ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд»**»

Виконав: студент 4 курсу, групи ПД-42
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Іванюк В.О.
(прізвище та ініціали)

Керівник Коба А.Б.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____О.В. Негоденко

«____»_____2022 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Іванюк Валерії Олександрівни
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд»»

Керівник роботи: Коба А.Б., старший викладач
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «18» лютого 2022 року №22.
2. Строк подання студентом роботи «3» червня 2022 року
3. Вхідні дані до роботи
Методи розробки мобільних ігор;
Науково-технічна література з питань, пов'язаних з розробкою мобільних додатків;

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1 Проаналізувати предметну область.
4.2 Проаналізувати ігрові рушії.
4.3 Встановити вимоги для розробленого додатку.
4.4 Здійснити програмну реалізацію ігрового проекту.
5. Перелік демонстраційного матеріалу (назва основних слайдів)
1. Титульний

2. Мета, об’єкт, предмет та наукова новизна дослідження
3. Актуальність роботи
4. Аналіз аналогів
5. Технічне завдання
6. Програмні засоби реалізації
7. Схема роботи гри
8. Основна структура проекту
9. Реалізовані класи
10. Реалізовані методи
11. Тестування
12. Процес роботи гри
13. Апробація результатів
14. Висновки
15. Кінцевий

6. Дата видачі завдання «11»квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітки
1	Підбір науково-технічної літератури	11.04.2022	
2	Дослідження аналогів та актуальності	13.04.2022	
3	Аналіз та вибір інструментів для розробки додатку	14.04.2022	
4	Проектування та реалізація	02.05.2022	
5	Вступ, висновки, реферат	17.05.2022	
6	Розробка обов’язкових демонстраційних результатів	18.05.2022	
7	Попередній захист робіт	29.05.2022	
8	Здача роботи	3.06.2022	

Студент _____ Іванюк В.О.
(підпис) (прізвище та ініціали)

Керівник роботи_____ Коба А.Б.

РЕФЕРАТ

Текстова частина бакалаврської роботи : 63 с., 2 табл., 25 рис., 2 дод., 28 джерел.

ТЕХНОЛОГІЯ UNITY, ІГРВИЙ РУШІЙ, ГРА, СЕРЕДОВИЩЕ РОЗРОБКИ, ТРИ В РЯД, ЖАНР, РОЗРОБКА, МЕХАНІКА, СЦЕНА, СКРИПТ, ОПЕРАЦІЙНА СИСТЕМА.

Об`єкт дослідження – процес гри жанру «Три в ряд».

Предмет дослідження – методи та засоби розробки ігор.

Мета роботи – розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд».

Методи дослідження — методи та засоби розробки комп'ютерних ігор. Проведено дослідження в предметній області ігор в жанрі «три в ряд», зроблено аналіз існуючих аналогів. Розглянуто та досліджено ігрові рушії для розробки ігор. На основі результатів виконаних досліджень розроблено мобільну гру для ОС IOS на Unity жанру «Три в ряд» з допомогою рушія Unity та середовища Visual Studio на мові програмування C#.

Проводячі аналізи існуючих ігр жанру "три в ряд", була з'ясовано, що багато користувачів не задовольняють існуючі ігри. По-перше, користувачів не задовольняє нав'язливий донат в іграх, деякі рівні неможливо пройти самостійно, не використовуючи додаткові елементи гри, які потрібно купувати за реальні кошти. По-друге, користувачам не подобається очікування відновлення життів в грі. Більшість з них грають в ігри даного жанру для того, щоб "вбити" свій час, тому час очікування змушує їх нервувати. Виникають складнощі і з пошуком простої гри "жанру три в ряд". Так як данні ігри завантажують дорослі для своїх дітей, щоб відволікти їх, наприклад, у черзі в лікарні. Тому гра повинна бути нескладної, щоб зосередити увагу дитини.

Цільова аудиторія даної гри діти до 12 років. Не зважаючи на те, що дана гра була розроблена для дітей вона буде користуватися популярністю і у дорослих.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об'єкта розробки. На сьогоднішній день багато людей полюбляють встановлювати ігри на свої смартфони, особливо на смартфони, що використовують операційну систему IOS. З розповсюдженням смартфонів, кількість ігор стрімко зростає, Ігри в жанрі «три в ряд» набирають оборотів і становляться популярними в наш час.

Об'єкт дослідження – процес гри жанру «Три в ряд».

Предмет дослідження – методи та засоби розробки ігор.

Мета дослідження – розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд».

Ринок комп'ютерної індустрії є самою масштабною частиною світового ринку.

2021 рік став найбільшим в історії з продажів комп'ютерних ігор. Ігровий ринок досяг $ 106,6 млрд і аудиторії в 2,2 млрд геймерів по всьому світу.

Одним з найбільших сегментів даного ринку є ніші багатокористувальних ігор, представлених в великій кількості різних жанрів. Серед цих жанрів перспективним напрямом жанру «Три в ряд», особливо на платформи IOS.

Потенціал ринку і наявність попиту спонукає початківців програмістів виявити себе в області розробки гри.

Для досягнення цієї мети потрібно:

- провести аналіз предметної області;
- провести аналіз ігрових рушіїв;
- освоїти інструментальні засоби;
- встановити вимоги до розробленого додатку;
- здійснити програмну реалізацію ігрового проекту.

Наукова новизна проекту – створення унікального додатку для операційної системи IOS з використанням рушія Unity та середовища Visual Studio на мові програмування C#.

У дипломному проекті був проведений аналіз додатків-аналогів, та виявлені переваги та недоліки, також додатково був проведений аналіз ігрових рушіїв, що дозволило в свою чергу переконатись у правильності вибору.

Додаток було розроблено на мові програмування C# з використанням .NET Framework у середовищі розробки Visual Studio 2019 для платформи IOS. Для розробки графіки та механіки гри було використану програму Unity 5.6. Музичний супровід: звуки та музика були взяті із різних джерел інтернету. Проводячи аналіз зацікавленості людей що люблять грати в ігри, було встановлено, що більшість людей надає перевагу розважальним та не складним іграм, які дозволяють розслабитись після роботи. Також багато людей більш мотивовані встановленням рекордів у грі, для того, щоб порівняти результати своєї гри з інші люди.

Додаток може використовувались на платформах IOS, що дозволить людям розслабитись після виснажливого робочого дня, а також принести позитивні емоції.

1 ВИВЧЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Ігри в жанрі match 3

Інформаційні технології (ІТ) зростали та розвивалися за останні 50 років; Неможливо придумати та планувати проект, бізнес чи іншу ініціативу без використання цієї технології. Коли говорити про інформаційні технології, то мають на увазі не лише персональні комп'ютери чи смартфони, а й сучасне обладнання на фабриках, автомобільній промисловості, авіаційній промисловості, різноманітну побутову техніку, ігри тощо, це так чи інакше не тільки полегшило наше повсякденне життя, але й також зменшується вартість та час загалом.

На сьогоднішній день існує дуже багато ігрових жанрів. Найбільш популярний із них - «три в ряд».

Ігри «три в ряд» — це тип казуальних ігор-головоломок [1]. Основне завдання полягає у формуванні ліній/ланцюжків/груп із 3 або більше однакових елементів. Традиційно дошка має квадратний малюнок і заповнена різними плитками, які потрібно переміщати, вибирати або обертати (рис. 1.1). Основним завданням є визначення та розташування візерунків на хаотичній дошці.

Рисунок 1.1 – Традиційний інтерфейс гри в жанрі «три в ряд»

Ігри жанру "три в ряд" охоплюють широкий спектр елементів дизайну, механіки та ігрового процесу. Вони включають суто покрокові ігри, але також можуть мати елементи екшену в аркадному стилі, такі як тиск часу, стрільба або координація рук і очей. Механіка підбору плиток також іноді вводиться як «міні-гра» в деякі більші, складніші ігри. Це додає гравцям відчути драйв і механіку гри.

Витоки ігор Match 3, які так полюбляють сьогодні, можна простежити до Tetris російського розробника Олексія Пажитнова в 1984 році і Chain Shot японського виробника ігор Куніакі Морібе у 1985 році [2]. Перша гра «Матч 3», випущена для DOS у 1994 році, дала початок шутерам з бульбашками. Ігри Match 3 стали популярними протягом 2000-х років у формі казуальних ігор, які розповсюджувалися або в які грали через Інтернет, зокрема серія ігор Bejeweled. Відтоді вони залишаються популярними, і гра Candy Crush Saga стала однією з найпопулярніших казуальних ігор у всьому світі.

Перелік особливостей, що роблять ігри в жанрі "три в ряд" такими популярними:

- Рівні. Найпозитивнішими емоціями є завдання подолати рівні та відчуття виконаного завдання.
- Візуальні зображення – Багато гравців не заперечують, що деякі ігри присвячені одному і тому ж сюжету. Вони насолоджуються відчуттям, яке вони мають від гарного мистецтва та цікавих рівнів.
- Історія – зараз є багато ігор, які додають шар історії поверх відповідної механіки. Наприклад, Gardenscapes є чудовим прикладом.
- Мета-гра – деякі ігри використовують механіку відповідності як просту, але дуже гнучку механіку для основного ігрового процесу. Ігри RPG є чудовим прикладом.
- Хронометраж – деякі ігри розраховані на час. Нові плитки безперервно додаються, і гравець змушений складати матчі до того, як поле заповниться.

Перш за все, багато мобільних ігор включають механіку підбору інтерфейсу у свій власний ігровий процес. Це дозволяє їм запропонувати власний поворот оригінальної гри Match 3. Ще одна причина, чому ці типи ігор настільки успішні,

полягає в тому, що ці ігри, як правило, легко продати. Більшість великих програм також витрачають значні кошти на залучення користувачів і платні маркетингові кампанії. Ці рекламні зусилля також можуть сприяти їхньому величезному успіху.

Щоб підкреслити масштаб цих ігор: згідно з повідомленням у блозі Sensor Tower, у 2018 році у серії King’s Candy Crush гравці витрачали в середньому 4,2 мільйона доларів на день, що перевищило загальну суму франшизи за 1,5 мільярда доларів.

