбивства тощо. Прихованість досягається за рахунок використання укриттів, тіней та елементів оточення. Також не буде виключена система взлому, крадіжки за допомогою використання спеціальних інструментів та підручних засобів. Додають також систему прокачування, або накопичення рівнів до наявних здібностей посилюючи їх.

MOBA розрахованих на багато користувачів онлайн-ігор, в яких команди гравців борються на спеціально створених картах, прагнучи знищити базу противника. Найчастіше карти зроблені дзеркально для двох сторін. У таких іграх гравці б’ються 5 на 5. Кожен герой має навички і має власну роль і позицію, є прокачування та золото для купівлі предметів з кожним рівнем герой ставати сильнішим. Присутня система балансу та умов, тому на кожного персонажа є протидія. Також є ключові етапи матчу які вирішать результат матчу. Після гри нараховуються очки рейтингу для підживлення бажання грати далі.

Стратегії (Strategy) жанр ігор, де гравці повинні планувати та приймати рішення для досягнення довгострокових цілей. Включає управління ресурсами, будівництво баз і ведення бойових дій. Гравці повинні добувати ресурси та керувати ними для розвитку своєї бази та армії. Також налагоджувати відносини з іншими країнами. Володіти вмінням контролювати окремих юнітів, встигати перемикатися на кожного та на всіх разом, мати розвинене критичне мислення і планувати все наперед для захоплення більшої кількості територій.

Пригодницький (Adventure) одним з найпопулярніших та різноманітних жанрів, пропонуючи гравцям захоплюючі сюжети, багатий геймплей та незабутні світи для дослідження. Пригодницькі ігри часто поєднують елементи інших жанрів, таких як екшен, головоломки та рольова

гра, створюючи унікальні та багатошарові ігрові досліди. У центрі уваги пригодницьких ігор часто стоїть захоплююча історія, сповнена несподіваних поворотів та інтриг. Сюжет відіграє ключову роль, залучаючи гравця до подій та мотивуючи його рухатися вперед. : гравцям надається можливість дослідити великі ігрові світи, знаходити приховані об’єкти, секрети та вирішувати головоломки Місця можуть бути як реальними, так і вигаданими. Для просування за сюжетом гравці часто стикаються з різними головоломками та завданнями, які потребують логічного мислення та уважності. Приклади відомих пригодницьких ігор Uncharted, серія ігор, розроблена компанією Naughty Dog, пропонує захоплюючі пригоди Нейтана Дрейка, дослідника та мисливця за скарбами. Ігри відрізняються кінематографічним стилем, захоплюючими екшн-сценами та чудовою графікою. Tomb Raider, серія ігр про Лару Крофт, археолога та авантюристку, яка подорожує світом у пошуках стародавніх артефактів. Ігри поєднують екшн, платформінг та головоломки.

Візуальна новела (Visual novella) жанр ігор, зосереджений на розповіді та діалогах, з використанням статичних зображень та тексту. Акцент робиться на розвитку персонажа як особистості. Доступна складна гілка діалогів, що впливає на розвиток гілки сюжету. Текст супроводжується статичними, трохи анімованими зображеннями. За рахунок різних гілок сюжету на які впливає гравець своїм вибором, доступна велика кількість закінчення сюжетної лінії. Має велику кількість доповнень від спільноти за рахунок чого має майже нескінченну рентабельність.

Рогалик (Roguelike) жанр ігор заточений під реграбельність. Має велику кількість механік одна з них відродження після смерті в початковому таборі (точці). При кожному новому запуску перемикання гравець може покращити себе, змінивши початкові параметри або змінивши спорядження. Також у таких іграх реалізовано збільшення складності поступово залежно від того, наскільки далеко зайшов гравець, у таких ситуаціях щоб завершити цикл треба мати при собі хороші предмети та досвід гри. При повному проходженні циклу гравець змінює зброю і починає спочатку. Такі ігри побудовані

процедурною генерацією, кожен новий забіг унікальний. Приклад збільшення складності у грі Dead Cells, після повного проходження гравець отримує «клітину мутації» починаючи з наступного забігу, з’являтимуться нові противники міні-боси отримувати вселення вже наявні вороги, боси поверхів також зазнають змін. При отриманні максимальної кількості клітин у гравця з’являється можливість.

Гоночний симулятор (Racing Simulator) велику увагу приділено управлінню машиною, її зчеплення з дорогою. Повний тюнинг на авто. Доступно стандартне керування (клавіатура та миша) або альтернативне кермо, коробка передач та педалі. Є два види гоночних симуляторів з відкритим світом Forza, де гравець сам вибирає, де і чим саме хоче займатися. Або більш простий варіант серія ігор Rally гравець може вибрати лише траси та змагатися з іншими гравцями на час подолання траси та займати місця у таблиці лідерів. У Forza всі супротивники у заїзді штучний інтелект на випадково отриманих машинах.

1.7 Мови програмування

Мова програмування являє собою формалізований спосіб запису команд для комп’ютера, який визначає правила написання програм та дії, які комп’ютер витрачає на їх виконання. З появою перших програмованих пристроїв було створено понад вісім тисяч мов програмування. Деякі з них використовуються лише вузьким колом розробників, тоді як інші стають широко відомими у світі. Професійні програмісти часто володіють кількома мовами програмування. Ці мови призначені для написання програм, які управляють обчислювальними процесами, обробляють різні об’єкти тощо. На відміну від природних мов, що використовуються спілкування для людей між собою, мови програмування спрямовані на взаємодію з Електронно обчислювальною машиною. Більшість з них мають спеціальні конструкції для

роботи з даними та управління обчисленнями. Крім того, мова програмування визначається не тільки через його стандарт, який формально описує його синтаксис та семантику, але й через програмні засоби, що забезпечують його трансляцію чи інтерпретацію.

1.7.1 C#

C [11] мова програмування, створена в Microsoft на початку 2000-х років. Розроблений як ключова частина платформи .NET, C# є потужним інструментом для розробки різноманітних додатків, від десктопних до вебсервісів та мобільних додатків. Історія C# почалася з появи як відповідь деякі недоліки, які мали інші мови програмування, такі як C++ і Java. Він поєднує в собі простоту використання з високою продуктивністю та безпекою, роблячи його привабливим для розробників усіх рівнів досвіду. Основні риси C#, такі як об’єктно-орієнтоване програмування, сильна типізація та можливість роботи на різних операційних системах, роблять його однією з найпопулярніших мов програмування в сучасному світі розробки програмного забезпечення.

Спільнота розробників ігор на базі C# дуже активно та широко. Існує безліч онлайн-ресурсів, навчальних матеріалів, форумів та спільнот, де можна отримати підтримку, поради та ресурси для розробки ігор на C#. Завдяки цим факторам та багатьом іншим, C# став однією з кращих мов програмування для створення ігор різного масштабу та типів. У світі C# існує багато бібліотек, які надають різні функції для створення програм. Наприклад, є бібліотеки для роботи з базами даних, створення вебзастосунків, обробки JSON-даних і навіть для написання тестів. Незалежно від того, що вам потрібно зробити, існує бібліотека, яка допоможе вам спростити це завдання. C# широко використовується у розробці ігор, особливо завдяки своїй інтеграції з різними ігровими двигунами. В Unity Engine C# використовується для написання

сценаріїв (скриптів), які управляють поведінкою об’єктів у грі. Ці скрипти можуть керувати рухом, взаємодією з ігровим світом, анімацією та багатьом іншим. Unreal Engine є два основних способи написання ігрової логіки: за допомогою візуальної системи Blueprint та з допомогою C++/C#.

1.7.2 C++

C++ [12] мова програмування, яка має багатий набір можливостей для створення різних видів програмного забезпечення. Хоча він не має такої широкої екосистеми бібліотек, як C#, у C++ все ж таки існують інструменти та бібліотеки для роботи з базами даних, веброзробки, мультимедіа та тестування. Він часто використовується для розробки високопродуктивного програмного забезпечення, особливо в областях, де потрібна максимальна продуктивність та контроль над ресурсами. З його допомогою вони можуть створювати ігрові движки, в яких реалізовано основну логіку гри, а також оптимізувати продуктивність для різних платформ. Крім того, C++ забезпечує доступ до різних бібліотек та API, що дозволяє розробникам інтегрувати різні функції, такі як графіка, звук та мережева взаємодія, у свої ігри. В цілому, C++ є важливим інструментом для створення якісних та продуктивних ігор на різних платформах.

1.7.3 Python

Мова програмування розроблена з урахуванням універсальності та доступності. Призначення Python універсальна, розробка вебдодатків та вебсерверів. Розробка штучного інтелекту він є основною мовою в програмуванні ii та машинного навчання. Автоматизація простих задач за допомогою скриштингу. Його легкий синтаксис ідеально підходить для

написання скриптів для оптимізації процесів в операційній системі, вебсервісах. Python надає інструменти для роботи з текстовими даними, обробка рядків та генерації звітів.

1.7.4 Java

Java мова програмування, яка використовується для створення різних типів програм, включаючи вебпрограми, мобільні програми для Android та багато іншого. Він надає розробникам безліч інструментів та бібліотек для роботи з базами даних, створення графічних інтерфейсів, обробки даних та багато іншого.

Завдяки своїй багатоплатформеність та високій продуктивності, Java залишається однією з найпопулярніших мов програмування у світі. Spring Framework один з фреймворків Java для роботи вебдодатків та вебсервісів. Він надає широкий спектр функцій, включаючи інверсію управління та впровадження залежностей, що робить його популярним вибором для створення великих та складних програм Java.

1.7.5 JavaScript

Одна з широко використовуваних мов програмування, яка застосовується у веброзробці для створення інтерактивних вебсайтів та вебдодатків.

Ось кілька бібліотек та фреймворків JavaScript, які часто використовуються при роботі над проєктами.

React.js бібліотека JavaScript для створення інтерфейсів користувача. React використовується для побудови компонентів інтерфейсу користувача, які оновлюються автоматично при зміні даних. Він часто використовується

для розробки односторінкових додатків (SPA) та мобільних додатків.

Angular фреймворк JavaScript, розроблений та підтримуваний Google. Він надає інструменти для створення великих та складних вебдодатків. Він включає безліч функцій, таких як ін’єкція залежностей, маршрутизація і обробка подій.

1.8 Нові технології для графіки в іграх

Нові технології для графіки в іграх постійно розвиваються, щоб забезпечити ігровий досвід із більш високою якістю візуальних ефектів та реалізмом [14].

Візуальні ефекти (VFX), це процес створення та впровадження у гру різних візуальних елементів, які покращують візуальний досвід гравця. Вони можуть включати спеціальні ефекти, такі як вибухи, вогонь, дим, сніг, дощ, а також поліпшення атмосфери і реалізму ігрового світу. VFX можуть бути створені за допомогою різних технологій та методів, таких як комп’ютерна графіка, симуляції фізики, захоплення руху, використання текстур та шейдерів, системи частинок та багато інших. Вони дозволяють розробникам ігор створювати вражаючі та захоплюючі візуальні ефекти, які роблять ігровий світ більш реалістичним, динамічним та цікавим для гравців.

