МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «РОЗРОБКА ГРИ "ТРИ У РЯД" З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ UNITY»

Виконав: студент 4 курсу, групи 6.1229
спеціальності 122 комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми комп'ютерні науки
(назва освітньої програми)
О.О. Яцун
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, доцент к.т.н. Борю С. Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри програмної інженерії, к.ф – м.н., доцент Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 комп'ютерні науки
(шифр і назва)

Освітня програма комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Чопоров С.В.
(підпис)

" ____ " _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Яцуну Олександру Олександровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка гри "Три у ряд" з використанням фреймворку Unity

керівник роботи доцент кафедри комп'ютерних наук, доцент к.т.н. Борю С. Ю.
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 06.06.2023

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Особливості розробки комп'ютерних ігор.
2. Розробка сценарія та алгоритмів.
3. Реалізація проекту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 22.02.2023

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.	21.03.2023	
2.	Збір вихідних даних.	26.03.2023	
3.	Обробка методичних та теоретичних	15.04.2023	
	джерел.		
4.	Розробка першого та другого розділу.	23.04.2023	
5.	Розробка третього розділу.	26.04.2023	
6.	Оформлення та нормоконтроль	06.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2023	

Студент _____ О.О. Яцун
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.Ю. Борю
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка гри "Три у ряд" з використанням фреймворку Unity»: 65 сторінок, 8 рис., 5 джерел, 6 додатків.

Об’єкт дослідження – процес розробки гри "Три у ряд".

Мета роботи: розробка онлайн ігри "Три у ряд" та вивчення процесу розробки такого типу гри.

Метод дослідження – аналіз літературних джерел, проектування та розробку гри "Три у ряд".

Для написання коду було використано середовище розробки та бібліотеку Unity і низку інших інструментальних засобів середовища Visual Studio. Основні завдання роботи включають вибір технологій розробки, створення архітектури гри, реалізацію ігрових механік, включаючи створення дошки, генерацію елементів, читання переміщень елементів, а також їх руйнування. В рамках дослідження також будуть розглянуті особливості ігор на рушії Unity, а також можливості та обмеження, пов'язані з вибраними технологіями та архітектурою гри. Результатом роботи буде функціональна та готова до використання гра " Три у ряд", а також аналіз та оцінка процесу розробки та можливих покращень для майбутніх проектів.

SUMMARY

Bachelor's qualification work "Development of the game "Three in a row" using the Unity framework": 65 pages, 8 figure, 5 references , 6 appendices .

The object of research is the process of developing the game "Three in a Row".

The purpose of the work: the development of the online game "Three in a row" and the study of the development process of this type of game.

Research method – analysis of literary sources, design and development of the game "Three in a row".

The development environment and Unity library and a number of other tools of the Visual Studio environment were used to write the code. The main tasks of the work include the selection of development technologies, the creation of the game architecture, the implementation of game mechanics, including the creation of the board, the generation of elements, reading the movements of elements, as well as their destruction. The study will also examine the specifics of Unity-powered games, as well as the capabilities and limitations associated with the selected technologies and game architecture. The result of the work will be a functional and ready-to-use Match 3 game, as well as an analysis and evaluation of the development process and possible improvements for future projects.

ЗМІСТ

ВСТУП

Створення ігрових додатків є відмінною можливістю для розвитку навичок у програмуванні та дизайні геймінг-світів. Unity - це потужна та дуже популярна платформа для розробки ігор, що дозволяє створювати ігри на різних платформах, включаючи ПК, мобільні пристрої, консолі та віртуальну реальність. Одним із завдань кваліфікаційної роботи є розробка гри у стилі Три у ряд на платформі Unity. Unity також використовується для створення візуальних ефектів у фільмах та інтерактивних додатків для освіти та виробництва. Unity має велику спільноту розробників, яка допомагає один одному вирішувати проблеми під час розробки, тому є дуже популярною серед розробників ігор.

Жанр "Три у ряд" (Match-3) є одним з найпопулярніших жанрів головоломок в ігровій індустрії. У цих іграх гравці повинні зібрати групу з трьох або більше однакових елементів на ігровому полі, щоб знищити їх і отримати очки. Головна мета гри полягає у досягненні заданого результату за обмежений час або кількість кроків. Зазвичай гра має різні рівні складності, що включають в себе нові елементи, які з'являються на полі, різноманітні перешкоди, а також різні бонуси та підказки, які можуть допомогти гравцеві. Такі ігри є простими у геймплеї та мають яскраву графіку та звуковий ефект, що дозволяє залучити до гри гравців різного віку та досвіду. У даній кваліфікаційній роботі будуть описані основні етапи розробки ігри в жанрі "Три у ряд" на платформі Unity. Вона включатиме створення ігрового світу, розробку геймплею та взаємодії з гравцем, додавання звукового супроводу та багато іншого. Загалом, проект дозволить детально ознайомитись з процесом розробки гри в жанрі "Три у ряд" на платформі Unity та отримати практичні навички у програмуванні та дизайні ігор.

1 ОСОБЛИВОСТІ РОЗРОБКИ КОМП’ЮТЕРНИХ ІГОР

1.1 Основні види комп’ютерних ігор

Розробка комп'ютерних ігор - це дуже складний та трудомісткий процес, який потребує високої кваліфікації фахівців у різних галузях. Для успішної розробки гри потрібні програмісти, графічні дизайнери, аніматори, композитори, звукорежисери, тестери та багато інших фахівців. Крім того, розробка ігор потребує великих інвестицій, тому від компаній-розробників часто вимагається високий рівень ризику. Однак, успішна гра може принести значну прибуток і стати великим успіхом в індустрії розваг.

Наприкінці, можна зазначити, що комп'ютерні ігри є не лише розважальним продуктом, але й інструментом для розвитку та навчання, як у дітей, так і у дорослих. Вони можуть допомогти вивченню нових навичок, розвитку креативності та стратегічного мислення. Тому розробка комп'ютерних ігор - це не тільки креативний процес, а й важлива складова сучасної культури та освіти.

У першій частині «Підготовка проєкту» [3] описано такі основні комп’ютерні жанри ігор як:

- Екшен (Action): це ігри, в яких гравець керує персонажем, який здійснює різноманітні дії, такі як біг, стрибки, стрілянину тощо.

- Стратегії (Strategy): у цих іграх гравець керує групою персонажів, розвиває свою базу, планує атаки та оборону.

- Рольові ігри (RPG): у таких іграх гравець керує персонажем, розвиває його, виконує квести, бореться з монстрами та іншими гравцями.

- Головоломки (Puzzle): це ігри, в яких гравець вирішує різні головоломки, завдання та інші складні задачі.

- Гонки (Racing): у таких іграх гравець керує автомобілем, мотоциклом або іншим транспортним засобом, змагаючись з іншими гравцями або комп'ютером.

- Симулятори (Simulation): у цих іграх гравець керує імітацією реального об'єкта, наприклад авіасимулятором, симулятором життя і т.д.

- Шутери (Shooters): це ігри, в яких гравець керує персонажем, який стріляє зі зброї та бореться з ворогами.

- Аркади: у таких іграх гравець змагається з іншими гравцями або комп'ютером в різних завданнях.

- Спортивні ігри (Sports): це ігри, в яких гравець керує командою або спортсменом у симуляції різних видів спорту.Також існують ігри, які створені для ігрових консолей, мобільні ігри, браузерні ігри. Мобільні ігри стають все більш популярними та доступними, і багато людей грають у них у свій вільний час.

Одним з найважливіших аспектів розробки ігор є використання потужного інструментарію та технологій, що надають розробникам можливість створювати складні та цікаві ігри. Основні двигуни розробки ігор - це програмні платформи, які надають розробникам доступ до необхідних інструментів для створення графіки, анімації, фізики, звуку, мережевої взаємодії та інших аспектів гри.

1.2 Основні інструментальні системи розробки КІ

Переглянув сайт [1] та визначив декілька основних інструментальних систем розробки комп'ютерних ігор, які включають інтегровані середовища розробки (IDE), графічні движки, мови програмування та редактори ресурсів. Наприклад, популярні IDE для розробки ігор включають Unity, Unreal Engine та GameMaker Studio.

Графічні движки, такі як Unity та Unreal Engine, надають потужні інструменти для візуального створення та налаштування графіки, фізики та штучного інтелекту в іграх.

Комп'ютерні ігри стали одним з найпопулярніших видів розваг для мільйонів людей по всьому світу. Розробка ігор - це дуже складний та багатоаспектний процес, який включає в себе роботу з графікою, звуком, мережевими технологіями, штучним інтелектом та багато іншого. Розробка комп'ютерних ігор - це креативний, але водночас вимогливий процес, що потребує високого рівня знань і вмінь.

Як зазначалося в [3] основні етапи розробки ігор такі:

- планування: на цьому етапі визначається концепція ігри, її ідея, сценарій, персонажі, ігровий світ, механіка гри та багато іншого;

- прототипування: на цьому етапі створюється прототип гри, що дозволяє перевірити роботу основних механік і геймплею;

- розробка графіки: дизайнери розробляють графіку для персонажів, об'єктів, та ігрового середовища;

- розробка звуку: композитори створюють музику, звукові ефекти та діалоги персонажів;

- розробка мережевого дизайну: на цьому етапі розробляється мережева архітектура гри, яка дозволяє гравцям підключатися до мережі та грати разом;

- розробка штучного інтелекту: програмісти створюють алгоритми для реалістичної поведінки персонажів та противників;

- тестування: на цьому етапі гра піддається випробуванням на різних платформах, щоб виявити та виправити помилки і баги;

- випуск: на останньому етапі гра готується до випуску та дистрибуції. Зазвичай гра публікується на цифрових платформах, таких як Steam, PlayStation Network, Xbox Live, або на мобільних платформах, таких як App Store та Google Play.

Кожен етап вимагає специфічних інструментів і технологій, що допомагають розробникам створювати якісні та захоплюючі комп'ютерні ігри.

1.3 Обзор популярних комп’ютених ігор жанру «Три у ряд»

У статтях стосовно популярності комп’ютерних ігор на форумі [1] зазначено, що жанр "Три у ряд" є одним з найпопулярніших серед гравців різного віку та ігрового досвіду. Ці ігри пропонують захоплюючу комбінацію розваги та головоломок, де гравець має завдання знаходити й з'єднувати трьох або більше однакових об'єктів для їх знищення. Вони задовольняють потреби тих, хто шукає швидку гру з миттєвими результатами, а також задумливих стратегічних гравців.

Основним сценарієм гри "Три у ряд" є збирати максимальну кількість очок або досягти певної мети у визначену кількість ходів. Гравець має застосовувати свою спостережливість та логічне мислення, щоб знаходити найбільш вигідні комбінації об'єктів та здійснювати стратегічні рухи. Ігрові механіки є простими та доступними, але водночас вимагають певного рівня стратегії та планування з боку гравця.