На завершення, ігри Match 3 були в центрі уваги рейтингів найприбутковіших мобільних ігор. Та швидше за все найближчим часом вони нікуди не дінуться.

1.2 Аналіз існуючих аналогів

Нижче проводиться аналіз та опис популярних ігор жанру «три в ряд».

1.2.1 Ігра Silver Tale

У Silver Tale [4] – це гра в якій геймер відправляється в дивовижну пригоду та вирішує складні квести-головоломки з трьома поєднаннями (рис. 1.2). Здоров'я короля знаходиться в критичному стані. Подорож до найглибшої печери і знаходження рідкісних трав, які можуть врятувати короля. По дорозі можна збирати рідкісні артефакти та починати створювати корисні предмети, які допоможуть геймеру у його подорожі. Геймер – єдина надія королівства.

Рисунок 1.2 – Інтерфейс гри «Silver Tale»

Особливості гри:

- Отримання масу задоволення від створення оновлюваних предметів для квестів;
- Дослідження складних рівнів цієї гри з елементами RPG;
- Відкрити сотні дивовижних предметів і виконати унікальні квести на скарб.

1.2.2 Гра Fishdom: Deep Dive

Fishdom: Deep Dive надає водний поворот у жанрі головоломки «три в ряд» [5]. Окрім прохолодної морської тематики, тут також можна побудувати акваріум. Можна зробити це, використовуючи монети, які отримані за проходження рівня головоломки 3 в ряд. Можна також придбати різні види риби та прикраси (рис. 1.3). Гра також дозволяє спілкуватися з водними тваринами.

Рисунок 1.3 – Інтерфейс гри «Fishdom: Deep Dive»

Рівні є досить складні. Існують різні цілі, які потрібно виконати, після завершення рівня, можна отримати монети. На жаль, не можна переграти старий рівень, тому є можливість заробити додаткові монети на попередньо зіграному рівні. На щастя, у Fishdom: Deep Dive є багато інших способів заробити монети.

Основні особливості гри:

- Понад 150 підводних декоративних елементів у 8 унікальних темах;
- Усі риби мають свою індивідуальність;
- Необмежена кількість акваріумів для проектування;
- 100 бонусних рівнів, розслаблюючий саундтрек, ексклюзивні шпалери та концепт-арт;
- Складний і веселий ігровий процес.

1.2.3 Гра Call Of Atlantis

Call of Atlantis — це гра-головоломка та гра в три ряди [6]. Вівтар Посейдона, який захищав місто Атлантиду, був викрадений (рис. 1.4). Посейдон забрав його з гніву через відсутність поклоніння і похвали з боку громадян Атлантиди. Сім кристалів вівтаря були розкидані по всьому світу. Завдання полягає в тому, щоб знайти ці предмети за допомогою розгадування головоломок та ігор.

Рисунок 1.4 – Інтерфейс гри «Call of Atlantis»

У кожному розділі гри Call of Atlantis потрібно пройти кілька рівнів з трьох раундів. Частинки стародавніх реліквій необхідно знайти, щоб наблизитись до місцезнаходження кристала для вівтаря Посейдона. Інші бонусні предмети можна отримати, щоб допомогти знищити додаткові плитки або дати додаткове життя. Кожен раунд розрахований на час. Усі предмети потрібно отримати до закінчення часу, інакше раунд доведеться повторити. Наприкінці кожного успішного раунду

буде розкриватися підсумок рахунку геймера, який відображає загальну суму балів і зароблені бонусні предмети.

У перервах між трьома раундами є головоломки для пошуку прихованих предметів, які потрібно розгадати. Сцена відображається як кімната, наповнена реліквіями та іншими предметами. Основна мета — зібрати певні стародавні реліквії, знайшовши приховані частини в кімнаті. Немає таймера, тому можна витрачати стільки часу, скільки потрібно, щоб розгадати головоломку. Доступна опція підказки, яка визначить місцезнаходження окремої частини реліквії. Раунд закінчується, коли всі реліквії будуть вилучені.

Функції Call of Atlantis:

- Дослідити сім стародавніх земель;
- Більше 65 унікальних рівнів;
- Чудові звукові та візуальні ефекти;
- Унікальна комбінація Match 3, Hidden Object і Adventure.

1.3 Аналіз ігрових рушіїв для розробки ігор

Потреба в ігровому рушії виникає тоді, як програмісти розробляють основну концепцію та сценарій гри. Ігровий рушій не тільки надає інструменти та функції, а також допомагає в розробці програм, які допомагають використовувати, модифікувати та створювати ігри за сюжетом замовника.

Перед розробкою гри доцільно провести аналіз існуючих ігрових рушіїв. В даній роботі проаналізовано найбільш популярні рушії у світі, а саме:

- Unity;
- Unreal Engine;
- CryEngine.

1.3.1 Рушій Unity

Unity – міжплатформне середовище розробки комп'ютерних ігор. Unity дозволяє створювати програми, що працюють під більш ніж 20 різними

операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, Інтернет-програми та інші (рис. 1.5).

Рисунок 1.5 – Інтерфейс середовища розробки «Unity»

Випуск Unity відбувся у 2005 році і з того часу триває постійний розвиток. Спочатку Unity призначався виключно для комп'ютерів Mac, але потім поступово виходили поновлення, що дозволяє працювати під Windows та інші ОС [7].

На Unity написані сотні ігор, додатків та симуляцій, які охоплюють безліч платформ та жанрів. Разом з тим, Unity використовується як великими компаніями - розробниками, так і незалежними студіями.

Редактор Unity має простий Drag&Drop інтерфейс, який легко налаштовувати. Він складається з різних вікон, завдяки чому можна налагоджувати гру прямо в редакторі. Двигун підтримує дві скриптові мови: C#, JavaScript (модифікація).

Проект в Unity ділиться на сцени (рівні) – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв та налаштувань. Сцени можуть містити як, власне, об'єкти (моделі), і порожні ігрові об'єкти – об'єкти, які мають моделі

(«пустушки»). Об'єкти, у свою чергу, містять набори компонентів, з якими і взаємодіють скрипти. До об'єктів можна застосовувати колізії (в Unity т. зв. колайдери – collider). У редакторі є система успадкування об'єктів – дочірні об'єкти повторюватимуть усі зміни позиції, повороту та масштабу батьківського об'єкта. Скрипти у редакторі прикріплюються до об'єктів як окремих компонентів [7].

При компіляції проекту створюється файл гри (для Windows), що виконується (.exe), а в окремій папці – дані гри (включаючи всі ігрові рівні та бібліотеки, що динамічно підключаються).

Середовище розробки MonoDevelop

MonoDevelop IDE (інтегроване середовище розробки) (http://monodevelop.com) – вільне мультиплатформене середовище розробки, призначене для створення програм на мовах C#, C, C++, Java, Visual Basic.NET, CIL, Nemerle, Boo. Вбудований у дистрибутив Unity3D як написання скриптів.

Можливості:

1. Підсвічування синтаксису – виділення синтаксичних конструкцій тексту з використанням різних кольорів, шрифтів та зображення. Застосовується полегшення читання вихідного тексту комп'ютерних програм, поліпшення візуального сприйняття.

2. Згортання коду, або фолдинг (англ. folding) – одна з функцій текстового редактора, що дозволяє приховувати певний фрагмент коду або тексту, що редагується, залишаючи лише один рядок.

3. Автодоповнення коду – функція у програмах, що передбачають інтерактивне введення тексту (редактори, оболонки командного рядка, браузери тощо) за доповненням тексту за введеною його частиною.

4. Вбудований налагоджувач – використовується на етапі налагодження комп'ютерної програми, на якому виявляють, локалізують та усувають помилки. Щоб зрозуміти, де виникла помилка, доводиться:

- впізнавати поточні значення змінних;
- з'ясовувати, яким шляхом виконувалася програма.

5. Модульне тестування – одиничне тестування, або модульне тестування (англ. unit testing) – процес у програмуванні, що дозволяє перевірити на коректність одиниці вихідного коду, набори з одного або більше програмних модулів разом із відповідними керуючими даними, процедурами використання та обробки.

Ідея полягає в тому, щоб писати тести для кожної нетривіальної функції чи методу. Це дозволяє досить швидко перевірити, чи не призвела чергова зміна коду до регресії, тобто появи помилок у вже відтестованих місцях програми, а також полегшує виявлення та усунення таких помилок.

Редактор скелетної анімації Spriter

Скелетна анімація – спосіб анімування двовимірних та тривимірних моделей у мультиплікації та комп'ютерних іграх.

Полягає в тому, що мультиплікатор або модельєр створює скелет, що являє собою зазвичай деревоподібну структуру кісток, в якій кожна наступна кістка «прив'язана» до попередньої, тобто повторює за нею рухи та повороти з урахуванням ієрархії в скелеті. Далі кожна вершина моделі «прив'язується» до будь-якої кістки кістяка. Таким чином, під час руху окремої кістки рухаються і всі вершини, прив'язані до неї. Завдяки цьому завдання аніматора сильно спрощується, тому що відпадає необхідність анімувати окремо кожну вершину моделі, а достатньо лише задавати положення та поворот кісток скелета.

Також завдяки такому методу скорочується обсяг інформації, необхідної для анімування. Досить зберігати інформацію про рух кісток, а рухи вершин обчислюються вже з них.

Spriter – програма для створення кісткової анімації, що працює з растровою графікою. Використовує модульний метод створення плавних анімацій. Цей спосіб має безліч переваг:

1. Економить час – неважливо, чи програміст є досвідченим чи початківцем художником з дизайну ігор, тут він зможе витрачати набагато менше часу на налаштування та налагодження персонажів, що анімуються, оскільки це дозволить повторно використовувати лише кілька модульних зображень.

2. Миттєва ітерація. Припустимо, необхідно змінити конфігурацію голови ігрового персонажа на етапі остаточної розробки. Якщо є Spriter у 2D-панелі інструментів, тоді необхідно буде лише змінити невелику частину зображень голови, тому що модулі налаштовані для використання у всіх кадрах анімації.

3. Налаштування користувача. Spriter набагато спрощує роботу дизайнера щодо створення будь-яких трюків, оскільки модульні зображення (частини тіла персонажів) можна вільно підштовхувати або повертати.