Функція Ray Tracing (RTX) передова технологія у графічному процесі комп’ютерних ігор, яка моделює та стежить за кожним променем світла, дозволяючи їм реалістично взаємодіяти з навколишнім середовищем та об’єктами. Вона створює більш реалістичні візуальні ефекти, такі як реалістичні відображення, тіні та освітлення, що робить ігрові сцени більш живими та захоплюючими для гравців. В основі технології RTX лежить ідея симуляції того, як світло поширюється та взаємодії в реальному світі, що дозволяє створювати більш реалістичні та захоплюючі візуальні враження.

Deep Learning Super Sampling (DLSS) технологія від NVIDIA, яка використовує штучний інтелект для покращення якості графіки в іграх шляхом збільшення чіткості зображення з використанням методу суперсемплінгу. Це дозволяє покращити візуальний досвід гравців при зниженні навантаження на апаратне забезпечення, забезпечуючи більш плавну ігрову продуктивність.

DirectX набір програмних засобів та API, розроблений Microsoft для забезпечення взаємодії між програмним забезпеченням та апаратним забезпеченням комп'ютера, включаючи графічні та звукові можливості. Ці інструменти надають розробникам зручні способи створення мультимедійних програм, забезпечуючи ефективне управління графікою, звуком, а також введенням і виведенням даних.

Vulkan стандартний інтерфейс для роботи з графічними картами та іншими пристроями, який надає високу продуктивність та ефективне керування ресурсами. Створений Khronos Group, він дозволяє розробникам створювати швидкі та якісні графічні програми та ігри для різних платформ, таких як ПК, мобільні пристрої та консолі, надаючи більше контролю над процесом та оптимізацією продуктивності.

1.9 Програми для розробки коду

У процесі створення програмного забезпечення відіграють важливу роль інтегровані середовища розробки (IDE) – програмні інструменти, призначені для написання та налагодження коду. Вони надають широкий спектр функціональних можливостей, від редагування коду до керування версіями проєкту, забезпечуючи комфортне та ефективне робоче оточення для розробників. У цьому огляді ми розглянемо кілька популярних IDE, які широко використовуються у сфері програмування.

1.9.1 Visual Studio

Visual Studio набір інструментів, що допомагає розробникам створювати програмне забезпечення. Він забезпечує зручний робочий простір для написання коду, виявлення та усунення помилок, а також управління проєктами. Visual Studio також підтримує інтеграцію з різними сервісами, такими як системи контролю версій та хмарні платформи, що робить його найпопулярнішим вибором серед розробників програмного забезпечення.

1.9.2 Visual Studio Code

Visual Studio Code легковажний та гнучкий текстовий редактор, розроблений Microsoft, який надає розробникам середовище для написання коду та роботи з проєктами. На відміну від повної версії Visual Studio, Visual Studio Code надає універсальне та розширюване оточення для роботи з різними мовами програмування та технологіями. Він володіє потужним і гнучким редактором коду з підтримкою синтаксичного виділення, автодоповнення, інтеграції з системами контролю версій та багатьма іншими функціями, що робить його популярним вибором серед розробників програмного забезпечення, особливо тих, хто віддає перевагу легкості та гнучкості.

1.10 Висновки до розділу 1

У цьому розділі ретельно розглядаються ключові аспекти, які сприяли формуванню та розвитку цього динамічного поля. Спочатку розглядаються проблеми сучасних компаній, що працюють у сфері розробки ігор. Опис реалізації перших ігор надає історичну перспективу, показуючи, як прості текстові ігри еволюціонували у складні візуальні проєкти. Особливо цікавою

є інформація про першу багатокористувацьку гру, систему PLATO та перші ігрові консолі, які заклали основу для подальшого розвитку багатокористувацьких ігор та інтерактивних платформ. Розділи про аркадні автомати та культові ігри, такі як Pac-Man, Snake і Tetris, ілюструють, як ці ігри стали культурним феноменом і вплинули на покоління геймерів. Вони демонструють, як унікальний геймплей і дизайн можуть призвести до величезного успіху. Важливою частиною глави є опис ігрових движків, які відіграють ключову роль у створенні сучасних ігор, надаючи розробникам необхідні інструменти для реалізації складної ігрової механіки та високоякісної графіки. Детальний аналіз типів графіки в іграх, включаючи 2D і 3D моделювання, підкреслює еволюцію візуальних технологій і їх вплив на ігровий досвід. Особливу увагу приділено інструментам для створення 2D-графіки, які залишаються актуальними для інді-розробників і ретро-ігор.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ГРИ

2.1 Суть гри та її концептуальна основа

Тетріс відео гра розроблена у 1984 році, Олексієм Пажитновим. Суть гри полягає в тому, щоб управляти падаючими фігурами, які називаються тетроміно, і розташовувати їх у безперервні горизонтальні лінії. Коли лінія повністю заповнена без пробілів, вона зникає, і гравець отримує бали. Якщо лінія не вдається, фігури накопичуються, і гра закінчується, коли вони досягають верхньої межі ігрового поля.

Основна ідея тетрісу – поєднання простоти і стратегії. Гравцеві дається набір фігур, відомих як тетраміно, які падають на ігрове поле. Кожне тетраміно складається з чотирьох квадратних блоків, які можуть мати різну форму, наприклад лінії, квадрати або літери «T», «C» і «L».

2.2 Реалізація проєкту та середовище розробки

Проєкт тетріс буде розроблено мовою програмування C# та середовище розробки Unity. У розробці є ключові моменти. Розробок ігрового поля масштабом 10x20, де і відбуватиметься основною геймплею. Додати вже всім відомі типи фігур I, O, T, S, L, J. Написати функцію для створення повноцінного рандому з появою фігури. Згенерувати логіку фігур тетраміно для автоматичного зниження на ігровому полі. Реалізувати функцію для обертання та переміщення підрахунку множника швидкості тетрино. Реалізуйте перевірку зіткнень тетраміно з межами ігрового поля та іншими тетраміно за та в межах ігрової зони. Написати функцію для очищення заповнених ліній з поля та нарахування очок за дану дію, також розробити множник за комбо видалених ліній одночасно. Також потрібно створити

інтерфейс для зручності користувача. До інтерфейсу входять такі елементи. Лічильник для відображення поточних та нарахованих очок, показувати наступну фігуру, реалізація прискорення фігури при досягненні певних рівнів. Налаштування початкового етапу гри, написання циклу для оновлення поля та рахунку при додаванні очків та правового видалення ліній. Додавання можливості зміни кольору фігури за кожного нового рівня, додати кнопки для можливості поставити гру на паузу, кнопка для перемикання на 3д вигляд. Додавання анімації після закінчення гри.

Unity development має великий вибір форматів проєктів. У магазині ресурсів Unity також є великий вибір моделей, більшість з них безкоштовні, зроблені самими розробниками ігор, на даний момент цей магазин використовує систему упаковки файлів Unity через те, що двигун некоректно працює з сторонніми бібліотеками, також досить часто проходять акції та роздачі ассетів і моделей.

2.2.1 Проєктування інтерфейсу для гри тетріс

Розроблений мною інтерфейс містить кілька ключових елементів для комфортної гри. У центрі розташована ігрова зона, де відбувається основна дія гри. У верхній частині екрана знаходиться інформаційна панель, яка відображає наступну фігуру, що дозволяє гравцеві планувати свої дії наперед. Кнопка перезапуску гри розташована поруч з ігровим полем для більшої зручності. Після закінчення гри гравець може легко почати нову гру, натиснувши цю кнопку. Шрифт інтерфейсу розроблено спеціально для контрасту з фоном, що забезпечує високу читабельність і дозволяє легко розрізняти текстові елементи. Інформація, важлива для гравця, така як поточний рахунок, кількість видалених ліній та рівень, відображається на екрані. Це допомагає гравцеві відстежувати свій прогрес та адаптувати стратегію гри, враховуючи швидкість падіння фігур, яка змінюється в

залежності від рівня. Інтерфейс гри виконано в темних тонах, щоб зменшити навантаження на очі гравця і не викликати дискомфорт під час тривалої гри. Інтерфейс представлено на рисунку 2.1.

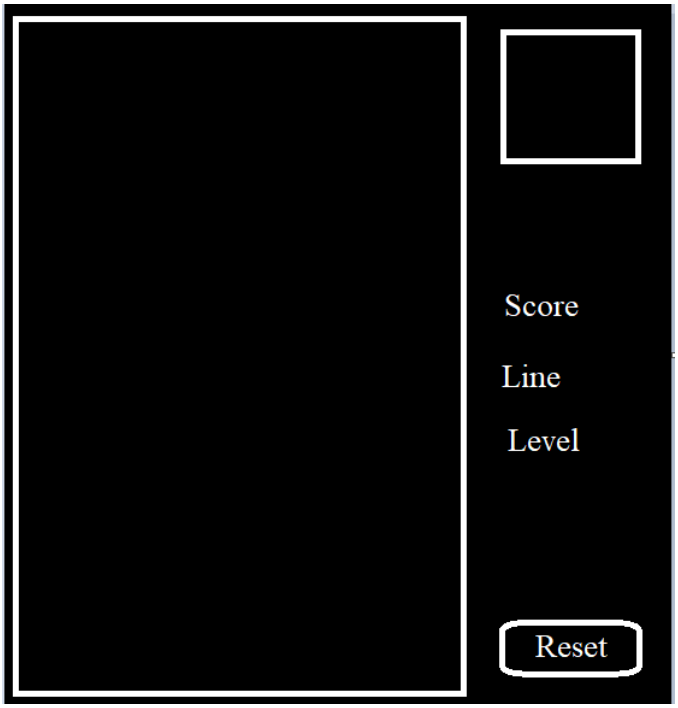

Рисунок 2.1 – Прототип інтерфейсу

2.2.2 Блок схема логіки Тетрісу

Початок гри, задаємо початкові параметри, завантажуємо ресурси та готуємо ігрове поле. Створення ігрової сітки формуємо порожнє поле з сітки осередків. Генерація випадкового тетроміно вибираємо одну з фігур (наприклад, I, O, T, S, Z, J, L). Відображення тетроміно у верхній частині сітки та готову до падіння. Перевіряючи введення користувача, очікуємо натискання клавіш для керування фігурою (вліво, вправо, поворот, прискорення падіння). Допустимий хід? Проводимо перевірку, чи допустиме переміщення фігури. Якщо так, продовжуємо. Так: Якщо хід допустимо: переміщення тетроміно, фігура переміщається згідно з введенням користувача. Повертаємось до перевірки введення користувача. Ні: Якщо хід

неприпустимий, фігура залишається на місці і не рухається. Перехід до наступного кроку. Перевірка завершених ліній, перевіряємо, чи є повністю заповнені горизонтальні лінії. Оновлення рахунку: Збільшуємо рахунок гравця за очищені лінії, не забуваючи про множник за кілька віддалених за раз. Генерація нової випадкової тетраміно фігури. Ні: Якщо завершених ліній немає: Генерація нового тетраміно Створюємо нову випадкову фігуру. Відображення тетраміно у верхній частині сітки. Перевірка завершення гри: Визначаємо, чи можна розмістити нову фігуру у верхній частині поля. Якщо ні, гра закінчена. Ні: Якщо гра не завершена, повертаємося до відображення нової фігури. Так: Якщо гра завершена, Виводимо на екран завершення гри: Показ екрану “Game Over”. Завершення роботи програми. Описану блок схему логіки тетрісу наведено на рисунку 2.2.