Гра "Три у ряд" може бути віднесена до жанру головоломок та логічних ігор. Гравець має розглядати ігрове поле з увагою, аналізувати можливі варіанти ходів та передбачати наслідки своїх рішень. Цей жанр дозволяє гравцям розвивати свої розумові навички, такі як спостережливість, концентрація, логічне мислення та прийняття рішень. Також ігри цього жанру доступні на різних платформах, включаючи комп'ютери, мобільні телефони та планшети. Це дозволяє гравцям насолоджуватися грою в будь-який зручний для них час та місце

Окрім того, гра має приємну графіку, музику та звукові ефекти, що створюють гарний настрій під час гри. Яскраві кольори, анімація та візуальні ефекти допомагають створити привабливе ігрове середовище та залучити гравців до процесу.

Загалом, гра "Три у ряд" відноситься до жанру головоломок та логічних ігор і надає гравцям можливість насолоджуватися захоплюючим геймплеєм, сприяє розвитку розумових навичок та забезпечує доступну розвагу на різних платформах.

Після аналізу одного з найпопулярніших магазину ігор [4] визначив основних лідерів серед ігор жанру «Три у ряд», а саме:

- Candy Crush Saga – ця гра має безліч рівнів, в яких гравець повинен збирати різнокольорові цукерки, розташовуючи їх у ряди по три або більше.

- Bejeweled – гра, випущена компанією PopCap, є однією з перших ігор у цьому жанрі. Гравець повинен збирати дорогоцінні камені, розташовуючи їх у ряди по три або більше.

- Farm Heroes Saga – гра від творців Candy Crush Saga, в якій гравець повинен зібрати різні фрукти та овочі, розташовуючи їх у ряди по три або більше.

- Jewel Quest – гра має велику кількість рівнів, в яких гравець повинен збирати дорогоцінні камені та розблоковувати складніші головоломки.

- Fishdom – гра зроблена в подібному стилі, але вона має різноманітніші завдання, включаючи розвиток власного акваріума та догляд за рибками.

Загалом, зважаючи на кількість завантажень та популярність ігор жанру "Три у ряд" на різних платформах, можна точно сказати, що вони продовжують залучати широке коло гравців своєю привабливістю та захоплюючістю.

1.4 Постановка задачі

Під час розробки комп’ютерної гри у жанрі «Три у ряд» визначив для такі основні завдання:

1. Розробка імплементації ігрового поля та відображення графіки.
2. Реалізація механік знаходження та з'єднання однакових об'єктів.
3. Розробка системи підрахунку очків та управління рівнями гри.

4. Інтеграція звукових ефектів та музики для створення належної атмосфери гри.

5. Тестування гри та виявлення та виправлення помилок.

У розробці гри "Три у ряд" з використанням фреймворку Unity, основною проблемою є забезпечення оптимальної продуктивності та якості гри. Потрібно знайти оптимальний спосіб реалізації логіки гри, щоб уникнути затримок та лагів, особливо при великій кількості елементів на полі.

Загальна мета роботи полягає у створенні якісної та захоплюючої гри "Три у ряд" за допомогою Unity на мові C#, яка задовольнятиме вимоги користувачів щодо геймплею, графічного дизайну та продуктивності. Для досягнення цієї мети необхідно успішно вирішити всі поставлені задачі та перевірити гру на наявність помилок та недоліків за допомогою тестування.

Оглянув існуючі комп'ютерні ігори жанру "Три у ряд" та способи їх розробки та програмування, що дало змогу отримати уявлення про те, як цей жанр реалізовується та які можливості є для створення власної гри.

2 РОЗРОБКА СЦЕНАРІЯ ТА АЛГОРИТМІВ

2.1 Розробка сценарію

Розробка сценарію гри "Три у ряд" є важливим етапом у процесі створення аркадних ігор. У [3] зазначається, що сценарій визначає загальну сюжетну лінію, мету гри, завдання гравця та основні елементи геймплею.

На [1] описуються різні ігри "Три у ряд" з різними механіками та варіаціями. Деякі з них фокусуються на збиранні кристалів або предметів шляхом з'єднання трьох і більше однакових елементів, інші можуть мати елементи розблокування рівнів або виконання завдань на час. Деякі ігри можуть включати в себе елементи стратегії або головоломок для досягнення мети.

Визначення тематики гри або стилю гри "Три у ряд". Наприклад, це може бути пригодницька гра в джунглях, космічна тематика або світ фантастичних чарівників. Важливо вибрати тему, яка буде цікавою для цільової аудиторії.

Створення головного героя або групи персонажів, які будуть відображати основну тематику гри. Визначення їх унікальних характеристик, навичків та мету в грі. Наприклад, може бути головний герой, який шукає загублені артефакти або пригодницька група, що рятують світ від злого угруповання.

Розробка сюжетної лінії, яка буде провадити гравця через різні рівні або етапи гри. Створення завдання, які гравець повинен буде виконати, щоб просунутися вперед.

Створення цікавих перешкод та бонусів, які можуть з'являтися на рівнях гри. Це можуть бути блоковані елементи, ускладнення у вигляді таймерів або обмеження рухів.

У ролі гравця головне завдання полягає в досягненні максимальної кількості очок або виконанні певного завдання за обмежену кількість ходів. Гравець повинен спостерігати за ігровим полем, аналізувати можливі комбінації об'єктів та приймати стратегічні рішення для досягнення мети. Чим більше комбінацій гравець знаходить і здійснює, тим більше очок він отримує.

Роль програми у грі "Три у ряд" полягає в управлінні грою, перевірці комбінацій об'єктів та підрахунку очків. Він відповідає за відображення ігрового поля, обробку рухів гравця та генерацію нових об'єктів на полі. Вона також здійснює перевірку на наявність комбінацій об'єктів та надає відповідні очки гравцю за успішні з'єднання.

Оцінка гри "Три у ряд" здійснюється на основі кількості набраних очків або досягнення певної мети. Гравець може отримувати різні рівні відзнак, відповідно до свого результату в грі. У своїй комп’ютерній грі жанру «Три у ряд» я вирішив встановити систему рангів за проходження рівнів, від "початківця" до "майстра", що відображають успішність гравця у досягненні цілей гри.

Способи розробки комп'ютерних ігор жанру "Три у ряд" можуть варіюватися, але одним з найпопулярніших інструментів для розробки таких ігор є движок Unity. Unity забезпечує зручне середовище розробки, підтримку різних платформ, інструменти для роботи з графікою, звуком, анімаціями та фізикою.

Програмування гри "Три у ряд" з використанням Unity включають розробку скриптів на мові програмування C#.

Скрипти контролюють рух об'єктів, обробляти введення гравця, розраховувати логіку "Три у ряд" (наприклад, перевіряти наявність збігів, виконувати дії після успішних комбінацій) та взаємодіяти з іншими компонентами гри.

2.2 Розробка інформаційної моделі

Після прочитання та аналізу розділу «Scripts» на сайті [5] визначив, що розробка інформаційної моделі є важливою частиною у розробці комп’ютерної гри «Три у ряд».

Створення ігрових активів для гри "Три у ряд" є важливим етапом в процесі розробки інформаційної моделі гри, оскільки активи становлять основну частину геймплею. Ігрові активи включають в себе елементи, які гравець взаємодіє з ними під час гри, такі як блоки з різними формами та кольорами, які необхідно збирати в лінії, щоб отримати очки та рухатися далі по рівнях гри.

Крім того, важливо враховувати оптимізацію графіки та розмір ігрових активів, щоб гра працювала на різних пристроях та мала достатню швидкість реакції. Також можна використовувати спеціальні ефекти та звукові елементи, щоб зробити гру більш захоплюючою та емоційно насиченою.

У процесі розробки гри "Три у ряд" необхідно забезпечити високу якість графіки та анімації ігрових активів, щоб зробити гру цікавішою та привабливішою для гравців. Крім того, важливо враховувати особливості цього жанру гри та використовувати кращі практики у створенні ігрових активів, щоб забезпечити успіх гри та задоволення гравців.

Рисунок 1-1 – Основні текстури рухомих блоків

Об’єкти були створенні за допомогою програми Adobe Photoshop. Всі картинки були одразу поміщені в папку "Images" для зручності їх використання.

Розробка ігор – це процес, який поєднує креативність, програмування та дизайн. Завдяки розвитку ігрових двигунів, таких як Unity, створення ігор доступне для всіх, хто має бажання та інтерес. Unity - один з найбільш популярних ігрових двигунів на ринку, який дозволяє створювати ігри для різних платформ, включаючи комп'ютери, мобільні пристрої, ігрові консолі та навіть віртуальну реальність. У цьому розділі ми розглянемо процес створення гри на Unity з самого початку.

Ми будемо використовувати C# як мову програмування, яка є однією з основних мов розробки в Unity.

Створення гри "Три у ряд" на рушії Unity включає послідовність дій як гри, так і гравця, врахована така послідовність дій:

1. Гравець запускає гру "Три у ряд" на своєму пристрої.

2. Гравець має можливість вибрати режим гри, наприклад, гру проти комп'ютера або мультиплеєрний режим.

3. На екрані з'являється ігрове поле, яке складається з сітки, де розташовуються ігрові об'єкти.

4. Алгоритм генерує початковий рівень, розташовуючи об'єкти на ігровому полі.

5. Гравець має можливість обирати два об'єкти на ігровому полі і міняти їх місцями з метою створення комбінації з трьох або більше однакових об'єктів.

6. Після кожного ходу гра перевіряє, чи утворилися нові комбінації об'єктів. Якщо так, вони зникають з ігрового поля, а гравець отримує очки.

7. Гра перевіряє, чи залишилися ще можливі комбінації об'єктів на ігровому полі. Якщо немає, гра завершується.

8. Гра відображає кількість набраних гравцем очків.

9. Після завершення ходу гравця або комп'ютера, алгоритм оновлює рівень шляхом додавання нових об'єктів на ігрове поле.

10. Гра повторює цей цикл ходів та перевірок до тих пір, поки гравець або комп'ютер не досягне певної мети, яка встановлена в грі.

Така послідовність дій передбачає взаємодію гравця з ігровим полем, зміну об'єктів на полі, перевірку комбінацій, набір очок та оновлення рівня гри. Кожен хід гравця впливає на подальший розвиток гри, алгоритми генерації рівнів та обробки комбінацій забезпечують гратимість та складність гри. Після закінчення цієї кваліфікаційної роботи буде створено гру на основі Unity, яку можна публікувати у інтернеті для загального користування. Однією з головних переваг загального доступу це можливість грати на будь якому пристрої, що має підключення до Інтернету, що дозволяє грати в будь який час і в будь якому місці. Це також означає, що ігри можуть бути швидко та легко оновлені, дозволяючи гравцям отримувати доступ до нових контентів та покращеної геймплейної механіки.