4. Необмежені зміни персонажів. Цей метод забезпечує надшвидке та безболісне створення нових персонажів на основі даних вже створеного персонажа. Також це ефективний з погляду пам'яті спосіб створення інструментів персонажу, які можуть змінюватися протягом гри (наприклад: збір бонусів, нова зброя тощо).

Аналіз популярних ігрових додатків показує, що на сьогоднішній день найбільшою популярністю користуються продукти, які мають простий візуальний стиль. Обчислювальна потужність сучасних смартфонів, звичайно, дозволить намалювати гарну картинку з величезною кількістю дрібних деталей, але маленький розмір екрану призведе до того, що об'єкти почнуть накладатися один на одного. Звідси випливає перше рішення щодо проекту, що розробляється. Візуальну складову гри буде мінімалізовано.

1.3.2 Рушій Unreal Engine

Unreal Engine - рушій, призначений для розробки ігор [8]. Він був розроблений Epic Games в 1988 році. Спочатку даний рушій розроблявся як шутер від першого лиця. У цей час він використовується для створення файтингів, RPG, Stealth та інших MMORPG. Для розробки ігор використовується мова програмування C ++. Ця мова є досить популярною серед девелоперів ігор та використовується як інструмент для розробки ігор.

У даний час Unreal Engine приваблює безліч розробників ігор. Маючи в своєму розпорядженні широкий спектр інструментів, Unreal Engine дає ліцензію користувача на створення власних ігор. Він найбільш популярний завдяки своїй легкості налаштувань, а також має різні інструментенти для простого створення

ігор. Більшість новичків вибирають Unreal Engine на початку кар'єри та презентаційного портфоліо.

Інтерфейс ігрового рушія представлений на рисунку 1.6.

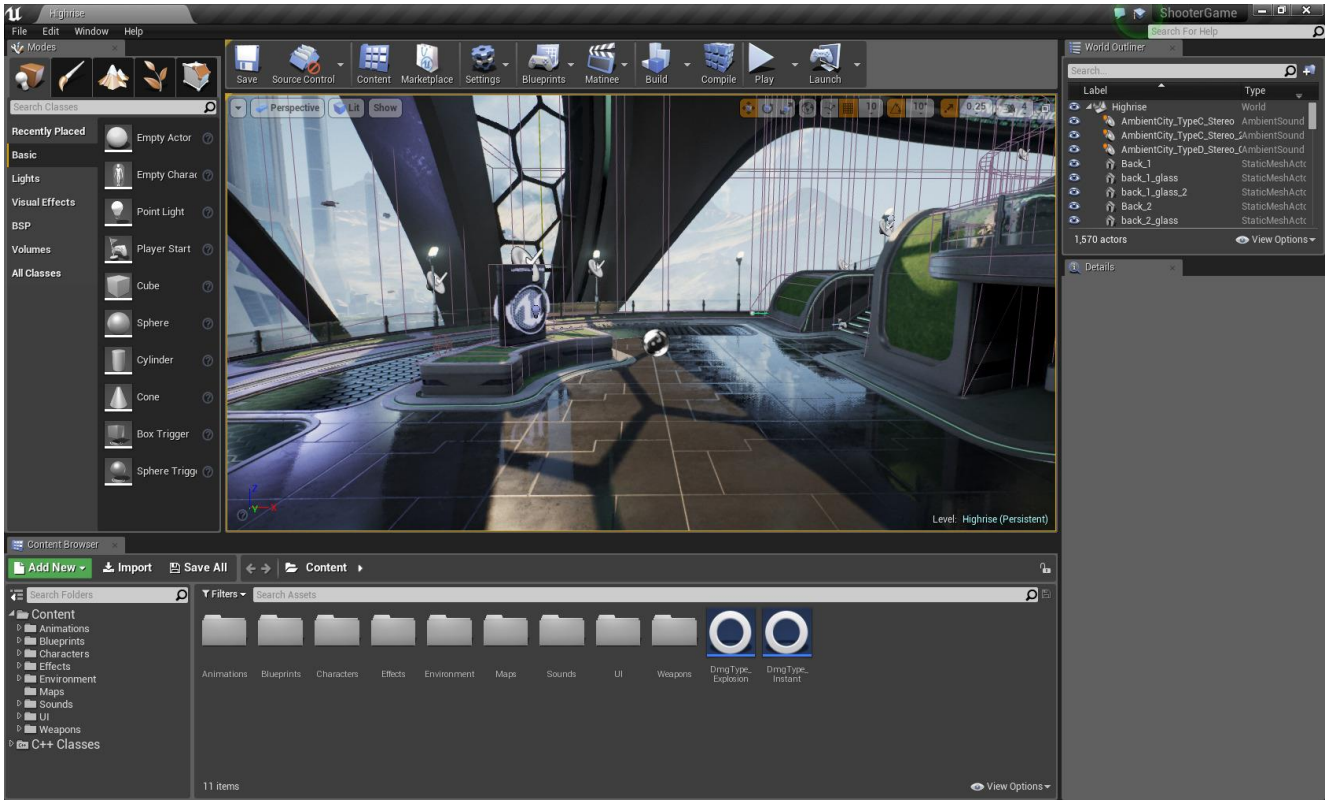


Рисунок 1.6 – Інтерфейс середовища розробки «Unreal Engine»

Основні переваги використання Unreal Engine:

- Якість графіки Unreal Engine.
- Юзабилити програми дуже затребувана.
- Користувацький інтерфейс Unreal Engine постійно оновлюється з використанням нових інструментів і опцій
- Він має прості коди і використовує вузли, які називаються Blueprints. Ці вузли допомагають користувачам створювати відеоігри та інші високоякісні ігри без написання скриптів і кодів.
- Використання мови програмування C ++, яка в свою чергу є досить поширена серед розробників.

На даний час Unreal Engine молодий ігровий рушій, та це не стає на заваді

створювати високоякісні відеоігри, які відповідають високим вимогам розробників.

1.3.3 Ігровий рушій CryEngine

Ігровий рушій CryEngine - перший комерційний двигун Crytek [9]. Його розробка була розпочата відразу після заснування компанії. Двигун спочатку розроблявся як технологічна демонстрація для американської компанії nVidia. Однак на виставці ECTS 2000 (англ. European Computer Trade Show – Європейська Комп'ютерна Виставка) Crytek справила велике враження на всіх великих видавців, відвідувачів та журналістів їхньою технічною демонстрацією, яка була показана у відділі nVidia. Після цього на основі двигуна було вирішено створити дві гри - "X-Isle" і "Engalus". Жодна з цих ігор так і не була випущена

2 травня 2002 року Crytek офіційно оголошує про те, що їхній ігровий двигун CryEngine повністю закінчений і готовий для ліцензування сторонніми компаніями.

Інтерфейс ігрового рушія представлений на рисунку 1.7.

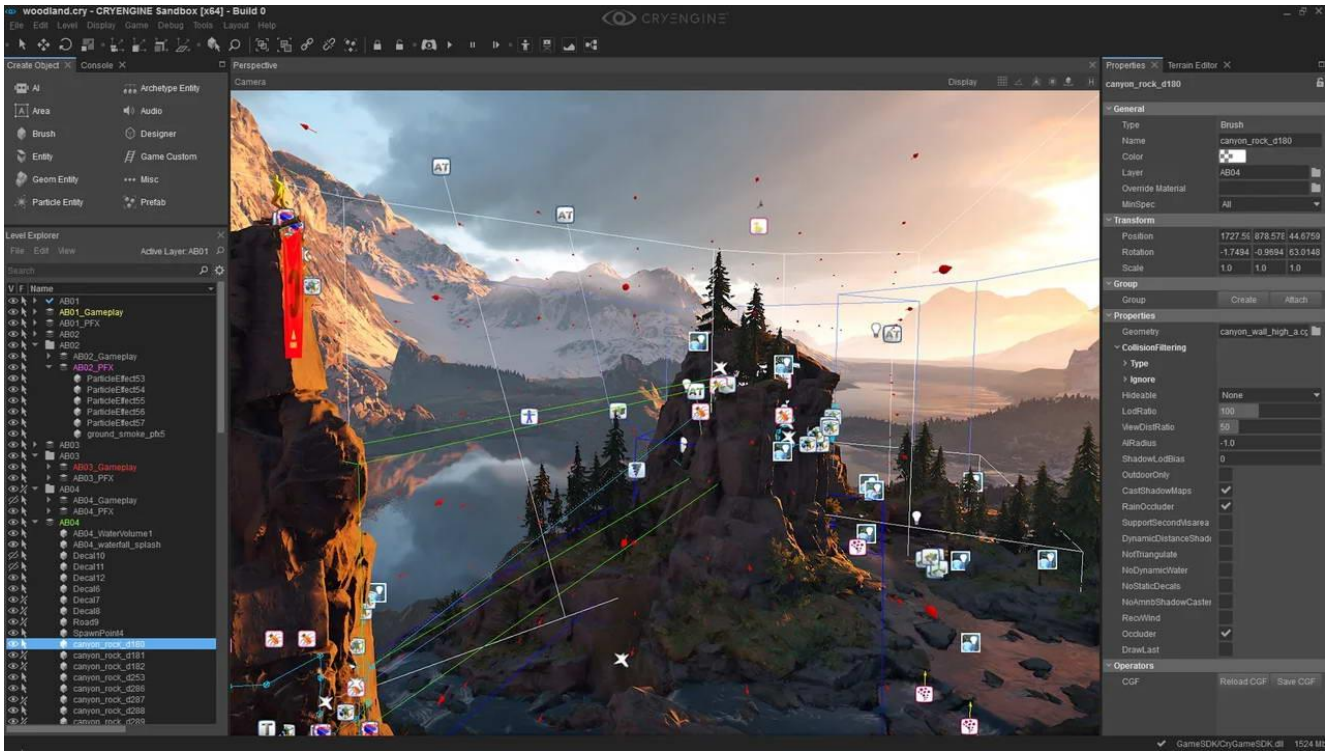


Рисунок 1.7 – Інтерфейс середовища розробки «CryEngine»

Особливості даного рушія:

- CryEngine Sandbox: редактор гри в реальному часі, що пропонує зворотний зв'язок «Що Ви бачите, то Ви і граєте».
- Рендер: інтегровані відкриті(outdoor) та закриті(indoor) локації без швів.
- Фізична система: підтримує інверсну кінематику персонажів, транспортні засоби, тверді тіла, рідину, ляльки (rag doll), імітацію тканини та ефекти м'якого тіла. Система об'єднана з грою та інструментами.
- Інверсна кінематика персонажів та змішана анімація.
- Система штучного інтелекту: включає в себе командний та скриптовий інтелект. Можливість створення поведінки ворогів не використовуючи код C++.
- Звукове оточення та механізм SFS: здатність точно відтворити звуки від природи з плавним сполученням без шва між середовищами та внутрішніми/зовнішніми розташуваннями в системі Dolby Digital 5.1. аудіо. Включає аудіо-підтримку EAX 2.0.
- Мережева система клієнта та сервера: Керує всіма мережними підключеннями для режиму з кількома гравцями. Це система мережі з низьким часом відгуку, заснована на архітектурі клієнт-сервер.
- Шейдери: скриптова система використовується для комбінування текстур по-різному збільшення візуальних ефектів. Підтримується реальне по-піксельне освітлення, вибоїсті відображення, заломлення, об'ємні ефекти жару, анімовані текстури, прозорі комп'ютерні дисплеї, вікна, кульові отвори та деякі інші ефекти.
- Ландшафт: Використовується розширена карта висот та скорочення полігонів для створення масивного, реалістичного середовища. Відстань може становити до 2 км, коли перетворено з ігрових модулів.
- Освітлення та тіні: комбінація розрахункових тіней та тіней реального часу, стенсильні тіні та lightmaps (карти тіней) для покращення динамічного оточення. Включає правильну перспективу з високою роздільною здатністю та об'ємні гладко-тіньові реалізації для драматичного та реалістичного внутрішнього

затінення. Підтримка просунутих технологій частинок та будь-який вид об'ємних ефектів освітлення на частинках.

- Туман: включає об'ємний, шаруватий та дальній туман для збільшення атмосфери та напруги.
- Інтеграція інструментальних засобів: об'єкти та будівлі, які створені на 3ds max або Maya, інтегровані в межах гри та редактора.
- Технологія Polybump: Автономна або повністю інтегрована з іншими інструментами, включаючи 3ds max.
- Скриптова система: Базується популярною мовою Lua. Ця зручна система дозволяє встановити та тонке настроювання параметрів зброю/гра, програвання звуків та завантаження графіки без використання коду C++.
- Модульність: Повністю написаний у модульному C++, з коментарями, документацією та розділами у багатьох DLL-файлах.

1.4 Етапи розробки ігор

Багато людей навіть не уявляють, наскільки тривалим та трудомістким процесом є створення відеоігор. Саме тому найчастіше над однією відеогрою працює велика команда. Кожна людина у команді — фахівець у своїй галузі: художник, програміст, звукорежисер, тестувальник тощо. Для спрощення створення гри будуть розглянуті базові етапи її розробки з невеликим акцентом на Unity, так як на сьогоднішній день його вважають найбільш затребуваним кросплатформовим ігровим двигуном, що за багатьма показниками перевершує своїх конкурентів [10].

Основні етапи розробки відеоігри можна розділити на три пункти:

- підготовчий етап або проектування:
- ціль;
- засіб;
- творча частина:
- ігрова механіка;

- рівні;
- оформлення;
- сюжет;
- звук;
- випуск продукту;
- тестування та усунення помилок.

Розглянемо ці пункти докладніше. Почнемо із проектування гри. Безперечно гра починається з мети. Необхідно вирішити, що має вийти. Не можна розпочати створення гри без основної ідеї, яка визначає всю гру із самого початку. Чим оригінальнішим буде задум, тим більше шансів, що гра сподобається користувачам.

Далі потрібно визначитися із жанром.

Жанр можна коригувати в процесі створення гри, але змінити його повністю неможливо. Жанр повинен бути один від початку гри до кінця.

Далі необхідно перейти до другого пункту проектування. Після того, як ціль була задана, потрібно вибрати засоби для досягнення обраної мети. До засобів можна віднести програмний код, тобто вибір мови програмування та подальша робота з ним, а також до засобів відноситься ігровий двигун. В даний час найкращим ігровим двигуном є Unity. На Unity написані тисячі ігор, додатків та симуляцій, які охоплюють безліч платформ та жанрів.

Unity дозволяє створювати програми, що працюють під більш ніж 20 різними операційними системами, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми такі як: iOS, Android, Windows, MacOS, Linux, WebGL, PlayStation 4, Xbox One, Wii U , Oculus Rift, Nintendo Switch та інші.

Після цього необхідно визначитись із механікою гри. Ігрова механіка є важливим компонентом системи створення ігор. По суті ігрова механіка - це певне зведення правил, за якими працює гра. Також можна сказати, що ігрова механіка - правила, що реалізують інтерактивну взаємодію гравця та ігри. Ігрові механіки можуть бути:

- простими – елементарна дія, яку робить гравець у грі, наприклад –

кидок кубика або переміщення у просторі з однієї точки в іншу;

- складовими — складаються з кількох простих дій.

Основою ігрової механіки є об'єкти. Це всілякі бонуси, декорації, головні та другорядні герої тощо. У всіх цих об'єктів є свої властивості. До того ж ігрова механіка визначає якими способами користувач керуватиме вищепереліченими об'єктами. Яка дія буде після натискання певної кнопки. Тобто, ігровою механікою управляє користувач, а існує ще фізичний рушій, який відповідає за ті дії, які відбуваються без прямої участі користувача. Також, не слід забути про штучний інтелект, який, наприклад, відповідає за поведінку ворогів головного персонажа. Безумовно штучний інтелект, не є важливою частиною створення гри, але він додає ефекту несподіванки для користувача, а це завжди цікаво.

Усі ці пункти необхідно враховувати під час розгляду ігрової механіки.

Далі необхідно переходити до рівнів. Рівень є певною локацією, наприклад, місто, ліс, приміщення і т.п. швидкістю. З таким підходом гра довше залишається цікавою. Однак поділ на рівні не обов'язковий, є розробники, які, навпаки, прагнуть робити гру без явного переходу між рівнями, створюючи враження безперервності.

Окремо необхідно відзначимо меню. Початкове меню вважається візитною карткою гри, тому воно також має бути красиво оформлене.

Одне з головних у створенні гри це її сюжет. Користувач має хотіти пройти гру до кінця. Хоча, в наш час сюжет все ж таки починає відходити на другий план, деякі просто пропускають діалоги персонажів, відеовставки тощо, якщо вони тривають довше хвилини, тому дані вставки повинні бути максимально короткими, але інформативними, адже без них гра, здебільшого стає нудніше.

Гарно намальований проект не можна вважати грою без відповідної музики. Музика служить передачею емоційного настрою. Фонова музика найбільше впливає на настрій гравця, а не лише наповнює тишу. Звукові ефекти так само важливі, це можуть бути кроки по дерев'яній підлозі ворога, що наближається, або звуки наближення автомобіля. Всі ці деталі необхідні цілісності картинки.

Третьою звуковою складовою гри є озвучення. Це не обов'язкова складова, тому що її можна замінити субтитрами, але все ж таки з озвученням набагато

цікавіше, адже з нею передаються емоції та характер героя, до того ж не всі читають субтитри.

Коли гра зібрана, залишається лише усунути помилки, вони будуть у будь-якому випадку. Повністю зібрана гра, але не перевірена на наявність помилок називається бета версією. Пошуком помилок займається тестувальник, буває, як тестувальників виступають звичайні гравці, таким чином вони отримують гру на вигідних умовах, але з помилками.

1.5 Сучасні системи програмування ігор

Розробники ігор мають знання, уяву, вмілі руки та працюючий комп'ютер, але вони не зможуть побудувати віртуальний світ без другого важливого інструменту – мови програмування.

Вибір правильної мови програмування, фреймворку або інших інструментів має важливе значення для успішного розробки програмного забезпечення. Це рішення вплине на те, як використовувати основні ресурси, такі як час, бюджет, обладнання і т. д. Ігрова індустрія дуже різноманітна, і вибір мови програмування для гри багато в чому залежить від цілей та завдань. Цьому вибору також сприяють багато інших факторів – цільова платформа, складність проекту чи жанр гри.

В основному найбільш доцільно розробляти ігри на двох мовах програмуваня: C# та C++.

Мова програмування C#

C# (вимовляється як «сі шарп») – проста, сучасна об'єктно-орієнтована мова програмування [11]. Розробка програмних компонентів проходить у формі автономних та самоописових пакетів, що реалізують окремі функціональні можливості. Особливістю таких компонентів є модель на основі властивостей, методів і подій. Кожен компонент наділений атрибутами, що надають відомості про компонент і вбудовані елементи документації. C# надає мовні конструкції, які безпосередньо підтримують таку концепцію роботи.

Перелік деяких функцій мови C#, що можуть забезпечити надійність та

стійкість додатків:

• Складання сміття автоматично звільняє пам'ять, зайняту знищеними та невикористовуваними об'єктами;

• Обробка винятків дає структурований та розширюваний спосіб виявляти та обробляти помилки;

• Сувора типізація мови не дозволяє звертатися до не ініціалізованих змінних, виходити за межі масиву або виконувати неконтрольоване наведення типів.

У C# існує єдина система типів. Всі типи C# є спадкоємцями одного кореневого типу object. Отже, всі типи використовують загальний набір операцій, і значення будь-якого типу можна зберігати, передавати та обробляти.

Мова програмування C++

C++ — це мова програмування середнього рівня, розроблена Б’ярном Страуструпом, починаючи з 1979 року в Bell Labs. C++ працює на різних платформах, таких як Windows, Mac OS і різні версії UNIX [12]. У цьому підручнику C++ використовується простий і практичний підхід до опису концепцій C++ для початківців до досвідчених інженерів-програмістів.