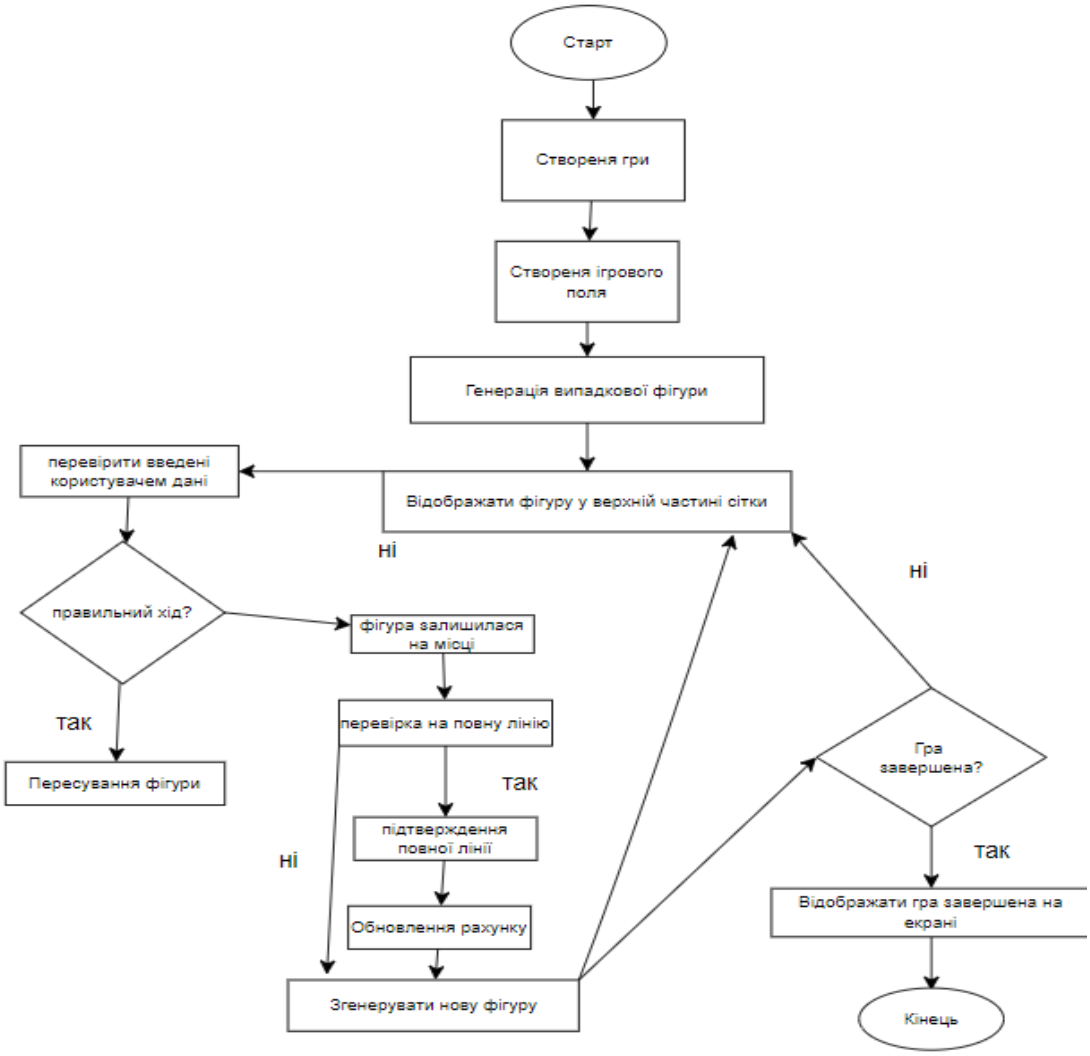

Рисунок 2.2 – Схема логіки тетрісу

Однією з найкорисніших діаграм в UML [15] є діаграма класів, яка точно відображає структуру системи, моделюючи її класи, властивості, операції та зв’язки між об’єктами, наведено на рисунку 2.3.

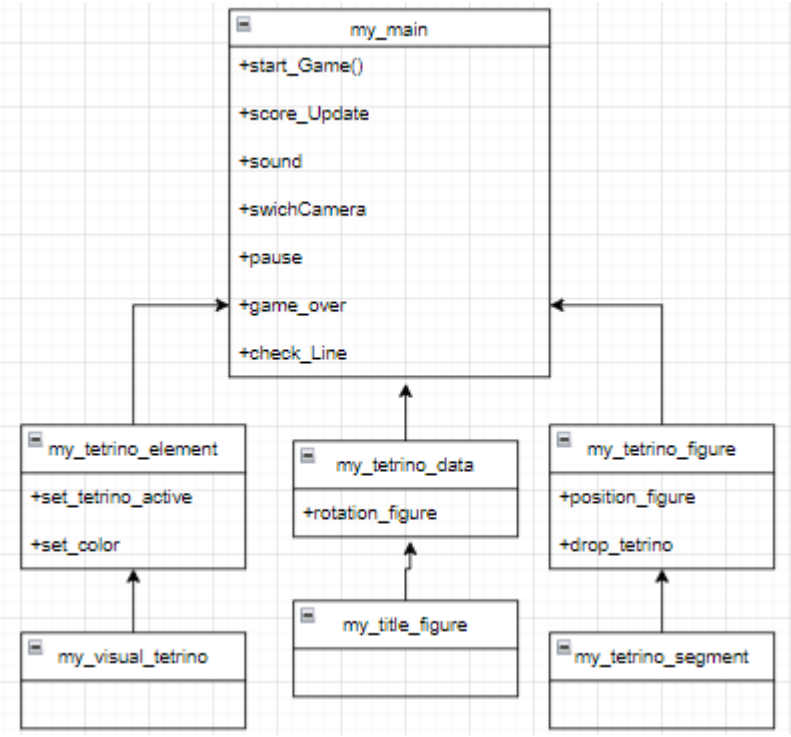

Рисунок 2.3 – Діаграма класів для тетрісу

Діаграма прецедентів, або **Use Case Diagram**, відображає взаємодію між основними учасниками (акторами) та функціоналом системи. Вона слугує для опису функціональних вимог системи шляхом ідентифікації різних сценаріїв використання [16]. Ці сценарії відображають способи, які учасники системи можуть взаємодіяти з нею, виходячи з їх потреб та цілей. Такий підхід дозволяє зв’язати вимоги до системи з їхніми реалізаціями та забезпечити повноту та правильність функціональності системи з урахуванням потреб користувачів.

Для програми було обрано перелік акторів та прецедентів наступним чином:

– **Player** (гравець) актор, представлений як користувач системи;
– **Enter to menu** (повернення до меню) сценарій переходу до головного меню під час початку або завершення гри;

– **Play game** (грати гру) сценарій початку гри;
– **Level Up** (підвищення рівня) показник ступеню прогресу в грі;
– **Game over** (кінець гри) поразка на конкретному рівні;
– **Back to menu** (повернення до меню) головне меню після завершення гри;
– **Remove line** (видалення ліній) ключова механіка гри для продовження процесу гри (див. рис. 2.4).

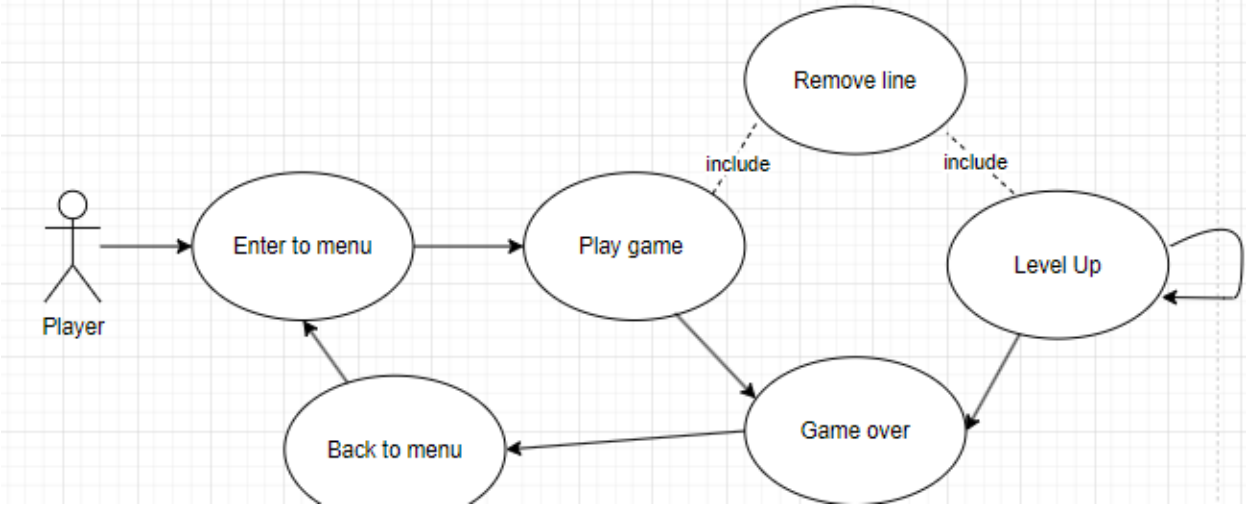

Рисунок 2.4 – Діаграма прецедентів для тетрісу

Діаграма машини станів математична модель, яка включає обмежений набір станів, що переходять один в одного залежно від вхідних даних, керуючи таким чином системою [17]. Ця концепція використовується в поведінковому патерні проєктування "Стан". Згідно з патерном, є один інтерфейс з одним або кількома методами, які викликає цільовий об’єкт, і кілька класів-нащадків, які реалізують ці методи по-різному. Це дозволяє змінювати поведінку об’єкта, якби змінився сам об’єкт. Цільовий об’єкт також має метод для зміни стану, що дозволяє здійснювати переходи між станами.

Користуючись визначенням вижче будуть реалізовані наступні стани

– **Creating** стан створення, використовується для додавання нових елементів;
– **Dropping** стан спуск, використовується лише у стані створення;

– **Cleaning** стан видалення, використовується для переходу від фіксації фігури до очищення поля від повних ліній;
– **Moving** стан пересування, використовується одночасно зі спуском та дозволяє змінити положення фігури;
– **Fixing** стан фіксації фігури, є наступним після перевірки зіткнення використовується для повної фіксації фігури під час цього стану фигурою неможливо рухати;
– **CollisionCheck** стан перевірки зіткнення, потрібен для коректної постанови фігури у просторі (див. рис. 2.5).