Для збереження стану гри, включаючи розташування фігур на ігровому полі, їх типи та значення, використовується об'єкт класу ArrayLayout, який містить двовимірний масив rowData[]. Кожен елемент масиву rowData представляє рядок на ігровому полі і містить масив значень типу bool, що відповідає наявності фігур на конкретних позиціях.

Дані про користувачів, такі як імена, рівні, досягнення тощо, можуть бути збережені у базі даних або використовуючи систему збереження даних фреймворку Unity, таку як PlayerPrefs.

Робоча форма, що складається з елементів, представлена за допомогою двовимірного масиву або іншої структури даних, яка відображає стан кожного елементу (зайнято або порожньо).

Інформація про фігури (їх тип, значення тощо) може бути збережена в окремому класі, наприклад, NodePiece, який містить поля для збереження цієї інформації. Об'єкти NodePiece розташовані відповідно до робочої форми, і їхнє розташування може змінюватись за допомогою руху користувача.

2.3 Розробка алгоритмів

У Розробці алгоритмів зосереджуємося на процесі реалізації сценарію гри "Три у ряд" з використанням рушія Unity. Після аналізу другої глави «Робота над проєктом» [3] встановлено основні алгоритми, які використовуються для реалізації різних елементів гри, включаючи генерацію рівнів та набір очок.

Також були виявлені основні задачі гри:

1. Генерація рівнів: Одним з ключових елементів гри "Три у ряд" є генерація рівнів, на яких гравець буде грати. У цьому розділі будуть розглянуті алгоритми, які використовуються для створення різних комбінацій об'єктів на ігровому полі. Були розглянути алгоритми генерації рівнів на основі випадкових чисел або з певними обмеженнями, що забезпечують рівновагу та інтерес гри.

Інформаційна модель генерації рівнів виглядає наступним чином:

ArrayLayout - це клас, що представляє структуру для збереження інформації про розміщення елементів на рівні гри. Він містить поле grid, яке є посиланням на об'єкт типу Grid, що відповідає за управління і відображенням рівня гри. Крім того, він містить масив rows, який представляє кожен рядок на рівні. Кожен рядок є структурою rowData, яка містить масив row типу bool, що відображає наявність елементів у відповідному рядку. Завдяки такій структурі даних можна визначати, які клітинки на рівні зайняті або вільні.

rowData - структура, що представляє рядок на рівні гри. Вона містить масив row типу bool, де кожен елемент вказує, чи зайнята відповідна клітинка рядка.

Алгоритм генерації рівнів має наступний хід дій:

1) Ініціалізація порожнього рівня. Створення об'єкта **ArrayLayout** і задання його розміру.
2) Заповнення рівня елементами гри включає в себе наступні дії:
   - вибір випадкового елемента з пулу доступних елементів гри;
   - розташування вибраного елемента вільною клітинкою на рівні;
   - позначення зайнятості відповідної клітинки у **ArrayLayout**;
   - перевірка створених ланцюжків.
3) Після заповнення рівня перевіряється, чи утворюються ланцюжки з трьох або більше однакових елементів. Якщо такі ланцюжки з'являються, вони будуть видалені, а клітинки, що залишилися порожніми, заповнені новими елементами.
4) Перевірка наявності можливих ходів. Після видалення ланцюжків перевіряється, чи є можливі ходи, які можуть бути зроблені гравцем. Якщо можливих ходів немає, застосовується алгоритм генерації рівня заново.
5) Повторення кроків 2-4, поки не буде досягнуто потрібного стану рівня (наприклад, заданої кількості ланцюжків, розташування спеціальних елементів тощо) або не будуть виконані певні умови завершення генерації рівня.

2. Набір очок: У грі "Три у ряд" гравець отримує очки за успішне з'єднання трьох або більше однакових об'єктів. В розділі про розробку алгоритмів описані алгоритми, які визначають, які комбінації об'єктів заслуговують на очки та які правила застосовуються при їх підрахунку. Також розглянуті алгоритми, які відповідають за анімацію та візуальне представлення набору очок.

Очки (Score): В грі "Три у ряд" очки використовуються для відстеження досягнень гравця та його прогресу. Очки набираються, коли гравець успішно утворює лінії з трьох або більше однакових елементів. Кожна утворена лінія приносить гравцеві певну кількість очок, яка залежить від складності та рівня гри.

Під час перевірки наявності ліній з однакових елементів, коли виявляється така лінія, до загального рахунку гравця додається певна кількість очок.

Кількість очок, яку гравець отримує за кожну лінію, може збільшуватися з кожним рівнем гри або за досягнення певних цілей. Очки відображаються на екрані гравця, у верхній частині гри або на панелі результатів. Для збереження та відстеження очків гравця використовуються наступні елементи:

1) Змінна або об'єкт, що зберігає кількість набраних очок гравцем.
2) Функція або метод, який оновлює відображення очків на екрані гравця та відображає актуальний рахунок.

Механізм збереження та завантаження рахунку гравця, який дозволяє зберігати прогрес гравця між ігровими сеансами. Додатково до звичайного набору очок, гра містить систему бонусів та винагород, які гравець отримує за досягнення певних цілей або виконання особливих дій. Ці бонуси можуть додавати додаткові очки до загального рахунку гравця або надавати особливі переваги в грі.

3. Управління грою: Розробка алгоритмів також включає в себе аспекти управління грою, обробку дій користувача та реакцію гри на ці дії. Розділ управління грою присвячений алгоритмам обробки натискання кнопок, перетягування об'єктів та взаємодії з ігровим полем.

Класи та компоненти:

GameBoard: цей клас відповідає за управління і відображенням грального дошки. Він містить методи для створення та оновлення дошки, обробки рухів гравця, перевірки умов перемоги та завершення гри.

Piece: цей клас представляє один елемент гри (наприклад, символ або фігуру). Він містить необхідні властивості та методи для маніпуляції елементом, такі як зміна стану, відображення тощо.

GameManager: цей клас відповідає за керуванням логікою гри, такою як розпочати нову гру, перевірити умови перемоги, обробити взаємодію гравця з елементами гри та інше.

Гравець взаємодіє з елементами гри шляхом кліку на них за допомогою миші або дотику на сенсорному пристрої. Після вибору елементу гравець може виконати рух, який вплине на розташування елементів на гральній дошці.

Управління рухами гравця відбувається за допомогою обробки подій кліку або дотику. Клас GameBoard відповідає за отримання інформації про вибрані елементи та обробку цих рухів.

Після кожного руху гравця, гра перевіряє умови перемоги. Якщо знайдено рядок, стовпець або діагональ з трьома однаковими елементами, гра визначає переможця або показує повідомлення про закінчення гри.

Інформаційна модель управління грою виглядає наступним чином:

GameBoard.InitializeBoard(): метод, який ініціалізує гральну дошку, встановлюючи початкове розташування елементів та їх властивості.

GameBoard.UpdateBoard(): метод, який оновлює стан гральної дошки після кожного руху гравця.

GameBoard.CheckWinCondition(): метод, який перевіряє умови перемоги, шукаючи рядки, стовпці та діагоналі з трьома однаковими елементами.

GameManager.StartNewGame(): метод, який розпочинає нову гру, скидаючи гральну дошку до початкового стану та очистивши рахунок гравця.

GameManager.EndGame(): метод, який викликається після завершення гри, показуючи повідомлення про перемогу або нічию.

4. Мережевий зв’язок: Окрім проходження звичайного рівня, гра "Три у ряд" також може мати режим гри проти реального супротивника. При проектуванні гри була переглянута можливість онлайн гри, для майбутньої реалізації, та алгоритмів, що лежать в основі роботи пошуку та з’єднання з іншою людиною за наявності мережі інтернет, також видозмінення дошки та створення додаткового лічильника для очок суперника.

Специфікація мережевого зв'язку:

1) Гра підтримує режим гри по мережі, де гравці можуть змагатися один з одним у реальному часі.
2) Комунікація між гравцями відбувається за допомогою мережевого протоколу, наприклад, TCP або UDP, що забезпечує передачу даних між клієнтами.

Класи та компоненти:

NetworkManager: цей клас відповідає за керування мережевим з'єднанням та обмін даними між клієнтами.

PlayerNetworkController: цей клас відповідає за обробку дій гравців у мережевому режимі гри, таких як вибір елементів та виконання рухів.

NetworkMessage: цей клас представляє повідомлення, які передаються між клієнтами для синхронізації гри та обміну даними.

Інформаційна модель управління грою виглядає наступним чином:

Під час створення гри гравці можуть вибрати режим гри по мережі та встановити підключення до сервера.

NetworkManager здійснює пошук доступних серверів або створює новий сервер для підключення гравців. Після успішного підключення гравці можуть обмінюватись повідомленнями, що включають інформацію про вибрані елементи та рухи гравців.

Кожен клієнт має свій екземпляр PlayerNetworkController, який слідкує за діями гравця та відправляє повідомлення про ці дії іншим гравцям.

Клієнти отримують повідомлення від інших гравців та оновлюють стан своєї гри відповідно до отриманих даних.

Позначення інформаційної моделі:

NetworkManager.ConnectToServer(): метод, який встановлює з'єднання з сервером для гри по мережі.

NetworkManager.StartServer(): метод, який створює новий сервер для підключення гравців.

PlayerNetworkController.SendMove(): метод, який відправляє повідомлення про рух гравця до інших клієнтів.

PlayerNetworkController.ReceiveMove(): метод, який отримує повідомлення про рух гравця від інших клієнтів та оновлює стан гри.

2.4. Розробка класів та необхідних методів

Основний публічний клас Match3 : MonoBehavior відповідає за ініціалізацію гри та управління взаємодією з грою. У класі є змінні, які відповідають за налаштування гри, такі як розмір ігрової дошки, зображення фігур і т.д.

MonoBehavior в Unity - це базовий клас, який використовується для створення компонентів, що додають функціональність до об'єктів в грі. Клас MonoBehavior є частиною Unity API і містить набір методів, які можуть бути перевизначені для виконання різних дій в різних етапах життєвого циклу об'єкта.

MonoBehavior надає можливість керувати поведінкою об'єктів, реагувати на події, виконувати анімацію, розміщувати графіку на екрані, взаємодіяти зі звуком та фізикою, та багато іншого. Клас містить такі методи, як Start(), Update(), FixedUpdate(), OnCollisionEnter(), OnTriggerEnter(), і багато інших, які можна використовувати для реалізації різних дій в грі.