Ось деякі з ключових переваг вивчення C++:

- Дуже близький до апаратного забезпечення, тому, що можна працювати на низькому рівні, що дає великий контроль щодо управління пам’яттю, кращої продуктивності та, нарешті, надійної розробки програмного забезпечення.
- Програмування на C++ дає чітке розуміння об'єктно-орієнтованого програмування. Зрозумілу реалізацію поліморфізму низького рівня.
- C++ є найбільш широко використовуваною мовою програмування в прикладному та системному програмуванні. Таким чином, можна вибрати сферу інтересів розробки програмного забезпечення.
- Дає зрозуміти різницю між компілятором, компонувальником і завантажувачем, різними типами даних, класами зберігання, типами змінних, їх областями тощо.

C++ присутній майже в кожній області розробки програмного забезпечення. Ось декілька з них:

- Розробка прикладного програмного забезпечення. Використовується при розробці майже всіх основних операційних систем, таких як Windows, Mac OSX та Linux. Крім операційних систем, основна частина багатьох браузерів, таких як Mozilla Firefox і Chrome, була написана на C++. C++ також використовувався при розробці найпопулярнішої системи баз даних під назвою MySQL.
- Розробка мов програмування. Широко використовується при розробці нових мов програмування, таких як C#, Java, JavaScript, Perl, UNIX C Shell, PHP і Python, Verilog тощо.
- Програмування обчислень. на C++ є найкращим для використання вченими через швидку швидкість та обчислювальну ефективність.
- Розробка ігор. Надзвичайно швидкий, що дозволяє програмістам виконувати процедурне програмування для інтенсивних процесорних функцій і забезпечує кращий контроль над апаратним забезпеченням, через що він широко використовується при розробці ігрових двигунів.
- Вбудована система. Активно використовується для розробки медичних та інженерних програм, таких як програмне забезпечення для апаратів МРТ, високоякісних CAD/CAM систем тощо.

Цей список можна продовжувати. Є різні області, де розробники програмного забезпечення, які із задоволенням використовують C++ для створення чудового програмного забезпечення. В таблиці нижче наведене порівняння найпопулярніших мов, які використовують для програмування ігор.

Таблиця 1.1 – Порівняльна таблиця мов програмування

Мова програмування / Критерій	C++	C#	JavaScript
Можливість компіляції	+	+	+
Створення об'єктів на сітці	+	+	-

Динамічна типізація	-	+	+
Статична типізація	+	+	-
Наявність бібліотек для роботи з графікою та мультимедіа	+	+	+
Функціональна парадігма	+\-	+\-	+\-
Узагальнене програмування	+\-	+	+

1.6 Висновок

В цьому розділі було ознайомлено з теоретичним поняттями. Було вивчено предметну область, зроблено аналіз існуючих аналогів розробленої гри, розглянуто основні рушії розробки ігор та мов програмування, за допомогою яких розробляються ігри.

2 ОПИС ПРОГРАМНОГО ДОДАТКУ

2.1 Опис технічного завдання

Завданням дипломного проекту є створення гри при створенні ряду класів, об’єкти яких будуть взаємодіяти між собою. Дана гра повинна бути жанру «три в ряд». Вимоги до проекту:

- графічний інтерфейс гри повинен відповідати сучасним вимогам, повинен бути простим, щоб не перевантажувати ігровий процес;
- нарахування зароблених очок в залежності від розташування однакових плиток в ряд (після розташованих в один ряд однакових плиток, вони мають анігілюватися, кількість однакових плиток повинна бути мінімум 3);
- ігрова механіка повинна бути інтуїтивно зрозуміла користувачеві;
- приваблива візуалізація для залучення уваги;
- музичний супровід впродовж ігри;
- звуки дій на натискання кнопок миші;
- можливість встановити рекорд;
- можливість розпочати гру заново.

Створимо UML діаграму, яка буде відображати як користувач буде взаємодіяти з нашою грою. Діаграма представлена нижче Рис 2.1.

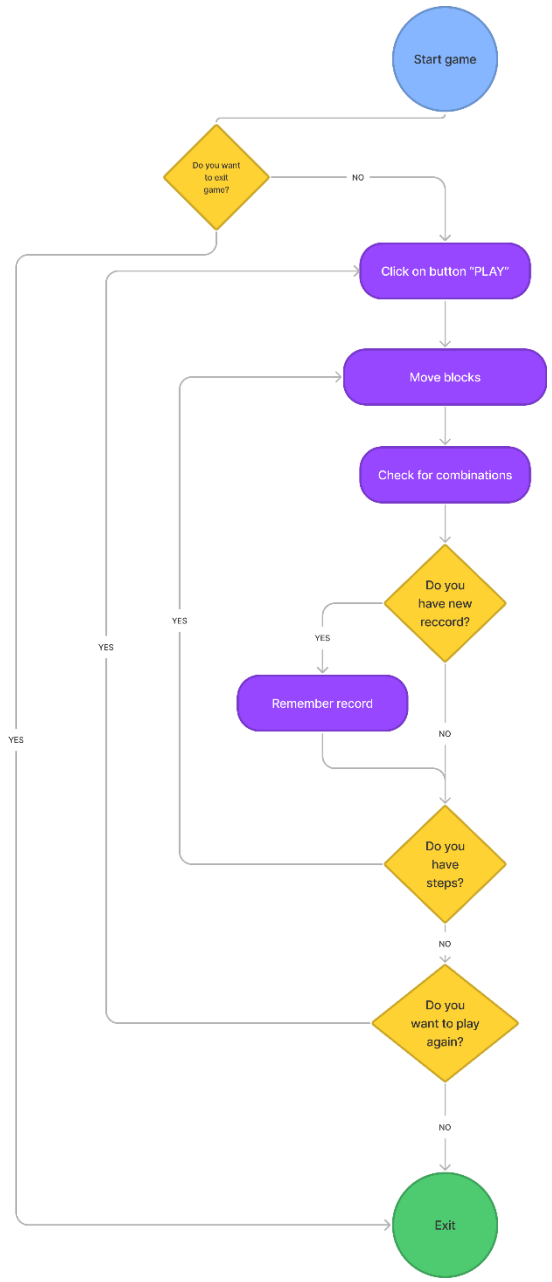


Рис. 2.1 – Діаграма діяльності

Створена діаграма відображає функціонал з яким буде взаємодіяти користувач. Діаграма є візуальним відтворенням технічного завдання, яке має бути відтвореним в нашому додатку.

2.2 Опис середовища та системи

2.2.1. Unity

Unity - це мультиплатформний інструмент для розробки дво- та 3D-програми та ігри під керуванням Windows та OS X. Програми на базі Unity працюють під керуванням Windows, OS X, Android, Apple iOS, Linux, а також ігрових консолей Wii, PlayStation 3 та XBox 360. Спеціальний плагін для браузера Unity, а також шляхом експериментальної реалізації у модулі Adobe Flash Player. Програми, створені за допомогою Unity, підтримують DirectX та OpenGL. Редактор Unity має простий інтерфейс Drag&Drop, який легко налаштовується та складається з різних вікон, тому ви можете налаштовувати гру прямо у редакторі. Двигун підтримує три скриптові мови: C#, JavaScript (модифікація). Проект Unity поділений на сцени (рівні) – окремі файли, які містять свої ігрові світи з власним набором об'єктів, сценаріїв та налаштувань. Сцени можуть містити як об'єкти (моделі), так і порожні ігрові об'єкти, тобто ті, які не мають моделі. Об'єкти, своєю чергою, містять набори компонентів, із якими взаємодіють скрипти. Об'єкти з видимою геометрією мають компонент Mesh Renderer, який робить їх модель видимою.

Unity також підтримує фізику твердого тіла та тканин, фізику Ragdoll (ганчіркова лялька). У редакторі є система наслідування об'єктів; дочірні об'єкти будуть повторювати всі зміни положення, повороту та масштабу батьківського об'єкта. Скрипти у редакторі прикріплюються до об'єктів як окремі компоненти.

Коли ви імпортуєте текстуру в двигун, ви можете згенерувати альфа-канал, рівні MIP, карту нормалей, карту освітлення, карту мапінгу, але ви не можете прикріпити текстуру безпосередньо до моделі - ви створите матеріал, з якого шейдер буде призначено, а потім матеріал буде прикріплений до моделей. Редактор Unity підтримує написання та редагування шейдерів. Крім того, він містить компонент для створення анімації, анімацію також можна попередньо створити у 3D-редакторі та імпортувати разом із моделлю, а потім розбити на файли.

Unity має вбудовану підтримку мережі. Двигун гри повністю інтегрований із середовищем розробки. Це дозволяє протестувати гру прямо в редакторі. Має вбудований ландшафтний генератор.

2.2.2 MonoDevelop

MonoDevelop - це інтегроване середовище розробки (IDE), що постачається разом з Uniti. IDE поєднує в цьому функції текстового редактора з додатковими можливостями для накладання і виконання інших завдань з управління проектами.

MonoDevelop має набір можливостей, що необхідні для сучасного інтегрованого середовища розробки:

- налаштування підсвітки синтаксису;
- автоматична генерація коду;
- виділення блоків кодів з можливістю їх згортання чи розгортання;
- інтелектуальні роботи із відступами в коді;
- можливості рефакторингу (перейменуванню класу чи методу, автоматична реалізація інтерфейсу у виробничого класах);
- віддана навігація за кодом (навігація за кліками, методами, властивостями);
- візуальний редактор форми для проекту;
- створення кількох розкладок інтерфейсу і переключення між ними;
- безліч стандартних шалонів проектів;
- автоматичне створення бінарних пакетів і архіву після компіляції коду;
- работа з базами даних;
- створення приложень з GUI, що підтримує кілька мов;
- інтеграція з Suvversion для управління вихідним кодом;
- підтримка NUnit для створення Uniт-тестів;
- автоматичне складання документації;
- розширення за рахунок доповненння та зовнішніх бібліотек;
- інтеграція із Microsoft Visual Studio і .NET Framework (в середовищі

Microsoft Windows).

На рисунку 2.2 представлений інтерфейс работи програми.