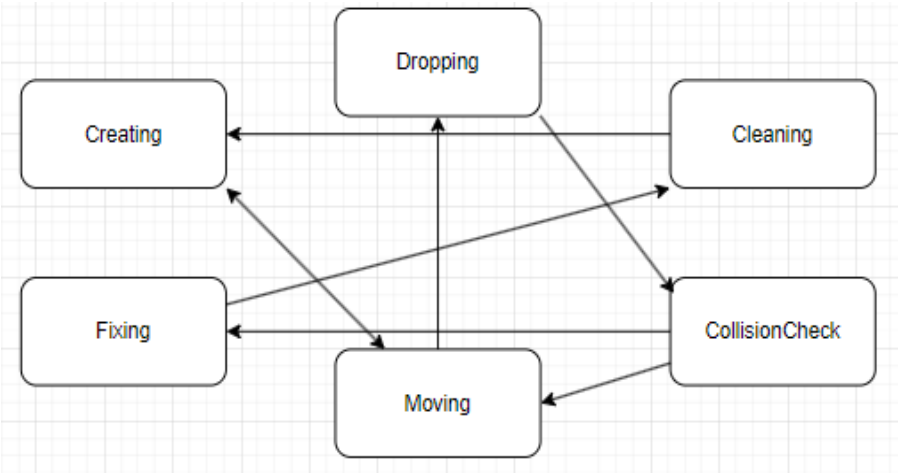

Рисунок 2.5 – Діаграма Машини Станів

2.2.3 Вибір звуків та музики

Правильний вибір музики дуже важливий для створення належної атмосфери в будь-якій грі. Наприклад, для гри в карти не підійде стиль важкої музики, такої як рок. Натомість для карткових ігор краще використовувати спокійну, ненав’язливо музику, яка не відволікає гравців і дозволяє зосередитися на процесі гри. Правильно підібрані звуки в грі дозволяють користувачеві відчути повне занурення в ігровий процес. Звукові ефекти і музика мають пряму кореляцію з емоційним станом гравця, допомагаючи йому краще оцінювати ситуацію на екрані і відчувати вплив своїх рішень та

дій. Наприклад, у Тетрисі, коли гравець повністю заповнює лінію, відтворюється звук і анімація видалення ліній. Це не тільки інформує гравця про успішне виконання дії, але й підсилює задоволення від гри, додаючи елемент досягнення. Крім того, звукові ефекти можуть сигналізувати про різні події в грі.

У тетрисі звук може супроводжувати кожне переміщення та обертання фігури, що допомагає гравцеві краще орієнтуватися в ігровому процесі. Звук фіксації фігури на місці чітко вказує, що хід завершено, і гравець може планувати наступний крок.

Іншим важливим аспектом є використання музичного супроводу під час прискорення гри. Коли рівень складності підвищується, і фігури починають падати швидше, музика може змінюватися, стаючи більш динамічною і напруженою. Це допомагає гравцеві відчути зростаючу інтенсивність гри і стимулює його швидше приймати рішення. Таким чином, звуковий супровід і музика у грі виконують кілька важливих функцій.Створення атмосфери: Правильно підібрана музика допомагає зануритися в гру і відчути її настрій.

Інформаційна функція: звукові ефекти сигналізують про важливі події та дії в грі, допомагаючи гравцеві орієнтуватися. Музика і звуки впливають на емоційний стан гравця, підсилюючи задоволення від гри і мотивуючи до подальших дій.

Підвищення залученості: інтерактивні звукові ефекти і музика роблять гру більш захоплюючою і цікавою. Усі ці аспекти важливі для створення які якісного ігрового досвіду, який буде подобатися гравцям і тримати їх у грі протягом тривалого часу.

2.2.4 Концепт фігур

Фігури в Тетрісі складаються з чотирьох квадратних клітинок, які з’єднані таким чином, що кожен квадрат має спільну сторону з іншим. Таких

фігур всього сім, і вони мають форми, які нагадують певні літери. Ми назвемо їх “T”, “Q”, “I”, “Z”, “S”, “J” та “L”.

Наведемо види фігур.

“T”-фігура: фігура схожа на літеру “T”. Вона складається з трьох клітинок, що утворюють горизонтальну лінію, і однієї клітинки, яка виступає знизу по центру. Вона дуже гнучка і дозволяє добре заповнювати простір.

“Q”-фігура: це квадрат 2x2, іноді її називають «блоком». Вона єдина симетрична фігура, яка не потребує обертання. Зручно використовувати для заповнення рівних поверхонь.

“I”-фігура: ця фігура виглядає як пряма лінія із чотирьох клітинок. Вона може бути як горизонтальною, так і вертикальною. Це ідеальна фігура для створення «Тетрісів» – одночасного заповнення чотирьох ліній.

“Z”-фігура: вона схожа на літеру “Z” і складається з двох горизонтальних рядів по дві клітинки, зміщених один відносно одного. Добре заповнює специфічні проміжки, але потребує обережного планування.

“S”-фігура: дзеркально відображає “Z”-фігуру. Вона також складається із двох горизонтальних рядів по дві клітинки, зміщених відносно один одного. Дуже корисна для заповнення проміжків, особливо у поєднанні з іншими фігурами.

«J» фігура: фігура має три клітинки у вертикальній лінії і одну клітинку, яка виступає ліворуч знизу. Вона нагадує літеру “J”. Вона гнучка і дозволяє створювати різні комбінації для заповнення простору.

“L” фігура: схожа на “J”-фігуру, але зеркально відображена. Складається з трьох клітинок у вертикальній лінії і однієї клітинки, що виступає справа знизу. Вона нагадує літеру “L”. Фігура має подібні властивості до «J» фігури і також добре заповнює простір. Кожна з цих фігур має свої унікальні властивості та способи використання, що робить гру в Тетріс захоплюючою та стратегічно цікавою. Вміння правильно використовувати ці фігури допомагає гравцеві ефективно заповнювати лінії та досягати високих результатів.

2.2.5 Анімація програшу

Для екрану програшу гравця буде створено анімацію, яка демонструє прокручування наявних на полі фігур. Ця анімація додасть динамічності і зробить завершення гри більш ефективним та цікавим. При визначенні, що гра завершена через неможливість розміщення нової фігури, відбувається виклик функції анімації. Зберігаються всі поточні фігури та їхнє розташування на ігровому полі, зупиняється будь-який інший ігровий процес, щоб гравець міг повністю зосередитися на анімації. Фігури починають плавно рухатися навколо своєї ос. Коли фігури досягають своєї початкової координати анімація починається знову створюючи ефект безперервного руху.

Прокручування фігур може починатися повільно, поступово прискорюючись, щоб підсилити емоційний ефект. Можна додати випадкові затримки для окремих фігур, щоб зробити анімацію більш цікавою і менш передбачуваною. Додаються ефекти зникнення та появи фігур, наприклад, зміна прозорості або кольору. Можливе використання інших графічних ефектів, таких як тіні чи сяйво, щоб підкреслити рух. Після декількох циклів прокручування, анімація плавно зупиняється. Гравець може вибрати перезапуск гри натисканням відповідної кнопки поруч з ігровим полем. Кнопка перезапуску стає доступною після та під час анімації. Динамічна анімація додає емоційності до моменту програшу, роблячи його менш рутинним і більш захоплюючим.

2.3 Висновки до розділу 2

В даному розділі було детально розглянуто основні аспекти розробки гри Тетріс, включаючи її концептуальну основу, технічну реалізацію, проєктування інтерфейсу, логіку гри, вибір звукового та музичного супроводу, а також анімацію програшу.

Розробка проєкту в середовищі Unity з використанням мови програмування C# забезпечує ефективну реалізацію всіх необхідних функцій, включаючи генерацію випадкових тетраміно, обробку введення користувача, перевірку зіткнень та очищення заповнених ліній. Особлива увага була приділена створенню зручного та інформативного інтерфейсу, який допомагає гравцеві відстежувати свій прогрес та приймати рішення.

Правильний вибір звуків та музики є ключовим для створення захоплюючої атмосфери гри. Звукові ефекти допомагають гравцеві орієнтуватися у грі та отримувати зворотній зв’язок про свої дії, а музика підсилює емоційний вплив гри. Анімація програшу додає динамічності та робить завершення гри менш рутинним, а також стимулює гравця на нові спроби.

Таким чином, всі ці елементи створюють цілісний, якісний та захоплюючий ігровий досвід, який може утримувати увагу гравця та мотивувати його до подальших ігор. Розроблений проєкт Тетріс відповідає сучасним стандартам та вимогам, забезпечуючи комфортну та цікаву гру для користувачів.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ГРИ

3.1 Створення сцени

При створенні сцени в Unity важливо вибрати правильну платформу розробки, яка визначить, на яких пристроях можна використовувати гру. Unity підтримує кілька платформ, включаючи ПК, мобільні пристрої, консолі, VR/AR і вебплатформу. Вибір залежить від цільової аудиторії та типу гри, яка буде розроблена (див. рис. 3.1).

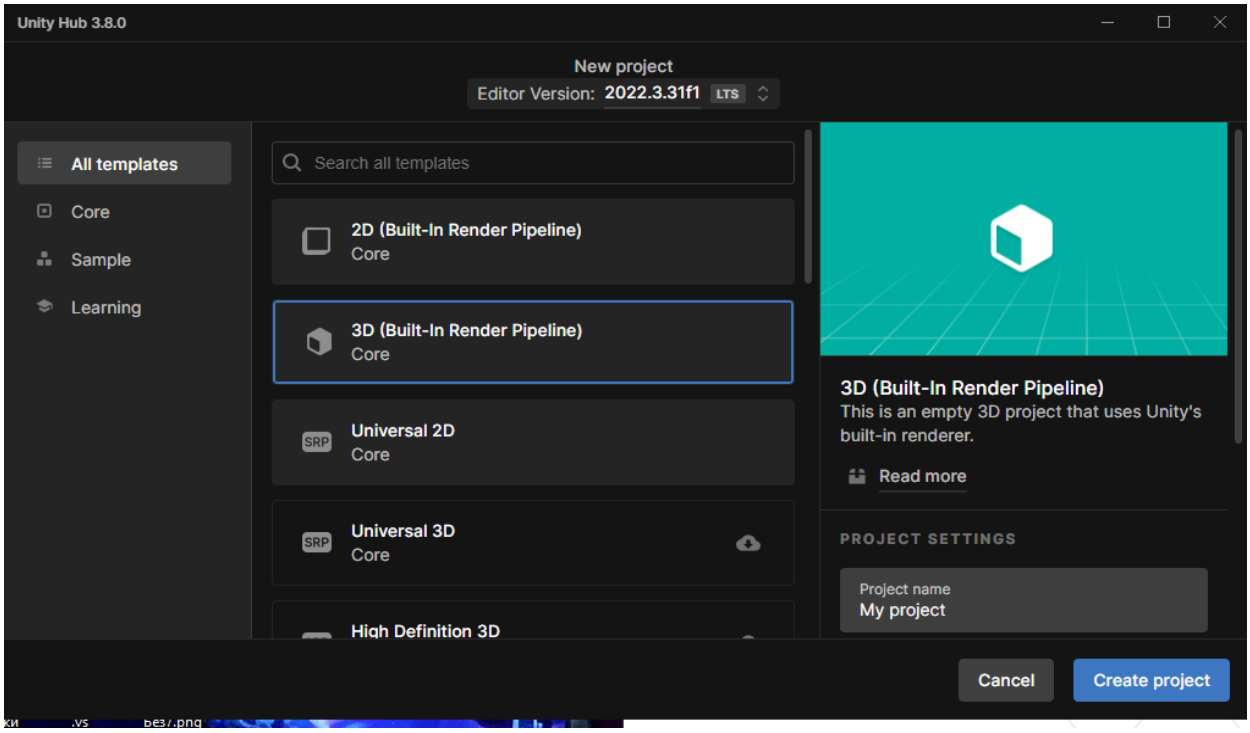

Рисунок 3.1 – Створення проєкту

3.1.1 Базові налаштування

У проєкті я створив папку з міткою «my» для кожного елемента, щоб забезпечити структуровану організацію ресурсів. Ці папки містять різні типи файлів, такі як скрипти, звуки, збірки, матеріали та елементи анімації. Усі ці

елементи знаходяться в папці Resources, що спрощує доступ до них під час виконання програми. Наприклад, шлях до основного скрипта my_main записується як Assets/Resources/my_script/my_main. Коли проєкт запускається, спочатку відбувається пошук і доступ до папки Resources, щоб завантажити необхідні файли. Найважливішим елементом проєкту є скрипти, оскільки без них проєкт не зможе функціонувати.