Наприклад, метод Start() викликається один раз при запуску об'єкта, і в ньому можна ініціалізувати початкові значення. Метод Update() викликається кожен кадр і використовується для оновлення логіки гри.

Метод OnCollisionEnter() викликається, коли об'єкт зіткнувся з іншим об'єктом і може бути використаний для виконання певних дій при зіткненні.

Клас MonoBehavior є ключовим у розробці ігор в Unity, оскільки дозволяє реалізувати логіку та поведінку об'єктів. Він може бути успадкований і використаний для створення власних компонентів, які додають функціональність до об'єктів гри.

Також маємо перелік методів, які сприяють створенню гри:

Start() - метод, який викликається при запуску гри. Він викликає метод StartGame(), який ініціалізує гру.

Update() - метод, який викликається кожен кадр гри. Він відповідає за оновлення гри після переміщення фігури гравцем.

ApplyGravityToBoard() - метод, який викликається після того, як гравець утворив лінію з трьох або більше фігур однакового кольору. Цей метод забезпечує падіння фігур на місце, де була знищена попередня лінія, та генерацію нових фігур на верхній частині дошки.

getFlipped(NodePiece p) - метод, який повертає FlippedPieces, який відповідає за перевернуті фігури.

StartGame() - метод, який викликається при запуску гри. Цей метод ініціалізує гру та викликає методи InitializeBoard(), VerifyBoard() і InstantiateBoard().

InitializeBoard() - метод, який створює головну дошку гри та заповнює її фігурами.

VerifyBoard() - метод, який перевіряє, чи є на дошці гри лінії з трьох або більше фігур однакового кольору.

InstantiateBoard() - метод, який відповідає за створення візуального представлення головної дошки гри. Він створює NodePieces (фігури) на дошці за допомогою відповідного зображення фігури, яке зберігається в класі Match3. Кожен NodePiece містить відповідну координату на дошці та колір фігури.

3 РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Вибір інструментарію та створення проєкту

З інформації сайту [5] Unity є програмним інструментарієм для розробки ігор, який підтримує різні платформи, включаючи ПК, мобільні телефони, консолі та віртуальну реальність. Цей інструмент дозволяє розробникам створювати різноманітні ігри за допомогою інтегрованого середовища розробки, що містить різні інструменти для графіки, фізики, звуку та інших елементів гри. Серед особливостей Unity можна виділити:

- Широкі можливості: Unity підтримує різні типи ігор та проектів, включаючи 2D та 3D ігри, ігри з різними жанрами, а також AR та VR проекти.
- Легкість використання: Unity має інтуїтивно зрозумілий інтерфейс і безліч інструментів, які спрощують розробку ігор.
- Кросплатформенність: Unity дозволяє створювати ігри для різних платформ, таких як Windows, Mac, Linux, iOS, Android, Xbox, PlayStation та інші.
- Підтримка різних мов програмування: Unity підтримує кілька мов програмування, таких як C#, C++, JavaScript та інші.
- Велика спільнота: Unity має величезну спільноту розробників, де можна знайти відповіді на багато питань та підказки щодо розробки ігор.
- Asset Store: Unity Asset Store пропонує широкий вибір готових ресурсів, таких як моделі персонажів, анімації, текстури та інші, які можуть бути використані в проекті.
- Можливості розширення: Unity дозволяє створювати власні інструменти та плагіни для оптимізації процесу розробки ігор.

Через свої широкі можливості та легкість у використанні, Unity став одним з найбільш популярних ігрових движків у світі. Він має простий і зрозумілий інтерфейс, який робить його досить легким для вивчення творцями ігор, що починають.

Одна з основних причин, чому Unity є гарним для початківців, полягає в тому, що він використовує мову програмування C#. C# вважається однією з найдоступніших і найпростіших розуміння мов програмування. Завдяки цьому новачки можуть швидко навчитися створювати ігри з використанням Unity.

Крім того, Unity має велику спільноту користувачів, які надають навчальні матеріали та ресурси для вивчення різних тонкощів середовища розробки, серед яких: уроки, відеоуроки, форуми та блоги. Більшість інформації про створення ігор на C# за допомогою рушія Unity було використано з сайту [2].

Нарешті, Unity має безкоштовну версію, яку можна використовувати для навчання та створення ігор. Безкоштовна версія має всі необхідні функції для створення базових ігор і є відмінним інструментом для творців ігор, що починають.

Для початку потрібно зареєструватися на офіційному сайті розробника, завантажити Unity Hub, і в ньому вибрати версію IDE, на якій приходитиме створення гри. Після повної інсталяції і починається повна робота над створенням гри. У програмі Unity Hub створюємо новий проект, обираємо 2D вигляд.

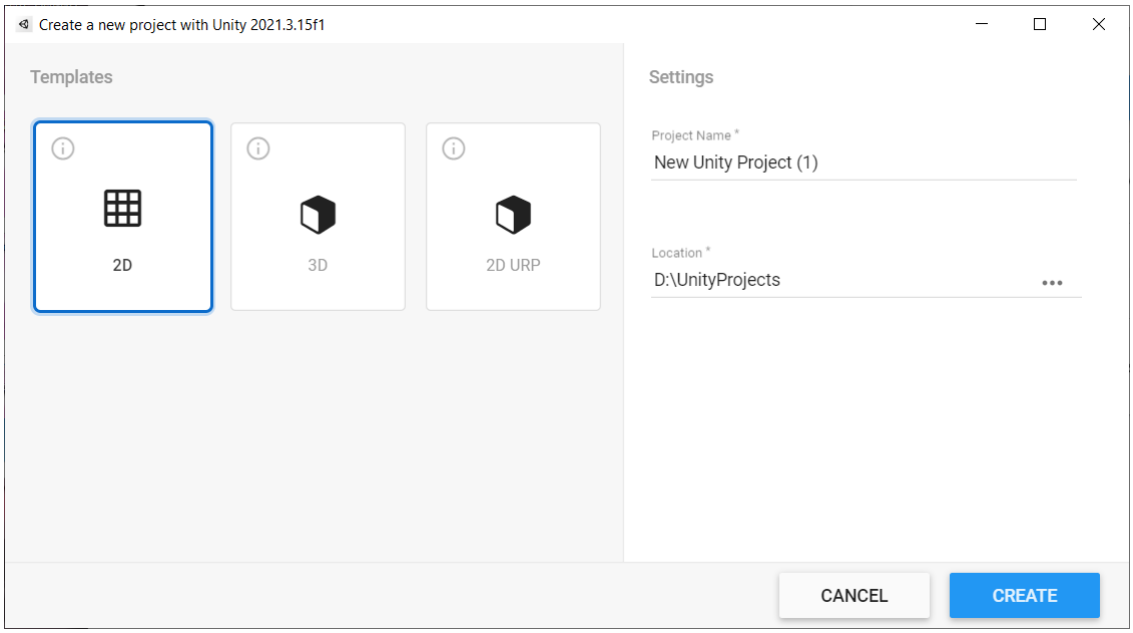


Рисунок 2-1 – Створення проекту

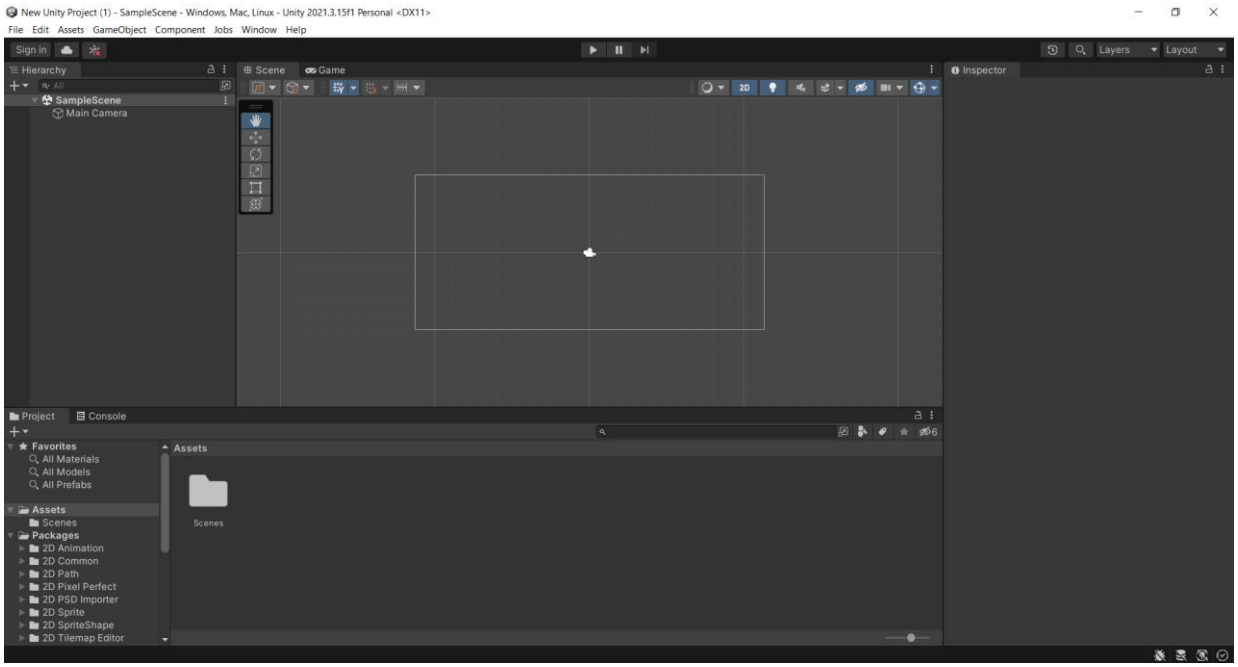


Рисунок 2-2 – Початкове вікно проекту

Отримаємо вікно у вигляді Рисунок 1.2, та починаємо підготовлювати базу для створення проекту.

Одним з основних аспектів налаштування проекту є розуміння, на яку аудиторію буде спрямована гра. У Unity для кожної платформи є власні налаштування компіляції, які визначають, як програма буде зібрана та працювати на конкретній платформі. Ось основні особливості кожної платформи Build Setting у Unity:

1. PC, Mac & Linux Standalone:

- Вибір платформи: Windows, Mac чи Linux.
- Розрядність: 32-bit чи 64-bit.
- Тип білда: Debug або Release.
- Параметри складання: опції складання, які залежать від платформи, наприклад, налаштування екрана та увімкнення/вимкнення функцій.

2. Android:

- Архітектура: вибір архітектури процесора, яка визначає, як програма буде працювати на пристроях Android.
- Тип білда: Debug або Release.
- Маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму.
- Підпис: сертифікат, який використовується для підписання програми перед завантаженням на Google Play Store.