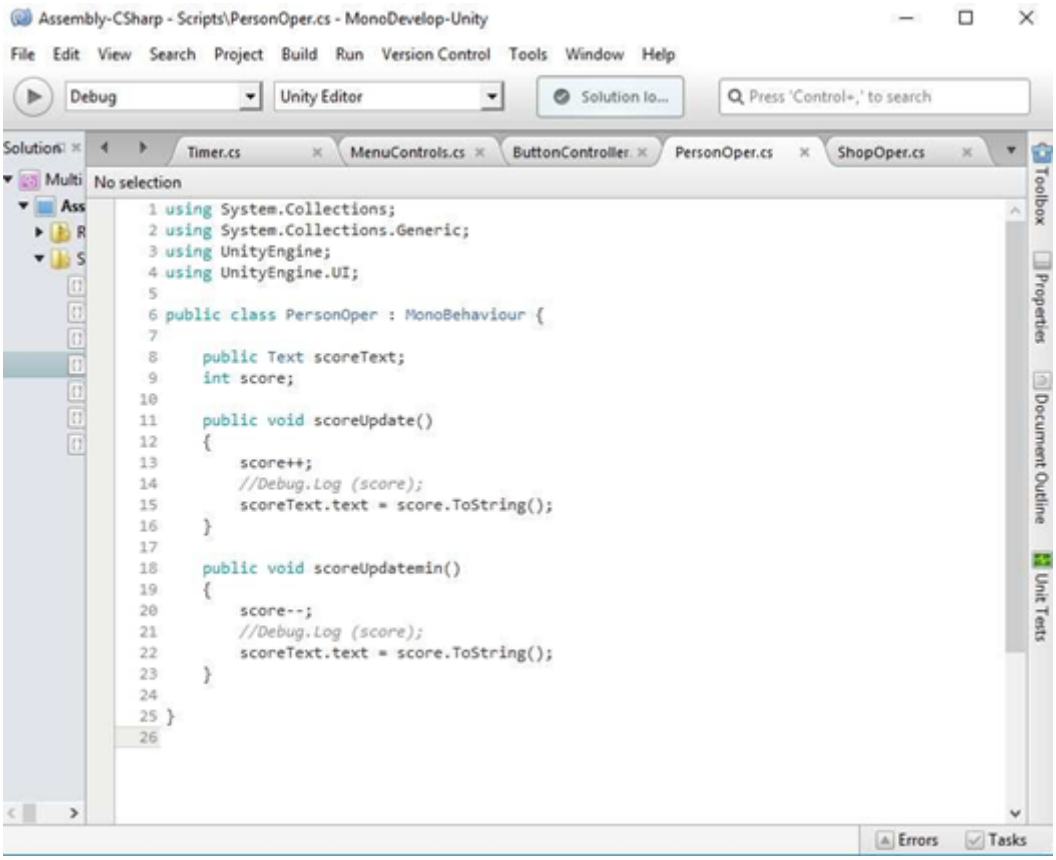


Рисунок 2.2 – Інтерфейс «MonoDevelop»

2.2.3 Структура C# скрипта в Unity

Unity використовує компонентний підхід. Компонент - це клас, наслідуваний від MonoVenaviour. Один компонент повинен відповідати за одну поведінку.

Скрипти можуть бути створені в панелі проекту. Для цього достатньо натиснути кнопку Create і вибрати мову, на якому буде створюватися скрипт. Прикріпити скрипт до об'єкта можна шляхом перекручування або натискання на кнопки Add Component.

Після того, як буде створено скрипт в Unity і відкрито за замовчуванням в MonoDevelop, можна буде побачити наступне:

using UnityEngine;

using System.Collections;

```
public class MainPlayer : MonoBehaviour {
// Use this for initialization void Start () {
}
// Update is called once per frame void Update () {}
}
```

Назва створеного скрипта повинна повністю відповідати назві створеного автоматичного класса, який наслідується від вбудованого класу MonoVenaviour.

Функція Update – найвикористовуваніша функція в Unity. Вона викликається один раз за кадр в кожному скрипті, що використовує її. Майже все, що вимагає змін або регулювання на постійній основі, прописується в цій функції. Переміщення нефізичних об'єктів, прості таймери і виявлення введення звичайно реалізуються в цій функції. Варто відзначити, що дана функція не викликається на основному таймлайні. Якщо один кадр займає більше часу для обробки, чим наступний, тоді час між викликами функції буде різним.

Функція Start – функція, яка викликається автоматично перед функцією Urdate, коли скрипт завантажений. Функцію можна використовувати для всього, що потрібно, тільки коли компонент скрипту включений. Це дозволяє відкинути будь-яку частину ініціалізації, поки вона не знадобиться.

2.3 Засоби розробки

Написання безпечного коду, що забезпечує безпеку без вразливостей, є критичним викликом для кібербезпеки. Написання коду без вразливостей вже давно принаймні так само складно, як написання коду без помилок. Хоча в програмному забезпеченні є багато інших потенційних джерел ризику безпеки, розробка коду без відомих класів уразливостей завжди здавалася посильною метою. Він покладається на людей-розробників, які використовують інструменти, методи та процеси для створення програмного забезпечення, яке не має особливо відомих типів дефектів.

Один з найефективніших підходів — дослідження мов та інструментів програмування — приніс технології, які, як показано, протистоять категоріям уразливостей, в основному, не допускаючи їх. Безпечні мови пам'яті, які керують виділенням та звільненням пам'яті, замість того, щоб вимагати від програміста це робити, унеможливлюють для розробників створення вразливостей переповнення буфера та деяких інших типів впливу через відсутні перевірки меж масиву, використання нульового покажчика та даних. витік через повторне використання пам'яті. Потокобезпечні мови можуть вирішувати випадки, коли умови гонки можуть бути використані, щоб підірвати перевірки, пов’язані з безпекою в програмі.

Рисунок 2.3 –Методи розробки

У межах спільноти розробників програмного забезпечення групи та організації, які мають на меті безпечну розробку програмного забезпечення, включили інструменти та методи у свій життєвий цикл розробки програмного забезпечення, щоб включити життєвий цикл безпечної розробки. Раннє програмне забезпечення з високою надійністю використовувало формальні методи для

визначення властивостей безпеки системи, а також перегляд коду, щоб використовувати людину для пошуку таких недоліків на рівні кодування. Корпорація Майкрософт створила свій життєвий цикл розвитку безпеки, додавши аналіз першопричин, навчання безпеки, моделювання загроз, конкретні вимоги до безпечного кодування та тестування безпеки, яке включало тестування на проникнення та нечітке тестування. Практика, як правило, впроваджується на основі потреб бізнесу, відчутного впливу на безпеку та відповідає усталеній або розвивається практиці розвитку.

Необхідні дослідження, які впливають на те, що працює, а що може працювати для безпечного розвитку. Здається, що нинішні дослідження відіграють, на жаль, обмежену роль у створенні, пропонуванні, оцінці та підтвердженні інструментів, методів і процесів, які використовуються на практиці для безпечного розвитку. Зокрема, дослідження рідко стосуються безпосередньо інструментів і методів, оскільки вони використовуються, у контексті, в якому вони використовуються. Нам потрібні додаткові дослідження ефективності та результатів безпечних інструментів, методів і процесів розробки. Це дослідження можна судити про його вплив на те, як розробка програмного забезпечення працює на практиці. Властивості дослідження впливають на ймовірність такого впливу.

Суворість дослідницького наукового експериментування вимагає ряду вимог до процесу, включаючи формулювання гіпотези, що перевіряється, контроль змінних експерименту, щоб переконатися, що експеримент фактично перевіряє гіпотезу, і аналіз експериментальних даних і результатів для математичного підтвердження гіпотези (або спростувати нульову гіпотезу). Хоча ці процеси можуть стати основою важливих фундаментальних досліджень безпечної розробки, вони часто уникають безладних реалій, пов’язаних із впровадженням техніки на практиці, саме тому, що ці безладні реалії ускладнюють проектування експериментів.

Негативні результати дослідження, які не підтверджують, що безпечна техніка розробки підвищує безпеку, хоча й важливі для галузі досліджень, навряд чи вплинуть на безпечний розвиток на практиці. Перший урок розробника безпеки

у великій технологічній компанії полягав у тому, що вказувати розробникам не робити чогось майже завжди було неефективним, якщо це не було поєднане з альтернативою, яку вони могли б використати для досягнення мети застарілої практики. «Не кидайте свою власну криптовалюту» має постати разом із криптобібліотекою, яку слід використовувати. Крім того, виявити, що інструмент або техніка експериментально неефективні для забезпечення безпеки, не доводить, що вони неефективні за межами контрольованого експерименту, у більш широкому, безладному та різноманітнішому контексті розробки програмного забезпечення.

Які з тих речей, які робляться в дослідженні, сподіваються на практичне перенесення в безпечний розвиток? Дві сучасні тенденції досліджень безпеки дають певну надію на безпечний розвиток. Одна з них полягає в тому, що безпечна розробка стала темою на конференціях з дослідження безпеки, охоплюючи такі теми, оцінка інструментів, які допомагають розробникам уникати вразливостей, і вимірювання здатності розробників кодувати функціональні можливості, що стосуються безпеки. .

Інша обнадійлива тенденція — оцінка артефактів. Багато розробок програмного забезпечення базується на існуючому програмному забезпеченні, використовуючи фреймворки, бібліотеки та відкритий код. Пропонування артефакту, який використовується для встановлення та підтвердження ідеї дослідження, зменшує бар’єри для передачі цієї ідеї в розробку програмного забезпечення. Доступ до коду з відкритим кодом з умовами ліцензії, зручними для повторного використання, може збільшити його потенціал для використання. Деякі дослідницькі стимули змінюються, щоб заохочувати подання артефактів у рамках процесу подання та публікації наукової роботи на конференціях з безпеки, таких як USENIX Security і ACSAC.