Основна логіка гри реалізована через скрипти, які пов’язуються з об’єктами сцени та керують їхньою поведінкою. Скрипти в проєкті поділяються на загальні та приватні.

До загальних скриптів входить my_main, який відповідає за створення ігрового поля, поведінку камери, реалізацію функцій ускладнення гри, управління списками, обертання фігур і паузу. Уся інформація з інших файлів зосереджується в main, забезпечуючи централізоване управління основними аспектами гри. Приватні скрипти, такі як tetrino_data, впливають лише на конкретні елементи, зокрема на фігури тетроміно. Цей скрипт керує генерацією тетроміно, їхнім рухом, обертанням і взаємодією з іншими елементами гри. Такий поділ на загальні та приватні скрипти допомагає забезпечити модульність і спрощує підтримку проєкту (див. рис. 3.2).

Рисунок 3.2 – Повний список скриптів для роботи гри тетрис

Така організація проєкту не лише полегшує навігацію та управління ресурсами, але й забезпечує високу ефективність розробки, зменшує ризик помилок та покращує читабельність коду. Це сприяє більш продуктивному процесу розробки та підтримки проєкту, забезпечуючи його стабільну роботу та легке впровадження нових функцій.

3.1.2 Створення префабів

У Unity префаби допомагають створювати та керувати ігровими об’єктами, роблячи процес розробки зручним та ефективним. У Тетрісі це особливо корисно для створення фігур, що складаються з блоків префабів рисунок 3.3.

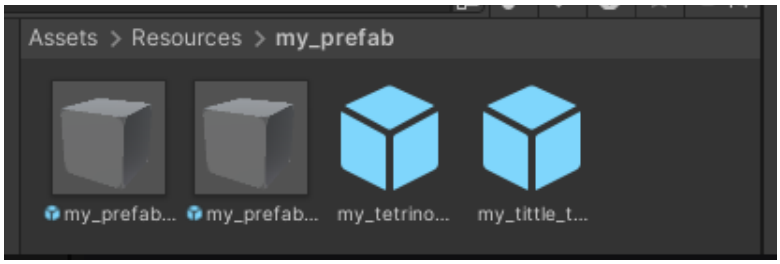

Рисунок 3.3 – Створені префаби

Наприклад, ви можете створити одну фігуру з кількох блоків та зберегти її як префаб. Потім цей префаб можна використовувати багаторазово, додаючи в гру нові екземпляри фігури без необхідності створювати їх з нуля щоразу.

Якщо потрібно змінити властивості фігури, достатньо змінити префаб і всі його екземпляри в грі оновлюються автоматично. Це спрощує створення нових фігур, управління ними та робить гру більш гнучкою та легкою для розробки.

3.2 Створення UI

Створення UI в Unity починається з додавання Canvas, яке є основним контейнером для всіх елементів інтерфейсу. Canvas автоматично підлаштовується під розміри екрана, забезпечуючи правильне відображення інтерфейсу на різних пристроях та дозволяючи створювати адаптивний дизайн.

У Canvas можна розміщувати різні UI-елементи рисунок 3.4, такі як кнопки, текстові поля, зображення та панелі. Наприклад, кнопки використовуються для взаємодії з користувачем, текстові поля – для відображення інформації або введення даних, а зображення та панелі можуть використовуватися для створення більш складних візуальних компонентів та розділів інтерфейсу.

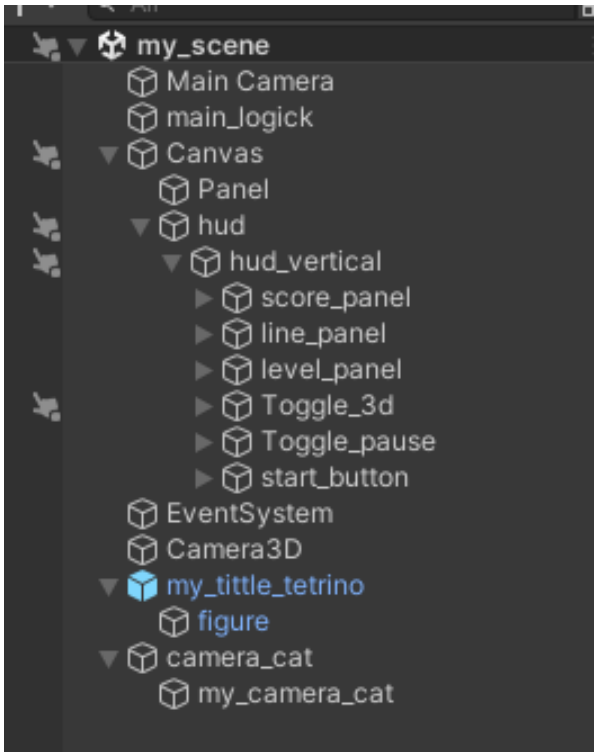

Рисунок 3.4 – Елементи UI

Для додавання цих елементів можна використовувати контекстне меню, викликане Canvas.

Кожен елемент інтерфейсу можна налаштувати інспекторі, змінюючи його властивості.

Наприклад, можна змінити текст, колір, розмір, шрифт та позицію елемента. Це дозволяє створювати індивідуальний та привабливий дизайн інтерфейсу, що відповідає стилю гри або додатку.

Щоб UI-елементи реагували на дії користувача, можна додавати до них скрипти, що виконують певні функції при взаємодії.

3.2.1 Додавання звукових ефектів

В проєкті, де присутні звуки при повному заповненні ліній, реалізована можливість відтворення одного з дев’яти випадкових звуків на рисунок 3.5. Це додає грі динамічності та різноманітності.

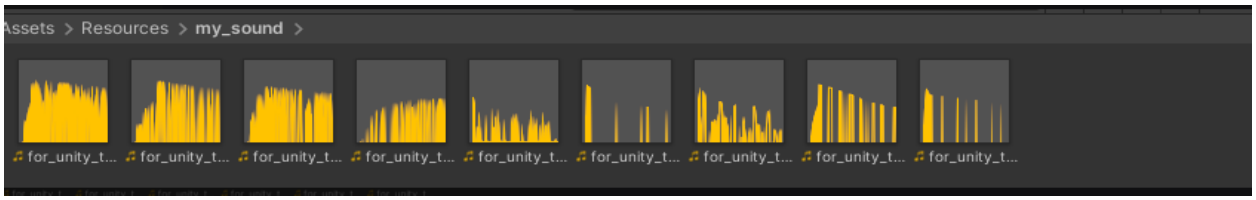

Рисунок 3.5 – Список звуків

Для працездатності та відтворення випадкових звуків було написано наступний код (див. рис. 3.6).

```
private AudioSource my_audio_sound;
   private AudioClip[] my_sounds;

   private void Start()
   {
      …
       my_sounds = Resources.LoadAll<AudioClip>("my_sound");
       my_audio_sound = GetComponent<AudioSource>();
       MyRandomSound();
…
           }
   }

private void MyRandomSound()
   {
       int index = UnityEngine.Random.Range(0, my_sounds.Length);
       my_audio_sound.clip = my_sounds[index];
   }
private void MyRemoveFullLine()
   {
     …
       {
          MyRandomSound();
          my_audio_sound.Play();
…
       }
   }
```

Рисунок 3.6 – Відтворення випадкових звуків при видалені лінії

Реалізація відтворення звуків здійснюється через об’єкт main_logick, до якого прив’язаний компонент AudioSource.

При повному заповненні лінії у грі, логіка відтворення звуку працює наступним чином.

Спочатку відбувається визначення випадкового числа, яке відповідає одному з восьми доступних звукових ефектів. Це число використовується для вибору звукового файлу зі списку заздалегідь підготовлених відеокліпів.

Компонент AudioSource, прив’язаний до об’єкта main_logick, відповідає за відтворення звуку. Він налаштовується таким чином, щоб відтворити вибраний звуковий ефект. Це дозволяє зберегти всі звукові налаштування в одному місці, спрощуючи керування звуками в проєкті.

3.3 Ігрові анімації

Щоб створити анімацію в Unity, створюються ключові кадри (keyframe) рисунок 3.7, які визначають положення, поворот, розмір та інші параметри об’єкта в різні моменти часу, це відбувається за рахунок змін координат простору.

Потім система анімації створює плавний перехід між цими ключовими кадрами та показує анімацію.

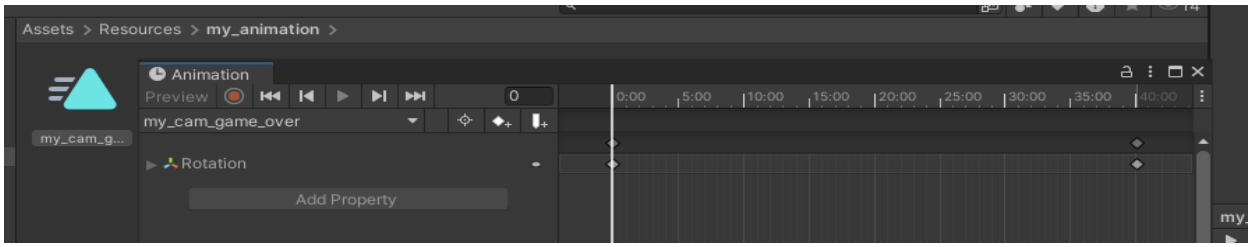

Рисунок 3.7 – Створення ключів

Тетріс має анімацію, яка запускається після програшу гравця. Ця анімація відображає плавне обертання об’єктів і є нескінченною, тобто

відтворюється циклічно (див. рис. 3.8). Для підключення цієї анімації в грі була створена друга камера під назвою «camera_cat», до якої додано файл анімації.

```
private GameObject my_animation_camera;

private void Start()
   {
  …
           my_animation_camera = GameObject.FindGameObjectWithTag("my_camera_animation");
       my_animation_camera.SetActive(false);
      …
          }
   }
…
 private IEnumerator my_update(float _time)
   {
 …
       if(!IsGameOver())
       {
       …
       }
       else
       {
          my_main_camera.SetActive(false);
          my_3d_camera.SetActive(false);
          my_animation_camera.SetActive(true);

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

Рисунок 3.8 – Анімація GameOver

3.4 Зміна положення фігури

Після створення префабу майбутньої фігури до нього потрібно підключити скрип my_tetrino_data для взаємодії та зміни фігури, не впливаючи на глобальний світ. Також в цьому файлі буде зберігатися список можливих фігур і реалізація повороту за допомогою векторних кардинат, був використаний Vector 3 (див. рис. 3.9). Повний код – у додатку А.1.