3. iOS:

- Тип білда: Debug або Release.
- Схема складання: налаштування складання, такі як вибір цільового пристрою та увімкнення/вимкнення функцій.
- Provisioning Profile: файл, який використовується для ідентифікації та перевірки розробника та програми перед його завантаженням на пристрої iOS.
- Ключі та сертифікати: використовуються для підписання та складання програми.

4. WebGL:

- Тип білда: Debug або Release.
- Розмір буфера: розмір буфера, який використовується для завантаження та відтворення веб-сторінки з WebGL-програмою.
- Стиснення текстур: увімкнення/вимкнення стиснення текстур для зменшення розміру файлів, що завантажуються.

5. Universal Windows Platform:

- Тип білда: Debug або Release.
- Тип пакета: вибір типу пакета програми, наприклад APPX або MSIX.
- Розмір буфера: розмір буфера, який використовується для завантаження та відтворення програми.
- Маніфест: конфігураційний файл, який визначає інформацію про пристрій, який використовуватиме програму.

Усі платформи націлені на конкретну групу користувачів. Так найбільш популярними є PC, Mac, Linux, Android та IOS. В кваліфікаційній роботі розглядається платформа PC, але в подальшому випадку може бути з легкістю інтегрована в мобільні пристрої на платформах Android чи IOS.

3.2 Реалізація класів та структур гри

3.2.1 Реалізація вихідного коду гри «Match3»

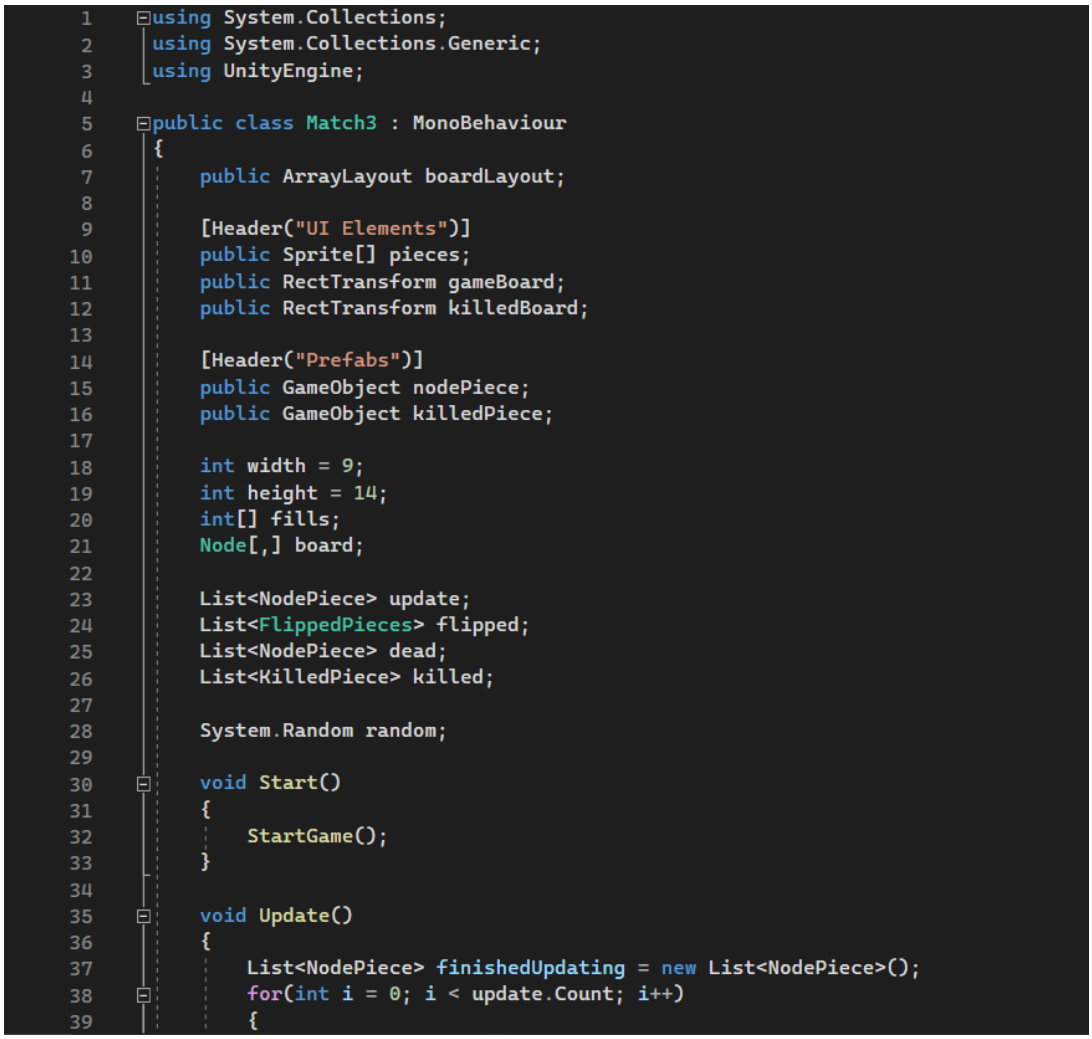
Основний код створений для гри, називається: Match3.

Код в собі містить:

- 3 класи;

- 6 змінних;

- 17 методів;



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

public class Match3 : MonoBehaviour
{
    public ArrayLayout boardLayout;

    [Header("UI Elements")]
    public Sprite[] pieces;
    public RectTransform gameBoard;
    public RectTransform killedBoard;

    [Header("Prefabs")]
    public GameObject nodePiece;
    public GameObject killedPiece;

    int width = 9;
    int height = 14;
    int[] fills;
    Node[,] board;

    List<NodePiece> update;
    List<FlippedPieces> flipped;
    List<NodePiece> dead;
    List<KilledPiece> killed;

    System.Random random;

    void Start()
    {
        StartGame();
    }

    void Update()
    {
        List<NodePiece> finishedUpdating = new List<NodePiece>();
        for(int i = 0; i < update.Count; i++)
        {
```

Рисунок ***Error! No text of specified style in document.****-3 – вихідний код Match3*

Реалізовано у вихідному коду «Match3»:

Класи:

Match3: Клас, який представляє основний контролер гри Match-3.

Node: Клас, що представляє вузол або комірку в грі, зберігає значення та посилання на об'єкт NodePiece.

FlippedPieces: Клас, який зберігає посилання на два NodePiece, що були перекинуті місцями під час гри.

Змінні класу Match3:

boardLayout: Об'єкт типу ArrayLayout, що містить інформацію про формування дошки гри.

pieces: Масив спрайтів, які представляють різні типи елементів у грі.

gameBoard: Об'єкт RectTransform, що представляє дошку гри.

killedBoard: Об'єкт RectTransform, що представляє дошку знищених елементів.

nodePiece: Префаб гри, який використовується для створення елементів на дошці гри.

killedPiece: Префаб гри, який використовується для створення знищених елементів.

Методи класу Match3:

Start(): Метод, який викликається при запуску гри.

Update(): Метод, який викликається на кожній ітерації оновлення гри.

ApplyGravityToBoard(): Метод, який застосовує гравітацію до дошки гри, зміщуючи елементи вниз.

getFlipped(): Метод, який повертає об'єкт FlippedPieces, що містить посилання на обидва NodePiece, що були перекинуті місцями.

StartGame(): Метод, який починає нову гру, ініціалізуючи всі необхідні змінні та створюючи дошку гри.

InitializeBoard(): Метод, який ініціалізує дошку гри, створюючи об'єкти типу Node для кожної комірки та встановлюючи їх значення.

VerifyBoard(): Метод, який перевіряє дошку гри, шукаючи можливі матчі-3. Цей метод перевіряє кожну комірку на дошці та перевіряє її сусідні комірки, щоб виявити можливі match-3.

RemoveMatches(): Метод, який видаляє зіткнення матчів-3 з дошки гри. Він перебирає всі комірки на дошці та перевіряє, чи є вони частиною матчу-3. Якщо так, то він видаляє елемент з комірки та додає його до знищених елементів на killedBoard.

FillBoard(): Метод, який заповнює порожні комірки на дошці новими елементами. Він перебирає кожну комірку на дошці та, якщо комірка порожня, випадковим чином вибирає новий елемент для заповнення комірки.

SwapPieces(): Метод, який обмінює два елементи місцями на дошці гри. Цей метод приймає посилання на два елементи (piece1 і piece2) і обмінює їх значення та позиції.

IsAdjacent(): Метод, який перевіряє, чи є два елементи сусідніми на дошці гри. Він порівнює позиції елементів та перевіряє, чи вони знаходяться поруч (горизонтально або вертикально).

OnPieceClicked(): Метод, який викликається при кліку на елемент на дошці гри. Він перевіряє, чи вибрано вже елемент для обміну. Якщо ні, то запам'ятовує вибраний елемент. Якщо так, то він спробує обміняти вибраний елемент з поточним клікнутим елементом.

CheckPotentialMatches(): Метод, який перевіряє, чи є можливі матчі-3 на дошці гри. Він перебирає кожну комірку на дошці та перевіряє, чи можна обміняти цю комірку зі сусідніми, щоб отримати match-3. Якщо так, то це вказує на потенційний match-3.

ClearBoard(): Метод, який очищує дошку гри. Він видаляє всі елементи з дошки та очищує знищену дошку.

RefillBoard(): Метод, який заповнює порожні комірки на дошці новими елементами. Цей метод заповнює порожні комірки на дошці новими елементами зверху та створює нові елементи для заміни попередніх.

SwapBack(): Метод, який повертає елементи на свої початкові позиції після невдалого обміну. Він обмінює місцями два елементи piece1 та piece2, щоб повернути їх на свої початкові позиції.

GetMatchCount(): Метод, який повертає кількість знайдених матчів-3 на дошці гри. Він перебирає всі комірки на дошці та перевіряє, чи вони входять в match-3.

3.2.2 Реалізація вихідного коду гри «Point»

Код в собі містить:

- 1 клас;

- 2 поля;

- 1 конструктор;

- 9 методів;

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

 [System.Serializable]
public class Point
{
    public int x;
    public int y;

    public Point(int nx, int ny)
    {
        x = nx;
        y = ny;
    }

    public void mult(int m)
    {
        x *= m;
        y *= m;
    }

    public void add(Point p)
    {
        x += p.x;
        y += p.y;
    }

    public Vector2 ToVector()
    {
        return new Vector2(x, y);
    }

    public bool Equals(Point p)
    {
        return (x == p.x && y == p.y);
    }

```

Рисунок ***Error! No text of specified style in document.****-4 – вихідний код Point*

Реалізовано у вихідному коду «Point»:

Класи:

Point: Клас, що представляє точку у двовимірному просторі. Включає в себе поля x та y, конструктор, методи маніпуляції з точками (множення, додавання), метод конвертації до типу Vector2, метод порівняння точок, а також статичні методи для створення точки з векторів та виконання математичних операцій над точками. Має також статичні поля для представлення спеціальних точок (нуль, одиниця, вгору, вниз, праворуч, ліворуч).