2.4 Вимоги до технічного забезпечення

Запуск комп’ютерних програм на голій машині – не нова концепція. Так почалися обчислення десятиліття тому, коли програма завантажується вручну за допомогою перемикачів і запускається натисканням кнопки запуску. Однак наші комп’ютерні системи складні, і вони виросли за межами пропорції, створюючи великий семантичний розрив між додатками та обладнанням. Наприклад, остання операційна система (ОС) Microsoft XP має 40 мільйонів рядків коду. Настав час переглянути еволюцію обчислювальної техніки та шукати рішення, використовуючи нові парадигми. ОС і середовище швидко змінювалися з останніх 30 років, що робить речі застарілими, перш ніж вони зможуть стати продуктивними протягом свого життя. Наприклад, Microsoft випустила понад 20 основних випусків ОС за останні 25 років. Кожна мова та середовище випускають новий випуск кожні шість місяців. Коли ОС або її середовище змінюються, це має ефект пульсації, що призводить до застарілих програм. Як ми можемо зупинити таке поширення продуктів і технологій і зробити програми стабільними? Не забезпечуючи жодної перевірки, одним із можливих підходів є уникнення операційної системи та її середовища. У деяких областях обчислень це робиться повільно, непомітно.

Після завантаження ПК він перебуває в реальному режимі, де адресація обмежена 1 МБ і немає захисту вашого коду. Це просто в режимі дискової операційної системи (DOS). Щоб завантажувати та запускати більші програми та мати доступ до більшої пам’яті, вам потрібно навчитися переходити в захищений режим за допомогою інструкцій на мові асемблера. Однак усі виклики базової системи введення/виводу (BIOS) доступні лише в реальному режимі. Якщо ви хочете використовувати BIOS для вводу-виводу, вам потрібно мати механізми перемикання із захищеного режиму в реальний для виконання викликів BIOS. Таким чином, ваша програма на C++ або якась керуюча програма повинні виконувати це перемикання, прозоре для користувача. Якщо ви плануєте використовувати програмні переривання (їх 255) замість переривань BIOS, то вам потрібно написати власний код на зборці для вирішення всіх операцій введення-виводу. Ми використовували як BIOS, так і програмні переривання і написали перемикач захищеного режиму в режим реального для роботи в обох режимах. Це

нетривіальна річ, яку потрібно зробити під час складання, яка вимагає від вас ознайомлення з документом специфікації архітектури Intel, який доступний на їх веб-сайті. Цей документ є корисним ресурсом для розуміння та реалізації переривань, схем адресації, засобів завдань, пасток і винятків.

2.5 Опис архітектури та структури програмного забезпечення

Архітектура програмного забезпечення дає пояснення того, як системи поводяться на структурному рівні. Системи, які ви використовуються, мають набір компонентів, розроблених для виконання певного завдання або набору завдань. Архітектура програмного забезпечення забезпечує фундамент, на якому все програмне забезпечення, яке є у компанії, можна змінити, створити або вилучити з експлуатації.

Архітектура програмного забезпечення впливає на якість, продуктивність, обслуговування та успіх системи на основі дизайну. Не розглядаючи архітектуру програмного забезпечення на регулярній основі, компанія відкриває себе для довгострокових наслідків, і проблеми, які можуть поставити їх системи під загрозу поломки, злому або низької продуктивності.

У сучасних системах існують загальні шаблони в архітектурі програмного забезпечення, які називаються архітектурними системами для програмного забезпечення. У більшості випадків для створення цілісної системи використовується кілька різних архітектурних систем, особливо для систем, які створювалися роками або працювали, або тих, які були побудовані різними розробниками.

Всередині проекту мають бути зрозумілі правила організації його структури. А саме:

- має бути зрозуміло, що де лежить;
- зрозуміло куди додати нові файли та вихідники;
- структура адаптивна і зручна;
- структура не дуже складна і не подрібнена.

Під час розробки гри було створено 7 головних папок всередині проекту, кожна папка має своє призначення (рис. 2.4), а саме:

- «Animations» - зберігає інформацію про всі анімації, що будуть використовуватись;
- «Audio» – містить всі аудіо файли: музичний супровід гри та різні звуки, для підсилення ефекту дії користувача;
- «Fonts» – зберігає додаткові шрифти, що будуть використовуватись в проекті для виводу тексту на екран;
- «Prefabs» – містить об’єкти гри із їхніми налаштуваннями;
- «Scenes» – складається із всіх побудованих сцен проекту;
- «Scripts» – містить всі скрипти, що були розроблені для гри;
- «Sprites» – зберігає всі картинки, що використовуються у грі.

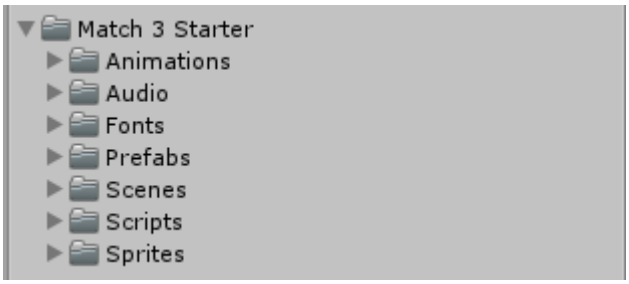


Рисунок 2.4 –Основна структура проекту

Добре організована структура проекту – гарантія того, що проект буде успішний.

2.5.1 Опис класів

Клас у C# є будівельним блоком, який веде до об'єктно-орієнтованого програмування. Це визначений користувачем тип даних, який містить власні члени даних і функції-члени, до яких можна отримати доступ і використовувати, створивши екземпляр цього класу. Клас C# схожий на план для об'єкта.

Наприклад: розглянемо клас автомобілів. Може бути багато автомобілів з різними назвами та марками, але всі вони будуть мати деякі спільні властивості, як-

от усі вони матимуть 4 колеса, обмеження швидкості, діапазон пробігу тощо. Отже, автомобіль – це клас і колеса, обмеження швидкості, пробіг. їх властивості.

Клас — це визначений користувачем тип даних, який має члени даних і функції-члени.

Члени даних — це змінні даних, а функції-члени — це функції, які використовуються для маніпулювання цими змінними, і разом ці члени даних і функції-члени визначають властивості та поведінку об’єктів у класі.

На прикладі класу Car членом даних буде обмеження швидкості, пробіг тощо, а функціями члена можуть бути гальмування, збільшення швидкості тощо.

Об'єкт - це екземпляр класу. Коли клас визначено, пам’ять не виділяється, але коли він створюється (тобто створюється об’єкт), пам’ять виділяється.

Визначення класу та оголошення об'єктів

Клас визначається в C# за допомогою ключового слова class, за яким слідує ім'я класу. Тіло класу визначається всередині фігурних дужок і закінчується крапкою з комою в кінці.

Оголошення об'єктів: коли визначено клас, визначається лише специфікація об'єкта; пам'ять або сховище не виділено. Для використання даних і функцій доступу, визначених у класі, потрібно створити об’єкти.

Синтаксис:

ClassName ObjectName;

Доступ до членів даних і функцій-членів: До членів даних і функцій-членів класу можна отримати доступ за допомогою оператора dot(‘.’) з об’єктом. Наприклад, якщо ім’я об’єкту – sfx, і потрібно отримати доступ до методу -члена з ім’ям Play (), тоді доведеться написати sfx. Play () .

Доступ до членів даних

Доступ до загальнодоступних членів даних також здійснюється таким же чином, але об’єкт не може мати прямий доступ до членів приватних даних. Доступ до члена даних залежить виключно від контролю доступу цього члена даних.

Цей контроль доступу надається модифікаторами доступу в C#. Існує три модифікатори доступу: відкритий, приватний і захищений.

Клас - це шаблон або план, який зв'язує властивості та функції сутності. Ви можете помістити всі сутності або об’єкти, що мають подібні атрибути, під одним дахом, відомим як клас. Класи далі реалізують основні концепції, такі як інкапсуляція, приховування даних та абстракція. У C#клас діє як тип даних, який може мати кілька об’єктів або екземплярів типу класу.

В проекті реалізовано такі класи:

1. «BoardManager» – клас, призначений для рисування дошки та управління нею;
2. «Tile» – клас, призначений для формування, рисування та обробляння дій із плитками;
3. «GUISizePingPong» – клас, для управління графічним інтерфейсом користувача;
4. «GameManager» – клас, призначений для керування сценою;
5. «GUIManager» – клас для нарахування кількості очок та кількості спроб перестановок плитки;
6. «SFXManager» – клас, що призначений для музичного супроводу для гри, а також формування звуків, які супроводжують дії користувача.

2.5.2 Опис методів

Відзначимо основні завдання прототипу для детальнішого аналізу та порівняння вибраних інструментів. Серед обмежень, пов'язаних із вибором середовища розробки, варто відзначити використання 2-D графіки та фізики.

Насамперед, прототип гри повинен показати можливості фізичного двигуна та обробки дій користувача. Під час аналізу слід враховувати співвідношення отриманого результату зі складністю його реалізації.

По-друге, нам потрібно продемонструвати роботу з графікою, обробку анімації, ефекти тощо. Для подальшого аналізу кожен прототип буде побудований на тих самих графічних матеріалах.

По-третє, потрібно виділити роботу з логікою гри, її взаємодію зі сценою, предметами. Таким чином, особливу увагу слід приділити аналізу складності реалізації логіки у різних частинах ігрового проекту.

Нарешті, необхідно проаналізувати можливості взаємодії гравця із грою, тобто. підтримка периферії та її обробки, робота з елементами графічного інтерфейсу та ін.

Виходячи з визначення проблеми та обмежень, приступаємо до опису методів, що були реалізовані у проекті:

1. Метод «CreateBoard» – призначений для створення дошки;
2. Метод «FindNullTiles» – призначений для знаходження порожніх плиток;
3. Метод «ShiftTilesDown» – призначений для зсуву всіх плиток вниз;
4. Метод «GetNewSprite» – призначений для того, щоб получити новий спрайт;
5. Метод «OnMouseDown» – призначений для обробки події натискання лівої кнопки миші;
6. Метод «SwapSprite» – призначений для зміни місцями сусідніх плиток при натисканні лівої кнопки миші;
7. Метод «GetAdjacent» – призначений для повернення сусідніх плиток відносно зміненої;
8. Метод «GetAllAdjacentTiles» – призначений для повернення всіх плиток;
9. Метод «FindMatch» – призначений для повернення всіх збігів
10. Метод «ClearMatch» – призначений для очистки збігів в одному ряді;
11. Метод «PingPongSize» – призначений для зміни розміру масштабу;
12. Метод «LoadScene» – призначений для завантаження сцени;

13. Метод «ReloadScene» – призначений для перезавантаження поточної сцени;

14. Метод «Load» – призначений для асинхронного завантаження сцени із зазначеним рядком;

15. Метод «ActivateScene» – призначений для змінювання сцени після її завантаження;

16. Метод «CurrentSceneName» – призначений для отримання назви поточної сцени;

17. Метод «ExitGame» – призначений для виходу з ігри;

18. Метод «GameOver» – призначений для показу заставки, після закінчення гри, а саме повернення на початкову заставку;

19. Метод «WaitForShifting» – призначений для очікування зміщення плиток;

20. Метод «PlaySFX» – призначений для включення фонової музики гри.