```
private void MyRotationType(TetrinoFigure _figure, int _rot)
   {
      switch (_rot)
      {
         case 0:
            if(_figure == TetrinoFigure.L)
            {
                  my_tetrino_array[0].transform.localPosition = new Vector3(0, 0, 0);
                  my_tetrino_array[1].transform.localPosition = new Vector3(0, 1, 0);
                  my_tetrino_array[2].transform.localPosition = new Vector3(0, -1, 0);
                  my_tetrino_array[3].transform.localPosition = new Vector3(1, -1, 0);
            }
            else if(_figure == TetrinoFigure.T)
            {
                  my_tetrino_array[0].transform.localPosition = new Vector3(0, 0, 0);
                  my_tetrino_array[1].transform.localPosition = new Vector3(1, 0, 0);
                  my_tetrino_array[2].transform.localPosition = new Vector3(-1, 0, 0);
                  my_tetrino_array[3].transform.localPosition = new Vector3(0, 1, 0);
            }
            else if(_figure == TetrinoFigure.I)
            {
                  my_tetrino_array[0].transform.localPosition = new Vector3(0, 0, 0);
                  my_tetrino_array[1].transform.localPosition = new Vector3(0, -1, 0);
                  my_tetrino_array[2].transform.localPosition = new Vector3(0, 1, 0);
                  my_tetrino_array[3].transform.localPosition = new Vector3(0, -2, 0);
            }
            else if(_figure == TetrinoFigure.Z)
            {
                  my_tetrino_array[0].transform.localPosition = new Vector3(0, 0, 0);
                  my_tetrino_array[1].transform.localPosition = new Vector3(0, 1, 0);
                  my_tetrino_array[2].transform.localPosition = new Vector3(-1, 1, 0);
                  my_tetrino_array[3].transform.localPosition = new Vector3(1, 0, 0);
            }
            else if(_figure == TetrinoFigure.O)
            {
                  my_tetrino_array[0].transform.localPosition = new Vector3(0, 0, 0);
                  my_tetrino_array[1].transform.localPosition = new Vector3(-1, 0, 0);
                  my_tetrino_array[2].transform.localPosition = new Vector3(-1, -1, 0);
                  my_tetrino_array[3].transform.localPosition = new Vector3(0, -1, 0);
            }  break;
…
```

Рисунок 3.9 – Реалізація повороту фігури

3.5 Реалізація паузи та 3D виду

Для реалізації паузи і 3Д камери необхідно підключити ігрову логіку, робиться наступним чином. Вибираємо кнопку паузи, додаємо для неї тег і

посилаємося на файл з основною логікою my_mine (див. рис. 3.10), аналогічні дії робимо і для 3D камери(див. рис. 3.11).

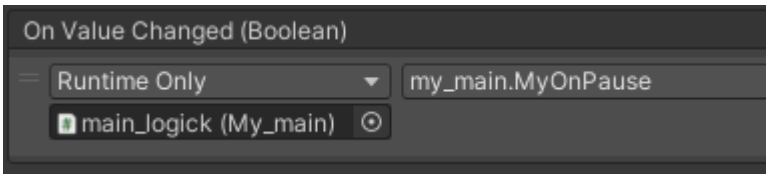

Рисунок 3.10 – Налаштування кнопки

```
 private GameObject my_3d_camera;
   private GameObject my_main_camera;
…
 private void Start()
   {
     ..
       my_main_camera = GameObject.FindGameObjectWithTag("MainCamera");
       my_3d_camera = GameObject.FindGameObjectWithTag("my_3d_camera");
    …
       MyOnSwitchCamera(false);
…
           }
   }
…
public void MyOnSwitchCamera(bool _is3D) {
       my_main_camera.SetActive(!_is3D);
       my_3d_camera.SetActive(_is3D);  }
```

Рисунок 3.11 – Камера 3D виду

Для запобігання паузи при старті рівня прибираємо галочки в пункті “is On” у коді для роботи паузи (див. рис. 3.12).

```
public void MyOnPause(bool _isPause)  {
       Time.timeScale = _isPause ? 0 : 1; }
```

Рисунок 3.12 – Умови паузи

3.6 Керування фігурою

Керування в Tetris є критично важливим елементом гри, оскільки дозволяє гравцеві активно взаємодіяти з ігровим процесом і досягати успіху.

Елементи управління дозволяють гравцеві переміщати, обертати і прискорювати падіння фігур (тетроміно), що необхідно для їх правильного розташування на ігровому полі. Це дає можливість створювати лінії, які будуть видалятися, звільняючи простір і наближаючи окуляри. Можливість керувати фігурками дозволяє гравцеві застосовувати стратегію і планувати свої дії. Гравець може розставляти фігури таким чином, щоб уникнути пропусків і створити умови для видалення кількох ліній одночасно, що приносить більше очок. Зі збільшенням складності гри фігурки починають падати швидше. Управління дозволяє гравцеві адаптуватися до збільшення швидкості, вчасно реагувати на зміну ситуації та уникати ситуацій, коли фігури накопичуються до верхньої межі екрана, що призводить до програшу. Кожна фігурка в Tetris має унікальну форму, а управління дозволяє оптимально використовувати їх можливості. Наприклад, довга палиця може бути особливо корисною для тетрісу (очистити чотири лінії одночасно), тоді як маленькі кубики можуть заповнити невеликі прогалини. Добре продумане управління робить гру більш захоплюючою та захоплюючою. Коли гравець відчуває, що він повністю контролює процес, це підвищує його інтерес і бажання продовжувати гру. Таким чином, управління в Tetris є фундаментальним компонентом, який забезпечує динаміку гри, розвиває стратегічне мислення і робить ігровий процес захоплюючим і захоплюючим (див. рис. 3.13).

```
private void Update()
   {
   if(my_figure)
    {
      if(Input.GetButtonDown("RotateTetrino"))
      {
         my_figure.GetComponentInChildren<my_tetrino_data>().MyRotation(true);
         if(CheckIntersect(my_figure))
             my_figure.GetComponentInChildren<my_tetrino_data>().MyRotation(false);
      }

      if(Input.GetButtonDown("LeftTetrino"))
      {
         my_curr_time = 0;
         my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
         if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
```

```
    }

    else if(Input.GetButtonDown("RightTetrino"))
    {
        my_curr_time = 0;
        my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
        if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
    }

    if(Input.GetButton("DownTetrino"))
        MyInputPress(MyDirecrionTetrino.DOWN, 0.04f);
    if (Input.GetButton("RightTetrino"))
        MyInputPress(MyDirecrionTetrino.RIGHT, 0.08f);
    else if(Input.GetButton("LeftTetrino"))
        MyInputPress(MyDirecrionTetrino.LEFT, 0.08f);

    if(Input.GetButtonUp("LeftTetrino") || Input.GetButtonUp("RightTetrino"))
        my_curr_time = 0;
 }
}

private void MyInputPress(MyDirecrionTetrino _dir, float _time)
{
    my_curr_time += Time.deltaTime;
    if(my_curr_time > _time)
    {
        my_curr_time = 0;

        if(_dir == MyDirecrionTetrino.LEFT)
     {
        my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
        if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
     }
    else if(_dir == MyDirecrionTetrino.RIGHT)
     {
        my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
        if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
     }
     else if(_dir == MyDirecrionTetrino.DOWN)
     {
        my_figure.MyDropTetrino(true);
        if(CheckIntersect(my_figure))
            my_figure.MyDropTetrino(false);
     }
    }
}

 private bool CheckIntersect(my_tetrino_figure _figure)
{
    for (int ind = 0; ind < _figure.GetSegments().Length; ind++)
    {
        int x = (int)_figure.GetSegments()[ind].transform.position.x;
        int y = (int)_figure.GetSegments()[ind].transform.position.y;

        bool is_intersect = IsIntersect(x, y);