Поля класу Point:

x: Ціле число, що представляє координату X точки.

y: Ціле число, що представляє координату Y точки.

Конструктор класу Point:

Point(int nx, int ny): Конструктор, який приймає значення nx та ny і ініціалізує поля x та y відповідно.

Методи класу Point:

mult(int m): Метод, який множить координати точки на значення m.

add(Point p): Метод, який додає координати точки p до поточної точки.

ToVector(): Метод, який перетворює точку до типу Vector2.

Equals(Point p): Метод, який перевіряє, чи рівна поточна точка точці p.

Статичні методи класу Point:

fromVector(Vector2 v): Статичний метод, який створює нову точку з координат вектора v.

fromVector(Vector3 v): Статичний метод, який створює нову точку з першими двома координатами вектора v.

mult(Point p, int m): Статичний метод, який множить координати точки p на значення m і повертає нову точку.

add(Point p, Point o): Статичний метод, який додає координати точки p до координат точки o і повертає нову точку.

clone(Point p): Статичний метод, який створює нову точку з такими ж координатами, як точка p.

zero: Статичне поле, яке представляє точку з координатами (0, 0).

one: Статичне поле, яке представляє точку з координатами (1, 1).

up: Статичне поле, яке представляє точку з координатами (0, 1).

down: Статичне поле, яке представляє точку з координатами (0, -1).

right: Статичне поле, яке представляє точку з координатами (1, 0).

left: Статичне поле, яке представляє точку з координатами (-1, 0).

3.2.3 Реалізація вихідного коду гри «NodePiece»

Код в собі містить:

- 1 клас;
- 6 полів;
- 9 методів;

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

public class NodePiece : MonoBehaviour, IPointerDownHandler, IPointerUpHandler
{
    public int value;
    public Point index;

    [HideInInspector]
    public Vector2 pos;
    [HideInInspector]
    public RectTransform rect;

    bool updating;
    Image img;

    public void Initialize(int v, Point p, Sprite piece)
    {
        img = GetComponent<Image>();
        rect = GetComponent<RectTransform>();

        value = v;
        SetIndex(p);
        img.sprite = piece;
    }

    public void SetIndex(Point p)
    {
        index = p;
        ResetPosition();
        UpdateName();
    }
```

Рисунок ***Error! No text of specified style in document.****-5 – вихідний код NodePiece*

Реалізовано у вихідному коду «NodePiece»:

Класи:

NodePiece: Клас, що представляє елемент (блок) вузла гри. Реалізує інтерфейси IPointerDownHandler та IPointerUpHandler для обробки подій натискання та відпускання на елементі. Включає в себе поля value (значення елемента), index (індекс елемента у сітці), pos (позиція елемент у грі), rect (прямокутник, що представляє графічний об'єкт елемент), updating (прапорець, який показує, чи елемент оновлюється), а також img (графічний компонент, що відповідає за відображення зображення елемента).

Поля класу NodePiece:

value: Ціле число, що представляє значення елемента.

index: Об'єкт класу Point, що визначає індекс елемента у сітці.

pos: Вектор, що визначає позицію елемента у грі.

rect: Об'єкт класу RectTransform, що представляє графічний об'єкт елемента.

updating: Логічне значення, яке показує, чи елемента оновлюється.

img: Об'єкт класу Image, що відповідає за відображення зображення елемента.

Методи класу NodePiece:

Initialize(int v, Point p, Sprite piece): Метод, який ініціалізує елемент зі значенням v, індексом p та зображенням piece.

SetIndex(Point p): Метод, який встановлює індекс елемента на позицію p та оновлює його позицію та ім'я.

ResetPosition(): Метод, який скидає позицію елемента до початкового значення на основі індексу.

MovePosition(Vector2 move): Метод, який зміщує позицію елемента на вектор move залежно від часу.

MovePositionTo(Vector2 move): Метод, який плавно пересуває позицію елемента до вектора move залежно від часу.

UpdatePiece(): Метод, який оновлює позицію елемента до цільової позиції pos. Повертає логічне значення, яке показує, чи елемент все ще оновлюється.

UpdateName(): Метод, який оновлює ім'я елемента на основі його індексу.

OnPointerDown(): Реалізація методу з інтерфейсу IPointerDownHandler. Обробляє подію натискання на елемент.

OnPointerUp(): Реалізація методу з інтерфейсу IPointerUpHandler. Обробляє подію відпускання елемента.

Загалом, цей код визначає клас NodePiece, який представляє елемент вузла гри. Він містить різноманітні методи для ініціалізації, переміщення та оновлення елемента, а також обробки подій натискання та відпускання. Цей клас може бути використаний для реалізації гри з рухомими елементами, наприклад, гри "Пазли".

3.2.4 Реалізація вихідного коду гри «MovePieces»

Код в собі містить:

- 2 класи;
- 5 полів;
- 5 методів;
- 1 логічний блок;

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

public class MovePieces : MonoBehaviour
{
    public static MovePieces instance;
    Match3 game;

    NodePiece moving;
    Point newIndex;
    Vector2 mouseStart;

    private void Awake()
    {
        instance = this;
    }

    void Start()
    {
        game = GetComponent<Match3>();
    }

    void Update()
    {
        if(moving != null)
        {
            Vector2 dir = ((Vector2)Input.mousePosition - mouseStart);
            Vector2 nDir = dir.normalized;
            Vector2 aDir = new Vector2(Mathf.Abs(dir.x), Mathf.Abs(dir.y));
```

Рисунок ***Error! No text of specified style in document.****-4 – Вихідний код MovePieces*

Реалізовано у вихідному коду «MovePieces»:

Класи:

MovePieces: Клас, що відповідає за переміщення елементів в грі. Містить посилання на інстанс гри (game), зберігає інформацію про переміщуваний елемент (moving), новий індекс елемента (newIndex) та початкову позицію миші (mouseStart). Реалізується як компонент Unity.

Match3: Інший клас, який відповідає за логіку гри типу "Match-3". Не включений у наведений код, але згадується як змінна game.

Поля класу MovePieces:

instance: Статичний екземпляр класу MovePieces для доступу до нього з інших класів.

game: Змінна типу Match3 для зберігання посилання на екземпляр класу Match3.

moving: Об'єкт класу NodePiece, який представляє переміщуваний елемент.

newIndex: Об'єкт класу Point, який містить новий індекс елемента.

mouseStart: Вектор, що містить початкову позицію миші.

Методи класу MovePieces:

Awake(): Метод, який викликається при створенні об'єкту класу. Встановлює посилання на поточний екземпляр класу у статичну змінну instance.

Start(): Метод, який викликається перед першим оновленням кадру. Ініціалізує змінну game шляхом отримання компонента Match3 з об'єкту.

Update(): Метод, який викликається на кожному кадрі. Відповідає за переміщення елемента відповідно до руху миші.

MovePiece(): Метод, який викликається для початку переміщення елемента. Встановлює переміщуваний елемент в змінну moving та зберігає початкову позицію миші у mouseStart.

DropPiece(): Метод, який викликається при закінченні переміщення елемента. Перевіряє, чи було змінено індекс елемента, і викликає відповідні методи гри для здійснення обміну або скидання елемента.

Внутрішній логічний блок коду:

В циклі Update перевіряється, чи існує переміщуваний елемент (moving). Якщо елемент переміщується, виконується обчислення нового індексу елемента в залежності від руху миші. Потім елемент плавно пересувається до нової позиції за допомогою методу MovePositionTo.

3.2.5 Реалізація вихідного коду гри «KilledPiece»

Код в собі містить:

- 1 класи;
- 6 полів;
- 2 методів;
- 1 логічний блок;

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

public class KilledPiece : MonoBehaviour
{
    public bool falling;

    float speed = 16f;
    float gravity = 32f;
    Vector2 moveDir;
    RectTransform rect;
    Image img;

    public void Initialize(Sprite piece, Vector2 start)
    {
        falling = true;

        moveDir = Vector2.up;
        moveDir.x = Random.Range(-1.0f, 1.0f);
        moveDir *= speed / 2;

        img = GetComponent<Image>();
        rect = GetComponent<RectTransform>();
        img.sprite = piece;
        rect.anchoredPosition = start;
    }
```

Рисунок ***Error! No text of specified style in document.****-5 – Вихідний код KilledPiece*

Реалізовано у вихідному коду «KilledPiece»:

Класи:

KilledPiece: Клас, що представляє знищений елемент. Містить поля для контролю падіння елемента (falling), швидкості (speed), гравітації (gravity), напрямку руху (moveDir), посилання на компонент RectTransform (rect) та Image (img). Реалізується як компонент Unity.

Поля класу KilledPiece:

falling: Змінна типу bool, що вказує, чи знаходиться елемент у процесі падіння.

speed: Швидкість руху елемента.

gravity: Значення гравітації, що впливає на елемент під час падіння.

moveDir: Вектор, який визначає напрямок руху елемента.

rect: Посилання на компонент RectTransform елемента.

img: Посилання на компонент Image елемента.

Методи класу KilledPiece:

Initialize(): Метод, який ініціалізує знищений елемент. Встановлює значення falling у true, генерує випадковий напрямок руху елемента, ініціалізує посилання на компоненти RectTransform і Image, встановлює спрайт елемента та початкову позицію.

Update(): Метод, який викликається на кожному кадрі. Оновлює рух знищеного елемента відповідно до гравітації та швидкості. Перевіряє, чи елемент вийшов за межі екрану, і встановлює значення falling у false, якщо це сталося.

Внутрішній логічний блок коду:

В циклі Update перевіряється, чи елемент знаходиться у стані падіння (falling). Якщо це так, обчислюється новий вектор руху елемента з урахуванням гравітації та затухання швидкості по осі X. Зміщення елемента обчислюється шляхом множення вектора руху на час та швидкість. Потім перевіряється, чи вийшов елемент за межі екрану, і якщо так, то значення falling встановлюється у false, що зупиняє оновлення руху елемента.

3.3 Розробка інструкції щодо впровадження та експлуатації

1. Впровадження гри:

- ознайомитись з вимогами до системи та платформи, на яку планується впровадження гри. Переконайтеся, що ваша гра відповідає цим вимогам;

- зібрати всі необхідні файли гри, включаючи виконуваний файл, ресурси, звукові ефекти, графічні зображення тощо;

- перевірити правильність розташування файлів та шляхів до них, щоб усі елементи гри відображалися коректно.

2. Впровадження на різних платформах:

- для мобільних платформ (Android, iOS) скомпілювати гру у відповідних форматах (APK, IPA) та перевірте її на сумісність з різними пристроями та операційними системами;

- для платформи ПК (Windows, Mac) згенерувати виконуваний файл гри (.exe, .app) та перевірити його на різних комп'ютерах з різними конфігураціями.

3. Тестування та налагодження:

- перед впровадженням гри здійснити тестування на різних платформах та пристроях, щоб виявити й виправити можливі помилки та проблеми з продуктивністю;

- переконатися, що всі функції гри працюють належним чином та гра має зручний та інтуїтивно зрозумілий інтерфейс.

4. Розповсюдження гри:

- визначити методи розповсюдження гри, такі як платні або безкоштовні веб-магазини, цифрові платформи, свої веб-сайти тощо;

- при необхідності розробити маркетингову стратегію для просування гри та залучення аудиторії.

5. Підтримка та оновлення:

- забезпечити можливість зворотного зв'язку користувачів гри для отримання повідомлень про проблеми та пропозиції щодо поліпшень;

- регулярно оновлювати гру, виправляючи помилки, додаючи нові рівні або функції, що розширюють геймплей.

6. Забезпечення безпеки:

- захистити інтелектуальну власність, включаючи авторські права на гру та використання ліцензійних матеріалів;

- застосувати заходи для запобігання піратства та незаконного використання гри.

ВИСНОВКИ

У роботі було розроблено комп’ютерну гру у жанрі “Три у ряд” на Unity. У процесі розробки були розглянуті основні концепції та інструменти, що використовуються під час створення ігор на Unity, такі як створення ігрового об'єкта, додавання компонентів, налаштування фізики, використання анімацій, звукових ефектів, ефектів візуалізації, використання скриптів на мові C# та багато інших. Для розробки гри була створена 2D сцена з використанням спрайтів та колайдерів, були налаштовані фізичні властивості об'єктів, додані скрипти мовою C.

1) Створено проект на Unity та налаштування дошки.

2) За допомогую графічного редактору AdobePhotoshop створені спрайти для ключових фігур на дошці.

3) Створені та налаштувані бонуси, перешкоди та інші ігрові елементи.

4) Розроблено ігрові механіки та ігрові рівні.

5) Реалізовано скрипти для створення, взаємодії та знищення об’ктів.

6) Протестовано та налагоджено гру.

Були розглянуті також деякі концепції, що використовуються в іграх «Три у ряд», такі як генерація рівнів, набір очок. Створення «Три у ряд» гри на Unity - це цікавий і захоплюючий процес, який дозволяє отримати досвід у роботі з ігровими концепціями та інструментами, а також навчитися використовувати різні аспекти розробки ігор, такі як анімації, звукові та візуальні ефекти, управління ігровими об'єктами та багато інші.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний форум від Unity для розробника ігор на їх рушію forum.unity.com [Електронний ресурс](дата відвідування 06.05.2023) режим доступу: forum.unity.com

2. Відео-уроки онлайн з програмування на мові C# та створення ігор на рушії Unity [Електронний ресурс](дата відвідування 12.05.2023) режим доступу: https://itproger.com/ua/course/unity

3. Зальцман Марк Комп’ютерні ігри: як це робиться - Логрус. 2000.

4. Інтернет-магазин «Google Play Store» [Електронний ресурс]
(дата відвідування 13.05.2023) режим доступу:
https://play.google.com/store/games

5. Unity Documentation - Офіційна документація Unity, що містить інформацію про розробку ігор на рушії Unity [Електронний ресурс] (дата відвідування 22.05.2023) режим доступу: https://docs.unity3d.com

ДОДАТОК А ВИХІДНИЙ КОД «MATCH3»

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

public class Match3 : MonoBehaviour
{
    public ArrayLayout boardLayout;

    [Header("UI Elements")]
    public Sprite[] pieces;
    public RectTransform gameBoard;
    public RectTransform killedBoard;

    [Header("Prefabs")]
    public GameObject nodePiece;
    public GameObject killedPiece;

    int width = 9;
    int height = 14;
    int[] fills;
    Node[,] board;

    List<NodePiece> update;
    List<FlippedPieces> flipped;
    List<NodePiece> dead;
    List<KilledPiece> killed;

    System.Random random;

    void Start()
    {
        StartGame();
    }

    void Update()
    {
        List<NodePiece> finishedUpdating = new List<NodePiece>();
        for(int i = 0; i < update.Count; i++)
        {
            NodePiece piece = update[i];
```

```
    if (!piece.UpdatePiece()) finishedUpdating.Add(piece);
}
for (int i = 0; i < finishedUpdating.Count; i++)
{
    NodePiece piece = finishedUpdating[i];
    FlippedPieces flip = getFlipped(piece);
    NodePiece flippedPiece = null;

    int x = (int)piece.index.x;
    fills[x] = Mathf.Clamp(fills[x] - 1, 0, width);

    List<Point> connected = isConnected(piece.index, true);
    bool wasFlipped = (flip != null);

    if (wasFlipped)
    {
        flippedPiece = flip.getOtherPiece(piece);
        AddPoints(ref connected, isConnected(flippedPiece.index, true));
    }

    if (connected.Count == 0)
    {
        if (wasFlipped)
            FlipPieces(piece.index, flippedPiece.index, false);
    }
    else
    {
        foreach (Point pnt in connected)
        {
            KillPiece(pnt);
            Node node = getNodeAtPoint(pnt);
            NodePiece nodePiece = node.getPiece();
            if (nodePiece != null)
            {
                nodePiece.gameObject.SetActive(false);
                dead.Add(nodePiece);
            }
            node.SetPiece(null);
        }

        ApplyGravityToBoard();
    }

    flipped.Remove(flip);
    update.Remove(piece);
```

```
    }
}

public void ApplyGravityToBoard()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = (height - 1); y >= 0; y--)
        {
            Point p = new Point(x, y);
            Node node = getNodeAtPoint(p);
            int val = getValueAtPoint(p);
            if (val != 0) continue;
            for (int ny = (y - 1); ny >= -1; ny--)
            {
                Point next = new Point(x, ny);
                int nextVal = getValueAtPoint(next);
                if (nextVal == 0)
                    continue;
                if (nextVal != -1)
                {
                    Node gotten = getNodeAtPoint(next);
                    NodePiece piece = gotten.getPiece();

                    node.SetPiece(piece);
                    update.Add(piece);

                    gotten.SetPiece(null);
                }
                else
                {
                    int newVal = fillPiece();
                    NodePiece piece;
                    Point fallPnt = new Point(x, (-1 - fills[x]));
                    if(dead.Count > 0)
                    {
                        NodePiece revived = dead[0];
                        revived.gameObject.SetActive(true);
                        piece = revived;

                        dead.RemoveAt(0);
                    }
                    else
                    {
                        GameObject obj = Instantiate(nodePiece, gameBoard);
```

```
                NodePiece n = obj.GetComponent<NodePiece>();
                piece = n;
            }

            piece.Initialize(newVal, p, pieces[newVal - 1]);
            piece.rect.anchoredPosition = getPositionFromPoint(fallPnt);

            Node hole = getNodeAtPoint(p);
            hole.SetPiece(piece);
            ResetPiece(piece);
            fills[x]++;
          }
          break;
        }
      }
    }
}

FlippedPieces getFlipped(NodePiece p)
{
    FlippedPieces flip = null;
    for (int i = 0; i < flipped.Count; i++)
    {
      if (flipped[i].getOtherPiece(p) != null)
      {
        flip = flipped[i];
        break;
      }
    }
    return flip;
}

void StartGame()
{
    fills = new int[width];
    string seed = getRandomSeed();
    random = new System.Random(seed.GetHashCode());
    update = new List<NodePiece>();
    flipped = new List<FlippedPieces>();
    dead = new List<NodePiece>();
    killed = new List<KilledPiece>();

    InitializeBoard();
    VerifyBoard();
    InstantiateBoard();
```

```
    }

    void InitializeBoard()
    {
        board = new Node[width, height];
        for(int y = 0; y < height; y++)
        {
            for(int x = 0; x < width; x++)
            {
                board[x, y] = new Node((boardLayout.rows[y].row[x]) ? - 1 : fillPiece(),
new Point(x, y));
            }
        }
    }

    void VerifyBoard()
    {
        List<int> remove;
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                Point p = new Point(x, y);
                int val = getValueAtPoint(p);
                if (val <= 0) continue;

                remove = new List<int>();
                while (isConnected(p, true).Count > 0)
                {
                    val = getValueAtPoint(p);
                    if (!remove.Contains(val))
                        remove.Add(val);
                    setValueAtPoint(p, newValue(ref remove));
                }
            }
        }
    }

    void InstantiateBoard()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                Node node = getNodeAtPoint(new Point(x, y));
```

```
            int val = node.value;
            if (val <= 0) continue;
            GameObject p = Instantiate(nodePiece, gameBoard);
            NodePiece piece = p.GetComponent<NodePiece>();
            RectTransform rect = p.GetComponent<RectTransform>();
            rect.anchoredPosition = new Vector2(32 + (64 * x), -32 - (64 * y));
            piece.Initialize(val, new Point(x, y), pieces[val - 1]);
            node.SetPiece(piece);
        }
    }
}

public void ResetPiece(NodePiece piece)
{
    piece.ResetPosition();
    update.Add(piece);
}

public void FlipPieces(Point one, Point two, bool main)
{
    if (getValueAtPoint(one) < 0) return;

    Node nodeOne = getNodeAtPoint(one);
    NodePiece pieceOne = nodeOne.getPiece();
    if (getValueAtPoint(two) > 0)
    {
        Node nodeTwo = getNodeAtPoint(two);
        NodePiece pieceTwo = nodeTwo.getPiece();
        nodeOne.SetPiece(pieceTwo);
        nodeTwo.SetPiece(pieceOne);

        if(main)
            flipped.Add(new FlippedPieces(pieceOne, pieceTwo));

        update.Add(pieceOne);
        update.Add(pieceTwo);
    }
    else
        ResetPiece(pieceOne);
}

void KillPiece(Point p)
{
    List<KilledPiece> available = new List<KilledPiece>();
```

```
    for (int i = 0; i < killed.Count; i++)
        if (!killed[i].falling) available.Add(killed[i]);

    KilledPiece set = null;
    if (available.Count > 0)
        set = available[0];
    else
    {
        GameObject kill = GameObject.Instantiate(killedPiece, killedBoard);
        KilledPiece kPiece = kill.GetComponent<KilledPiece>();
        set = kPiece;
        killed.Add(kPiece);
    }

    int val = getValueAtPoint(p) - 1;
    if (set != null && val >= 0 && val < pieces.Length)
        set.Initialize(pieces[val], getPositionFromPoint(p));
}

List<Point> isConnected(Point p, bool main)
{
    List<Point> connected = new List<Point>();
    int val = getValueAtPoint(p);
    Point[] directions =
    {
        Point.up,
        Point.right,
        Point.down,
        Point.left
    };

    foreach(Point dir in directions)
    {
        List<Point> line = new List<Point>();

        int same = 0;
        for(int i = 1; i < 3; i++)
        {
            Point check = Point.add(p, Point.mult(dir, i));
            if(getValueAtPoint(check) == val)
            {
                line.Add(check);
                same++;
            }
        }
```

```
            if (same > 1)
                AddPoints(ref connected, line);
        }

        for(int i = 0; i < 2; i++)
        {
            List<Point> line = new List<Point>();

            int same = 0;
            Point[] check = { Point.add(p, directions[i]), Point.add(p, directions[i + 2])
};
            foreach (Point next in check)
            {
                if (getValueAtPoint(next) == val)
                {
                    line.Add(next);
                    same++;
                }
            }

            if (same > 1)
                AddPoints(ref connected, line);
        }

        for(int i = 0; i < 4; i++)
        {
            List<Point> square = new List<Point>();

            int same = 0;
            int next = i + 1;
            if (next >= 4)
                next -= 4;

            Point[] check = { Point.add(p, directions[i]), Point.add(p, directions[next]),
Point.add(p, Point.add(directions[i], directions[next])) };
            foreach (Point pnt in check)
            {
                if (getValueAtPoint(pnt) == val)
                {
                    square.Add(pnt);
                    same++;
                }
            }
```

```
        if (same > 2)
            AddPoints(ref connected, square);
    }

    if(main)
    {
        for (int i = 0; i < connected.Count; i++)
            AddPoints(ref connected, isConnected(connected[i], false));
    }

    return connected;
}

void AddPoints(ref List<Point> points, List<Point> add)
{
    foreach(Point p in add)
    {
        bool doAdd = true;
        for(int i = 0; i < points.Count; i++)
        {
            if(points[i].Equals(p))
            {
                doAdd = false;
                break;
            }
        }

        if (doAdd) points.Add(p);
    }
}

int fillPiece()
{
    int val = 1;
    val = (random.Next(0, 100) / (100 / pieces.Length)) + 1;
    return val;
}

int getValueAtPoint(Point p)
{
    if (p.x < 0 || p.x >= width || p.y < 0 || p.y >= height) return -1;
    return board[p.x, p.y].value;
}

void setValueAtPoint(Point p, int v)
```

```
    {
        board[p.x, p.y].value = v;
    }

    Node getNodeAtPoint(Point p)
    {
        return board[p.x, p.y];
    }

    int newValue(ref List<int> remove)
    {
        List<int> available = new List<int>();
        for (int i = 0; i < pieces.Length; i++)
            available.Add(i + 1);
        foreach (int i in remove)
            available.Remove(i);

        if (available.Count <= 0) return 0;
        return available[random.Next(0, available.Count)];
    }

    string getRandomSeed()
    {
        string seed = "";
        string acceptableChars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdeghijklmnopqrstuvwxyz12345678
90!@#$%^&*()";
        for (int i = 0; i < 20; i++)
            seed += acceptableChars[Random.Range(0, acceptableChars.Length)];
        return seed;
    }

    public Vector2 getPositionFromPoint(Point p)
    {
        return new Vector2(32 + (64 * p.x), -32 - (64 * p.y));
    }
}

[System.Serializable]
public class Node
{
    public int value;
    public Point index;
    NodePiece piece;
```

```
    public Node(int v, Point i)
    {
        value = v;
        index = i;
    }

    public void SetPiece(NodePiece p)
    {
        piece = p;
        value = (piece == null) ? 0 : piece.value;
        if (piece == null) return;
        piece.SetIndex(index);
    }

    public NodePiece getPiece()
    {
        return piece;
    }
}

[System.Serializable]
public class FlippedPieces
{
    public NodePiece one;
    public NodePiece two;

    public FlippedPieces(NodePiece o, NodePiece t)
    {
        one = o; two = t;
    }

    public NodePiece getOtherPiece(NodePiece p)
    {
        if (p == one)
            return two;
        else if (p == two)
            return one;
        else
            return null;
    }
}
```