        if(is_intersect)
            return is_intersect;
    }

    return false;
}
```

Рисунок 3.13 – Управління фігурою

ВИСНОВКИ

Створення будь-якої гри починається з основної ігрової сцени, в якій буде реалізовано положення гравця та його мета з додатковим функціоналом

На кожному етапі гри важливо пам’ятати про структурність і порядок для більш продуктивної та легкої реалізації проєкту. При створенні важливо подбати про зручність гравця це інтерфейс і управління.

Також вплив надає фактор реграбельності та презентабельність реалізованих механік, адже без таких пунктів гравець швидше за все довго затримуватиметься у грі не буде. Музика та звукові супроводи являються основою для будь-якої гри, правильно підібраний підібраний звуковий супровід багаторазово посилює занурення і бажання продовжувати грати, не варто забувати і про оптимізацію зробивши гарну гру з правильно реалізованим етапами в ній буде мало сенсу якщо в неї ніхто не зможе.

Загалом створення гри не така вже й проста затія, адже дотримання стільких параметрів забирає дуже багато часу і сил. При виборі цього шляху розробник має бути готовий до труднощів та критики.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity. URL: https://www.pubnub.com/guides/unity/ (дата звернення: 18.04.2024).
2. Виконують одразу кілька ролей у командах. Результати опитування українських незалежних розробників. URL: https://gamedev.dou.ua/articles/ukrainian-indie-gamedev-2022/ (дата звернення: 18.04.2024).
3. Документація Godot. URL: https://docs.godotengine.org/en/stable/ (дата звернення: 18.04.2024).
4. Plato and the genesis of computer learning. URL: https://grainger.illinois.edu/news/magazine/plato (дата звернення: 20.04.2024).
5. History of Arcade Games URL: https://www.betson.com/the-history-of-arcade-games/ (дата звернення: 20.04.2024).
6. Tetris. URL: https://www.britannica.com/topic/Tetris (дата звернення: 22.04.2024).
7. Tennis for Two The story of an early computer game by John Anderson. URL: https://www.pong-story.com/1958.htm (дата звернення: 22.04.2024).
8. RPGMaker. URL: https://rpgmakerofficial.com/ (дата звернення: 23.04.2024).
9. 3D-моделювання для ігор. URL: https://klona.ua/uk/blog/3d-modeling-and-visualization-uk/3d-modelyuvannya-dlya-igor-stvorennya-model (дата звернення: 23.04.2024).
10. GameMaker: Studio. URL: https://gamemaker.io/ (дата звернення: 24.04.2024).
11. A History Timeline About C#. URL: https://historytimelines.co/timeline/c (дата звернення: 24.04.2024).
12. A Detailed History Of C++ Explained With Timeline Infographic. URL: https://unstop.com/blog/history-of-cpp (дата звернення: 25.04.2024).

13. Види ігор. URL: https://games-yes-no.webnode.com.ua/vidi-igor/ (дата звернення: 23.04.2024).
14. Еволюція графіки в іграх: від пікселів і спрайтів до більш складних моделей. URL: https://versum.ua/jevoljucija-grafiki-v-igrah (дата звернення: 22.04.2024).
15. What Is A Use Case Diagram In UML? URL: https://online.visual-paradigm.com/diagrams/tutorials/use-case-diagram-tutorial/ (дата звернення: 23.04.24).
16. Що таке діаграма класів UML і найкращий творець діаграм класів UML. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (дата звернення: 23.04.24).
17. Діаграма станів (statechart diagram). URL: https://studfile.net/preview/9828818/ (дата звернення: 23.04.24).

ДОДАТОК А

Лістинг коду

А.1 Лістинг скрипту my_tetrino_data

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

public enum TetrinoFigure { L, Z, I, O, T }

public class my_tetrino_data : MonoBehaviour
{
    private GameObject pref_cube;

    private GameObject[] my_tetrino_array;

    private int my_rotation;

    private TetrinoFigure my_type;

    public Color my_color { get; private set; }

    private void Awake()
    {
        my_rotation = 0;
        my_tetrino_array = new GameObject[4];


        pref_cube = Resources.Load("my_prefab/my_prefab_cube") as GameObject;
    }

    public void MySetColor(Color _col)
    {
        for (int ind = 0; ind < transform.childCount; ind++)
        {
            GameObject go = transform.GetChild(ind).gameObject;
            Material mat = go.GetComponent<MeshRenderer>().material;
            mat.color = _col;
```

```
            my_color = _col;
        }
    }

    public GameObject[] GetTetrinoArray { get { return my_tetrino_array; } }

    public void MyRotation(bool _isPositive)
    {
        if(_isPositive)
        {
            my_rotation++;
            my_rotation = my_rotation % 4;
        }
        else
        {
            my_rotation--;
            if (my_rotation < 0)
                my_rotation = 3;
        }


        MyRotationType(my_type, my_rotation);
    }

    private void MyRotationType(TetrinoFigure _figure, int _rot)
    {
        switch (_rot)
        {
            case 0:
                if(_figure == TetrinoFigure.L)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(1, -1, 0);
                }
                else if(_figure == TetrinoFigure.T)
                {
```

```
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, 1, 0);
                }
                else if(_figure == TetrinoFigure.I)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -2, 0);
                }
                else if(_figure == TetrinoFigure.Z)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(1, 0, 0);
                }
                else if(_figure == TetrinoFigure.O)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -1, 0);
                }
                break;
```

```
            case 1:
                if(_figure == TetrinoFigure.L)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(1, 1, 0);
                }
                else if(_figure == TetrinoFigure.T)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(-1, 0, 0);
                }
                else if(_figure == TetrinoFigure.I)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(2, 0, 0);
                }
                else if(_figure == TetrinoFigure.Z)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
```

```
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, 1, 0);
                }
                else if(_figure == TetrinoFigure.O)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -1, 0);
                }
                break;
            case 2:
                if(_figure == TetrinoFigure.L)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(-1, 1, 0);
                }
                else if(_figure == TetrinoFigure.T)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -1, 0);
                }
                else if(_figure == TetrinoFigure.I)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
```

```
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -2, 0);
                }
                else if(_figure == TetrinoFigure.Z)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(1, 0, 0);
                }
                else if(_figure == TetrinoFigure.O)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -1, 0);
                }
                break;
            case 3:
                if(_figure == TetrinoFigure.L)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(-1, -1, 0);
                }
                else if(_figure == TetrinoFigure.T)
```

```
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(0, 1, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(0, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(1, 0, 0);
                }
                else if(_figure == TetrinoFigure.I)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(2, 0, 0);
                }
                else if(_figure == TetrinoFigure.Z)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, 1, 0);
                }
                else if(_figure == TetrinoFigure.O)
                {
                        my_tetrino_array[0].transform.localPosition = new
Vector3(0, 0, 0);
                        my_tetrino_array[1].transform.localPosition = new
Vector3(-1, 0, 0);
                        my_tetrino_array[2].transform.localPosition = new
Vector3(-1, -1, 0);
                        my_tetrino_array[3].transform.localPosition = new
Vector3(0, -1, 0);
                }
```

```
                break;

            default:
                break;
        }
    }

    public void MyInitialize(TetrinoFigure _myType)
    {
        for(int ind = 0; ind < transform.childCount; ind++)
            Destroy(transform.GetChild(ind).gameObject);

          switch (_myType)
          {
            case TetrinoFigure.L:
                my_type = _myType;
                GameObject obL = Instantiate(pref_cube, new Vector3(),
Quaternion.identity);
                obL.AddComponent<my_tetrino_segment>();
                obL.transform.SetParent(transform, false);


                GameObject obL2 = Instantiate(pref_cube, new Vector3(0, 1,
0), Quaternion.identity);
                obL2.AddComponent<my_tetrino_segment>();
                obL2.transform.SetParent(transform, false);


                GameObject obL3 = Instantiate(pref_cube, new Vector3(0, -1,
0), Quaternion.identity);
                obL3.AddComponent<my_tetrino_segment>();
                obL3.transform.SetParent(transform, false);


                GameObject obL4 = Instantiate(pref_cube, new Vector3(1, -1,
0), Quaternion.identity);
                obL4.AddComponent<my_tetrino_segment>();
                obL4.transform.SetParent(transform, false);


                for(int ind = 0; ind < my_tetrino_array.Length; ind++)
                    my_tetrino_array[ind] =
transform.GetChild(ind).gameObject;
```

```
                break;
            case TetrinoFigure.Z:
                my_type = _myType;
                GameObject obZ = Instantiate(pref_cube, new Vector3(),
Quaternion.identity);
                obZ.AddComponent<my_tetrino_segment>();
                obZ.transform.SetParent(transform, false);


                GameObject obZ2 = Instantiate(pref_cube, new Vector3(0, 1,
0), Quaternion.identity);
                obZ2.AddComponent<my_tetrino_segment>();
                obZ2.transform.SetParent(transform, false);


                GameObject obZ3 = Instantiate(pref_cube, new Vector3(-1, 1,
0), Quaternion.identity);
                obZ3.AddComponent<my_tetrino_segment>();
                obZ3.transform.SetParent(transform, false);


                GameObject obZ4 = Instantiate(pref_cube, new Vector3(1, 0,
0), Quaternion.identity);
                obZ4.AddComponent<my_tetrino_segment>();
                obZ4.transform.SetParent(transform, false);


                for(int ind = 0; ind < my_tetrino_array.Length; ind++)
                    my_tetrino_array[ind] =
transform.GetChild(ind).gameObject;
                break;
            case TetrinoFigure.I:
                my_type = _myType;
                GameObject obI = Instantiate(pref_cube, new Vector3(),
Quaternion.identity);
                obI.AddComponent<my_tetrino_segment>();
                obI.transform.SetParent(transform, false);

                GameObject obLI = Instantiate(pref_cube, new Vector3(0, -1,
0), Quaternion.identity);
                obLI.AddComponent<my_tetrino_segment>();
                obLI.transform.SetParent(transform, false);
```

```
                GameObject obI3 = Instantiate(pref_cube, new Vector3(0, 1,
0), Quaternion.identity);
                obI3.AddComponent<my_tetrino_segment>();
                obI3.transform.SetParent(transform, false);

                GameObject obI4 = Instantiate(pref_cube, new Vector3(0, 2,
0), Quaternion.identity);
                obI4.AddComponent<my_tetrino_segment>();
                obI4.transform.SetParent(transform, false);

                for(int ind = 0; ind < my_tetrino_array.Length; ind++)
                    my_tetrino_array[ind] =
transform.GetChild(ind).gameObject;
                break;
            case TetrinoFigure.O:
                my_type = _myType;
                GameObject obO = Instantiate(pref_cube, new Vector3(),
Quaternion.identity);
                obO.AddComponent<my_tetrino_segment>();
                obO.transform.SetParent(transform, false);


                GameObject obO2 = Instantiate(pref_cube, new Vector3(-1, 0,
0), Quaternion.identity);
                obO2.AddComponent<my_tetrino_segment>();
                obO2.transform.SetParent(transform, false);

                GameObject obO3 = Instantiate(pref_cube, new Vector3(-1, -1,
0), Quaternion.identity);
                obO3.AddComponent<my_tetrino_segment>();
                obO3.transform.SetParent(transform, false);

                GameObject obO4 = Instantiate(pref_cube, new Vector3(0, -1,
0), Quaternion.identity);
                obO4.AddComponent<my_tetrino_segment>();
                obO4.transform.SetParent(transform, false);

                for(int ind = 0; ind < my_tetrino_array.Length; ind++)
                    my_tetrino_array[ind] =
transform.GetChild(ind).gameObject;
                break;
            case TetrinoFigure.T:
                my_type = _myType;
```

```
                GameObject obT = Instantiate(pref_cube, new Vector3(),
Quaternion.identity);
                obT.AddComponent<my_tetrino_segment>();
                obT.transform.SetParent(transform, false);

                GameObject obT2 = Instantiate(pref_cube, new Vector3(1, 0,
0), Quaternion.identity);
                obT2.AddComponent<my_tetrino_segment>();
                obT2.transform.SetParent(transform, false);

                GameObject obT3 = Instantiate(pref_cube, new Vector3(-1, 0,
0), Quaternion.identity);
                obT3.AddComponent<my_tetrino_segment>();
                obT3.transform.SetParent(transform, false);

                GameObject obT4 = Instantiate(pref_cube, new Vector3(0, 1,
0), Quaternion.identity);
                obT4.AddComponent<my_tetrino_segment>();
                obT4.transform.SetParent(transform, false);

                for(int ind = 0; ind < my_tetrino_array.Length; ind++)
                    my_tetrino_array[ind] =
transform.GetChild(ind).gameObject;
                break;
            default:
                break;
          }
    }
}
```