ДОДАТОК Б ВИХІДНИЙ КОД «MOVEPIECES»

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

public class MovePieces : MonoBehaviour
{
   public static MovePieces instance;
   Match3 game;

   NodePiece moving;
   Point newIndex;
   Vector2 mouseStart;

   private void Awake()
   {
      instance = this;
   }

   void Start()
   {
      game = GetComponent<Match3>();
   }

   void Update()
   {
      if(moving != null)
      {
         Vector2 dir = ((Vector2)Input.mousePosition - mouseStart);
         Vector2 nDir = dir.normalized;
         Vector2 aDir = new Vector2(Mathf.Abs(dir.x), Mathf.Abs(dir.y));

         newIndex = Point.clone(moving.index);
         Point add = Point.zero;
         if (dir.magnitude > 32) //If our mouse is 32 pixels away from the starting
point of the mouse
         {
            //make add either (1, 0) | (-1, 0) | (0, 1) | (0, -1) depending on the
direction of the mouse point
            if (aDir.x > aDir.y)
               add = (new Point((nDir.x > 0) ? 1 : -1, 0));
```

```
            else if(aDir.y > aDir.x)
                add = (new Point(0, (nDir.y > 0) ? -1 : 1));
        }
        newIndex.add(add);

        Vector2 pos = game.getPositionFromPoint(moving.index);
        if (!newIndex.Equals(moving.index))
            pos += Point.mult(new Point(add.x, -add.y), 16).ToVector();
        moving.MovePositionTo(pos);
    }
  }

  public void MovePiece(NodePiece piece)
  {
    if (moving != null) return;
    moving = piece;
    mouseStart = Input.mousePosition;
  }

  public void DropPiece()
  {
    if (moving == null) return;
    Debug.Log("Dropped");
    if (!newIndex.Equals(moving.index))
      game.FlipPieces(moving.index, newIndex, true);
    else
      game.ResetPiece(moving);
    moving = null;
  }
}
```

ДОДАТОК В ВИХІДНИЙ КОД «NODEPIECE»

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

public class NodePiece : MonoBehaviour, IPointerDownHandler,
IPointerUpHandler
{
   public int value;
   public Point index;

   [HideInInspector]
   public Vector2 pos;
   [HideInInspector]
   public RectTransform rect;

   bool updating;
   Image img;

   public void Initialize(int v, Point p, Sprite piece)
   {
      img = GetComponent<Image>();
      rect = GetComponent<RectTransform>();

      value = v;
      SetIndex(p);
      img.sprite = piece;
   }

   public void SetIndex(Point p)
   {
      index = p;
      ResetPosition();
      UpdateName();
   }

   public void ResetPosition()
   {
      pos = new Vector2(32 + (64 * index.x), -32 - (64 * index.y));
```

```
    }

    public void MovePosition(Vector2 move)
    {
        rect.anchoredPosition += move * Time.deltaTime * 16f;
    }

    public void MovePositionTo(Vector2 move)
    {
        rect.anchoredPosition = Vector2.Lerp(rect.anchoredPosition, move,
Time.deltaTime * 16f);
    }

    public bool UpdatePiece()
    {
        if(Vector3.Distance(rect.anchoredPosition, pos) > 1)
        {
            MovePositionTo(pos);
            updating = true;
            return true;
        }
        else
        {
            rect.anchoredPosition = pos;
            updating = false;
            return false;
        }
    }

    void UpdateName()
    {
        transform.name = "Node [" + index.x + ", " + index.y + "]";
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        if (updating) return;
        MovePieces.instance.MovePiece(this);
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        MovePieces.instance.DropPiece();
    }
}
```

ДОДАТОК Г ВИХІДНИЙ КОД «POINT»

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

[System.Serializable]
public class Point
{
    public int x;
    public int y;

    public Point(int nx, int ny)
    {
        x = nx;
        y = ny;
    }

    public void mult(int m)
    {
        x *= m;
        y *= m;
    }

    public void add(Point p)
    {
        x += p.x;
        y += p.y;
    }

    public Vector2 ToVector()
    {
        return new Vector2(x, y);
    }

    public bool Equals(Point p)
    {
        return (x == p.x && y == p.y);
    }

    public static Point fromVector(Vector2 v)
    {
```

```
    return new Point((int)v.x, (int)v.y);
}

public static Point fromVector(Vector3 v)
{
    return new Point((int)v.x, (int)v.y);
}

public static Point mult(Point p, int m)
{
    return new Point(p.x * m, p.y * m);
}

public static Point add(Point p, Point o)
{
    return new Point(p.x + o.x, p.y + o.y);
}

public static Point clone(Point p)
{
    return new Point(p.x, p.y);
}

public static Point zero
{
    get { return new Point(0, 0); }
}
public static Point one
{
    get { return new Point(1, 1); }
}
public static Point up
{
    get { return new Point(0, 1); }
}
public static Point down
{
    get { return new Point(0, -1); }
}
public static Point right
{
    get { return new Point(1, 0); }
}
public static Point left
{
```

```
        get { return new Point(-1, 0); }
    }
}
```

ДОДАТОК Д ВИХІДНИЙ КОД «POINT»

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

public class KilledPiece : MonoBehaviour
{
    public bool falling;

    float speed = 16f;
    float gravity = 32f;
    Vector2 moveDir;
    RectTransform rect;
    Image img;

    public void Initialize(Sprite piece, Vector2 start)
    {
        falling = true;

        moveDir = Vector2.up;
        moveDir.x = Random.Range(-1.0f, 1.0f);
        moveDir *= speed / 2;

        img = GetComponent<Image>();
        rect = GetComponent<RectTransform>();
        img.sprite = piece;
        rect.anchoredPosition = start;
    }

    // Update is called once per frame
    void Update()
    {
        if (!falling) return;
        moveDir.y -= Time.deltaTime * gravity;
```

```
        moveDir.x = Mathf.Lerp(moveDir.x, 0, Time.deltaTime);
        rect.anchoredPosition += moveDir * Time.deltaTime * speed;
        if (rect.position.x < -64f || rect.position.x > Screen.width + 64f ||
rect.position.y < -64f || rect.position.y > Screen.height + 64f)
            falling = false;
    }
}
```

ДОДАТОК Е ВИХІДНИЙ КОД «ARRAYLAYOUT»

```
using UnityEngine;
using System.Collections;

[System.Serializable]
public class ArrayLayout  {

        [System.Serializable]
        public struct rowData{
                public bool[] row;
        }

    public Grid grid;
    public rowData[] rows = new rowData[14]; //Grid of 7x7
}
```