A.2 Лістинг скрипту my_mine

```
using System;
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
```

```
public struct MyHud
{
    public Text txt_score;
    private int my_int_score;

    public Text txt_line;
    private int my_int_line;

    public Text txt_level;
    public int my_int_level { get; private set; }

    public float my_speed;

    private int my_counter_line;

    public void AddScore(int _score)
    {
        my_int_score += _score;
        txt_score.text = my_int_score.ToString();
    }

    public void AddLine(int _line)
    {
        my_int_line += _line;
        txt_line.text = my_int_line.ToString();

        my_counter_line += _line;
        if(my_counter_line > 9)
            AddLevel(1);

        my_counter_line = my_counter_line % 10;
    }

    public void AddLevel(int _level)
    {
        my_int_level += _level;
        txt_level.text = my_int_level.ToString();
        my_speed -= my_int_level > 5 ? 0.02f : 0.05f;
    }
}

public class my_main : MonoBehaviour
{
```

```
    private int wid = 13, hei = 21;
    private float my_step = 1;
    private float my_curr_time;

    private GameObject pref_tetrino;
    private UnityEngine.Object pref_tetrino_object;
    private my_tetrino_figure my_figure;
    private TetrinoFigure my_figure_random;

    private my_tetrino_element[,] my_array;
    private MyHud my_hud;

    private GameObject my_3d_camera;
    private GameObject my_main_camera;
    private GameObject my_animation_camera;

    private my_tittle_figure my_tittle;

    private AudioSource my_audio_sound;
    private AudioClip[] my_sounds;

    private void Start()
    {
        my_curr_time = 0;
        my_array = new my_tetrino_element[wid, hei];
        pref_tetrino = Resources.Load("my_prefab/my_tetrino_figure") as
GameObject;
        pref_tetrino_object =
Resources.Load("my_prefab/my_prefab_tetrino_o");

        my_tittle = FindObjectOfType<my_tittle_figure>();

        my_sounds = Resources.LoadAll<AudioClip>("my_sound");
        my_audio_sound = GetComponent<AudioSource>();
        MyRandomSound();

        my_main_camera = GameObject.FindGameObjectWithTag("MainCamera");
        my_3d_camera = GameObject.FindGameObjectWithTag("my_3d_camera");
        my_animation_camera =
GameObject.FindGameObjectWithTag("my_camera_animation");
        my_animation_camera.SetActive(false);
        MyOnSwitchCamera(false);
```

```
        my_hud.txt_score =
GameObject.FindGameObjectWithTag("my_score").GetComponent<Text>();
        my_hud.txt_line =
GameObject.FindGameObjectWithTag("my_line").GetComponent<Text>();
        my_hud.txt_level =
GameObject.FindGameObjectWithTag("my_level").GetComponent<Text>();
        my_hud.my_speed = 0.5f;
        my_hud.AddLevel(1);

        my_figure_random = CreateRandomFigure();
        CreteFigure(my_figure_random);
        my_figure_random = CreateRandomFigure();


my_tittle.GetComponentInChildren<my_tetrino_data>().MyInitialize(my_figure_ra
ndom);

        for(int y = 0; y < hei; y++)
            for (int x = 0; x < wid; x++)
            {
                GameObject go = Instantiate(pref_tetrino_object, new
Vector3(x * my_step, y * my_step, 0),
                    Quaternion.identity) as GameObject;

                my_array[x, y] = go.GetComponent<my_tetrino_element>();

            }

    }

    private void MyRandomSound()
    {
        int index = UnityEngine.Random.Range(0, my_sounds.Length);
        my_audio_sound.clip = my_sounds[index];
    }

    private TetrinoFigure CreateRandomFigure()
    {
        return(TetrinoFigure)UnityEngine.Random.Range(0, 5);
    }

    private void CreteFigure(TetrinoFigure _figure)
    {
```

```
        my_figure = Instantiate(pref_tetrino, new Vector3(my_step * 6,
my_step * (hei - 2), 0),
            Quaternion.identity).GetComponent<my_tetrino_figure>();

my_figure.GetComponentInChildren<my_tetrino_data>().MyInitialize(_figure);

        if(my_hud.my_int_level < 3)

my_figure.GetComponentInChildren<my_tetrino_data>().MySetColor(UnityEngine.Ra
ndom.ColorHSV(0.4f, 0.6f, 1, 1, 1, 1, 1, 1));
        else if(my_hud.my_int_level < 5)

my_figure.GetComponentInChildren<my_tetrino_data>().MySetColor(UnityEngine.Ra
ndom.ColorHSV(0.5f, 0.7f, 1, 1, 1, 1, 1, 1));
        else if(my_hud.my_int_level < 7)

my_figure.GetComponentInChildren<my_tetrino_data>().MySetColor(UnityEngine.Ra
ndom.ColorHSV(0.8f, 1.0f, 1, 1, 1, 1, 1, 1));
        else if(my_hud.my_int_level < 10)

my_figure.GetComponentInChildren<my_tetrino_data>().MySetColor(UnityEngine.Ra
ndom.ColorHSV(0.2f, 0.4f, 1, 1, 1, 1, 1, 1));
        else if(my_hud.my_int_level >= 10)

my_figure.GetComponentInChildren<my_tetrino_data>().MySetColor(UnityEngine.Ra
ndom.ColorHSV(0.3f, 0.5f, 1, 1, 1, 1, 1, 1));



        StartCoroutine(my_update(my_hud.my_speed));

    }

    private IEnumerator my_update(float _time)
    {
        while(true)
        {
            yield return new WaitForSeconds(_time);
            my_figure.MyDropTetrino(true);

            if(CheckPreIntersect(my_figure))
                break;
        }
```

```
        AddToArray();
        Destroy(my_figure.gameObject);
        MyRemoveFullLine();

        if(!IsGameOver())
        {
            CreteFigure(my_figure_random);
            my_figure_random = CreateRandomFigure();

my_tittle.GetComponentInChildren<my_tetrino_data>().MyInitialize(my_figure_ra
ndom);
        }
        else
        {
            my_main_camera.SetActive(false);
            my_3d_camera.SetActive(false);
            my_animation_camera.SetActive(true);

            my_tittle.gameObject.SetActive(false);

        }

    }

    private void AddToArray()
    {
        GameObject[] go =
my_figure.GetComponentInChildren<my_tetrino_data>().GetTetrinoArray;

        for (int ind = 0; ind < go.Length; ind++)
        {
            int x = (int)go[ind].transform.position.x;
            int y = (int)go[ind].transform.position.y;

            my_array[x, y].set_tetrino_active(true); //смерть//
            my_array[x,
y].set_color(my_figure.GetComponentInChildren<my_tetrino_data>().my_color);
        }
    }

    private void MyRemoveFullLine()
    {
        int[] removeLine =  MyCheckFullLine();
```

```
        for (int ind = 0; ind < removeLine.Length; ind++)
        {
            for (int x = 0; x < wid; x++)
                my_array[x, removeLine[ind]].set_tetrino_active(false);

            my_hud.AddScore(removeLine.Length == 4 ? 750 : 350);

        }

        if(removeLine.Length != 0)
        {
            MyRandomSound();
            my_audio_sound.Play();
            int[] empty_line = MyCheckEmpryLine();
            bool[,] arr_new_tetrino = new bool [wid, hei];

            int start_y = 0;

            my_hud.AddLine(removeLine.Length);

            for (int y = 0; y < hei; y++)
            {
                if(MySkipTheLine(empty_line, y))
                    continue;

                for (int x = 0; x < wid; x++)
                    arr_new_tetrino[x, start_y] = my_array[x,
y].get_isActive_tetrino();

                start_y++;
            }

            MySetTetrinoArray(arr_new_tetrino);
        }

    }

    public void MyOnSwitchCamera(bool _is3D)
    {
        my_main_camera.SetActive(!_is3D);
        my_3d_camera.SetActive(_is3D);

    }
```

```
public void MyOnStartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

public void MyOnPause(bool _isPause)
{
    Time.timeScale = _isPause ? 0 : 1;

}

private bool IsGameOver()
{
    for (int ind = 0; ind < wid; ind++)
    {
        if(my_array[ind, hei - 3].get_isActive_tetrino())
            return true;

    }

    return false;
}

private void MySetTetrinoArray(bool[,] _arr_new)
{
    for (int y = 0; y < hei; y++)
        for (int x = 0; x < wid; x++)
            my_array[x, y].set_tetrino_active(_arr_new[x, y]);


}

private bool MySkipTheLine(int[] _empty_line, int _y)
{
    for (int y = 0; y < _empty_line.Length; y++)
    {
        if(_empty_line[y] == _y)
            return true;
    }

    return false;

}
```

```
private int[] MyCheckEmpryLine()
{
  List<int> arr = new List<int>();

    for (int ind = 0; ind < hei; ind++)
    {
        int count_line_x = 0;

        for (int x = 0; x < wid; x++)
        {
            if(my_array[x, ind].get_isActive_tetrino())
                break;
            else
                 count_line_x++;
        }

        if(count_line_x == wid)
            arr.Add(ind);
    }

    return arr.ToArray();
}

private int[] MyCheckFullLine()
{
    List<int> arr = new List<int>();

    for (int ind = 0; ind < hei; ind++)
    {
        int count_line_x = 0;

        for (int x = 0; x < wid; x++)
        {
            if(my_array[x, ind].get_isActive_tetrino())
                count_line_x++;
            else
                break;
        }

        if(count_line_x == wid)
            arr.Add(ind);
    }
```

```
        return arr.ToArray();
    }

    private void Update()
    {
    if(my_figure)
     {
        if(Input.GetButtonDown("RotateTetrino"))
        {

my_figure.GetComponentInChildren<my_tetrino_data>().MyRotation(true);
            if(CheckIntersect(my_figure))

my_figure.GetComponentInChildren<my_tetrino_data>().MyRotation(false);
        }

        if(Input.GetButtonDown("LeftTetrino"))
        {
            my_curr_time = 0;
            my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
            if (CheckIntersect(my_figure))
                my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
        }

        else if(Input.GetButtonDown("RightTetrino"))
        {
            my_curr_time = 0;
            my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
            if (CheckIntersect(my_figure))
                my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
        }

        if(Input.GetButton("DownTetrino"))
            MyInputPress(MyDirecrionTetrino.DOWN, 0.04f);
        if (Input.GetButton("RightTetrino"))
            MyInputPress(MyDirecrionTetrino.RIGHT, 0.08f);
        else if(Input.GetButton("LeftTetrino"))
            MyInputPress(MyDirecrionTetrino.LEFT, 0.08f);

        if(Input.GetButtonUp("LeftTetrino") ||
Input.GetButtonUp("RightTetrino"))
            my_curr_time = 0;
```

```
 }
}

private void MyInputPress(MyDirecrionTetrino _dir, float _time)
{
    my_curr_time += Time.deltaTime;
    if(my_curr_time > _time)
    {
        my_curr_time = 0;

        if(_dir == MyDirecrionTetrino.LEFT)
     {
        my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
        if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
     }

    else if(_dir == MyDirecrionTetrino.RIGHT)
     {
        my_figure.MySetDirection(MyDirecrionTetrino.RIGHT);
        if (CheckIntersect(my_figure))
            my_figure.MySetDirection(MyDirecrionTetrino.LEFT);
     }
     else if(_dir == MyDirecrionTetrino.DOWN)
     {
        my_figure.MyDropTetrino(true);
        if(CheckIntersect(my_figure))
            my_figure.MyDropTetrino(false);
     }
    }
}

 private bool CheckIntersect(my_tetrino_figure _figure)
{
    for (int ind = 0; ind < _figure.GetSegments().Length; ind++)
    {
        int x = (int)_figure.GetSegments()[ind].transform.position.x;
        int y = (int)_figure.GetSegments()[ind].transform.position.y;

        bool is_intersect = IsIntersect(x, y);

        if(is_intersect)
            return is_intersect;
```

```
        }

        return false;
    }
    private bool CheckPreIntersect(my_tetrino_figure _figure)
    {
        for (int ind = 0; ind < _figure.GetSegments().Length; ind++)
        {
            int x = (int)_figure.GetSegments()[ind].transform.position.x;
            int y = (int)_figure.GetSegments()[ind].transform.position.y;

            bool is_intersect = IsIntersect(x, y);

            if(is_intersect)
            {
                _figure.MyDropTetrino(false);
                return is_intersect;
            }
        }

        return false;
    }

    private bool IsIntersect(int _x, int _y)
    {
       try
       {
            if(my_array[_x, _y].get_isActive_tetrino())
                return true;
       }
       catch(System.Exception ex) { return true; }

       return false;
    }
}
```