2.6 Висновки до розділу

У другому розділі розповідається про технічне завдання. Також описані середовище та системи які потрібні для створення власної гри. Розповідається про засоби розробки та які необхідні вимоги до технічного забезпечення. Також розповідається про класи які використовувалися у роботі та методи якими користувалися для розробки ігор.

Ми розглянули найпопулярніші сучасні інструменти для розробки ігор, розділили їх на класи та проаналізували їхній базовий заявлений функціонал. Для детальнішого аналізу виберемо по одному з найкращих представників кожного класу для розробки прототипу гри.

Серед ігрових рушіїв для себе ми обрали Unity. Він повністю задовольняє нас своїми основними характеристиками, також має сильну підтримку з боку компанії, що значно знижує поріг входу.

РОЗДІЛ 3. ЕТАПИ РОЗРОБКИ ТА СТВОРЕННЯ ПРОГРАМНОГО ДОДАТКУ

3.1 Опис розроблювального алгоритму

Завдяки бурхливому розвитку науки інформатики і проникненню їх у різні галузі народного господарства слово " алгоритм " стало найпоширенішим і найбільш вживаним у життєвому плані поняттям для широкого кола фахівців. Більше того, з переходом до інформаційного суспільства алгоритми стають одним з найважливіших факторів цивілізації.

Відомо, що математична теорія алгоритмів склалася зовсім у зв'язку з бурхливим розвитком інформатики та обчислювальної техніки, а виникла надрах математичної логіки на вирішення її проблем. Вона, перш за все, дуже вплинула на світогляд математиків і їх науку.

Тим не менш, взаємовплив теоретичних областей, пов'язаних з обчислювальною технікою, та теорії алгоритмів також, безсумнівно.

Теорія алгоритмів вплинула на теоретичне програмування. Зокрема, велику роль теоретичному програмуванні відіграють моделі обчислювальних автоматів, які, за суттевістю, є обмеженнями тих представницьких обчислювальних моделей, створених раніше у теорії алгоритмів. Трактування програм, як об'єктів обчислення, оператори, що використовуються для складання структурованих програм (послідовне виконання, розгалуження, повторення), прийшли в програмування з теорії алгоритмів. Зворотний вплив виявилося, наприклад, у тому, що виникла потреба у створенні та розвитку теорії обчислювальної складності алгоритмів. Отже, можна сказати, що теорія алгоритмів застосовується у інформатиці, а й у інших галузях знань.

Поняття алгоритму та його властивості

Слово «алгоритм» означає «процес або набір правил, яких слід дотримуватися під час обчислень або інших операцій з вирішення проблем». Тому Алгоритм відноситься до набору правил/інструкцій, які крок за кроком визначають, як має виконуватися робота, щоб отримати очікувані результати.

Це можна зрозуміти, взявши приклад приготування за новим рецептом. Щоб приготувати новий рецепт, потрібно читати інструкції та кроки та виконувати їх по черзі в заданій послідовності. Отриманий таким чином результат – нове блюдо, приготоване ідеально. Аналогічно, алгоритми допомагають виконати завдання в програмуванні, щоб отримати очікуваний результат.

Розроблений алгоритм не залежить від мови, тобто це прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Оскільки не слід дотримуватись жодних письмових інструкцій, щоб приготувати рецепт, а лише стандартний. Так само не всі письмові інструкції з програмування є алгоритмами. Для того, щоб деякі інструкції були алгоритмом, вони повинні мати такі характеристики:

Чітко і однозначно: алгоритм повинен бути зрозумілим і недвозначним. Кожен із його кроків має бути зрозумілим у всіх аспектах і вести лише до одного значення.

Добре визначені вхідні дані: якщо алгоритм каже приймати вхідні дані, це повинні бути чітко визначені вхідні дані.

Добре визначені результати: алгоритм повинен чітко визначити, який вихід буде отримано, і він також повинен бути чітко визначений.

Скінченність: Алгоритм повинен бути скінченним, тобто він не повинен опинитися в нескінченних циклах або тому подібних.

Можливість: Алгоритм повинен бути простим, загальним і практичним, щоб його можна було виконувати за наявності доступних ресурсів. Він не повинен містити якусь майбутню технологію чи щось таке.

Незалежний від мови: розроблений алгоритм повинен бути незалежним від мови, тобто це повинні бути просто прості інструкції, які можна реалізувати будь-якою мовою, але результат буде таким же, як і очікувалося.

Переваги алгоритмів:

- це легко зрозуміти;
- алгоритм — це поетапне представлення рішення заданої задачі;
- в алгоритмі проблема розбивається на менші частини або кроки, отже, програмісту легше перетворити її на справжню програму.

Недоліки алгоритмів:

- написання алгоритму займає багато часу, тому займає багато часу.
- розгалуження та циклічні оператори важко показати в алгоритмах.

Як розробити алгоритм?

Для того, щоб написати алгоритм, необхідні такі речі як передумова:

- задача, яку потрібно вирішити за допомогою цього алгоритму.
- обмеження проблеми, які необхідно враховувати при розв'язуванні задачі;
- вхідні дані, які потрібно зробити для вирішення проблеми;
- результат, якого слід очікувати, коли проблема буде вирішена;
- рішення цієї задачі, в заданих обмеженнях;

Потім алгоритм записується за допомогою вищевказаних параметрів так, щоб він вирішував задачу.

Приклад: Розглянемо приклад, зсуву всіх плиток вниз відносно аннігельованого рядка.

Крок 1: Виконання попередніх умов

Як обговорювалося вище, для того, щоб написати алгоритм, повинні бути виконані його передумови.

Задача, яку потрібно розв'язати за цим алгоритмом: зсунути всі плитки на ігровій дошці.

Обмеження задачі, які необхідно враховувати при розв'язуванні задачі: зсунутись повинні тільки ті плитки, що знаходяться вище аннігельованих плиток.

Вхідні дані, які потрібно зробити для розв'язання задачі: позиція по координаті x, висота по yStart, затримка міщення shiftDelay.

Вихід, якого слід очікувати, коли задачу буде розв’язано: будуть зсунуті всі плитки відносно аннігельованих.

Розв’язання цієї задачі за наведених обмежень: Рішення складається із визначенням позиції рядка, який було аннігельовано, а потім зсув плиток відносно цього рядка.

Крок 2: Розробка алгоритму

Тепер розробимо алгоритм за допомогою наведених вище умов:

Алгоритм визначення плиток, які необхідно знищити, та відносно них зсунути верхні плитки:

СТАРТ

Передати на вхід методу: координати по осі x, де буде знищуватись рядок; початок по координаті yStart та затримку зсуву shiftDelay.

Визначення всіх однакових плиток (не менше 3).

Знищення всіх однакових плиток, які можуть йти зліва на право або згори вниз (кількість має бути не менше 3-хо однакових плиток).

Зсув всіх решту плиток відносно знищених .

КІНЕЦЬ

Крок 3: Тестування алгоритму шляхом його реалізації.

3.2 Розробка власного продукту (гри)

3.2.1 Розробка користувацького інтерфейсу

Користувальницький інтерфейс у середовищі комп'ютерних розробок носить назву UI. UI - це все інформаційне середовище, яке відображається на екрані, і з якою взаємодіє гравець. Раніше для створення UI вимагалося писати велику кількість скриптів. На даний момент створена об'єктна модель створення інтерфейсів за допомогою візуалізації.

Для створення будь-якого UI елемента в середовищі Unity необхідно клікнути правою кнопкою миші у вікні ієрархії, вибрати UI, і, відповідно, вибрати необхідний графік. На рисунку 9 представлені види UI в Unity.

Будь-який графічний елемент створюється дочірнім до об'єкта Canvas. Об'єктна модель створення інтерфейсу в Unity підсумовує, що всі елементи повинні бути дочірніми до Canvas.

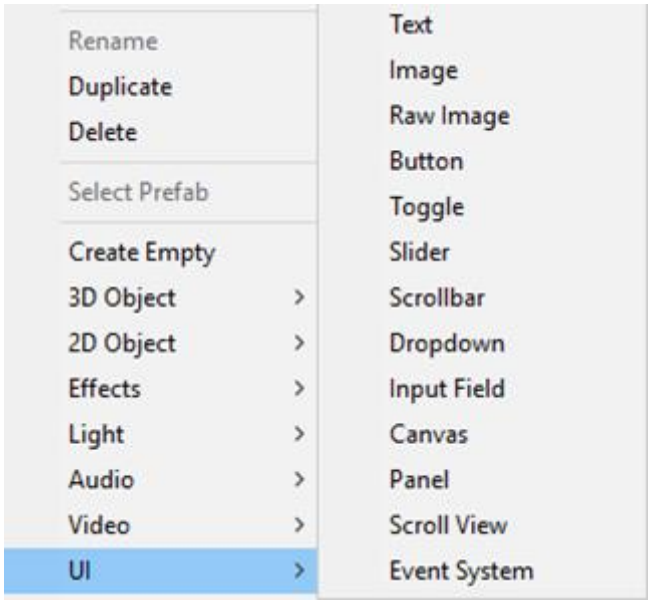


Рисунок 3.1 – UI в Unity

Будь-який графічний елемент створюється дочірнім до об'єкта Canvas. Об'єктна модель створення інтерфейсу в Uniti підсумовує, що всі елементи повинні бути дочірніми до Canvas.

Canvas (Полотно)

Canvas - це компонент, який відображає графічні елементи (картинки, текст, кнопки і т.д.). Canvas зображений на рисунку 3.2.

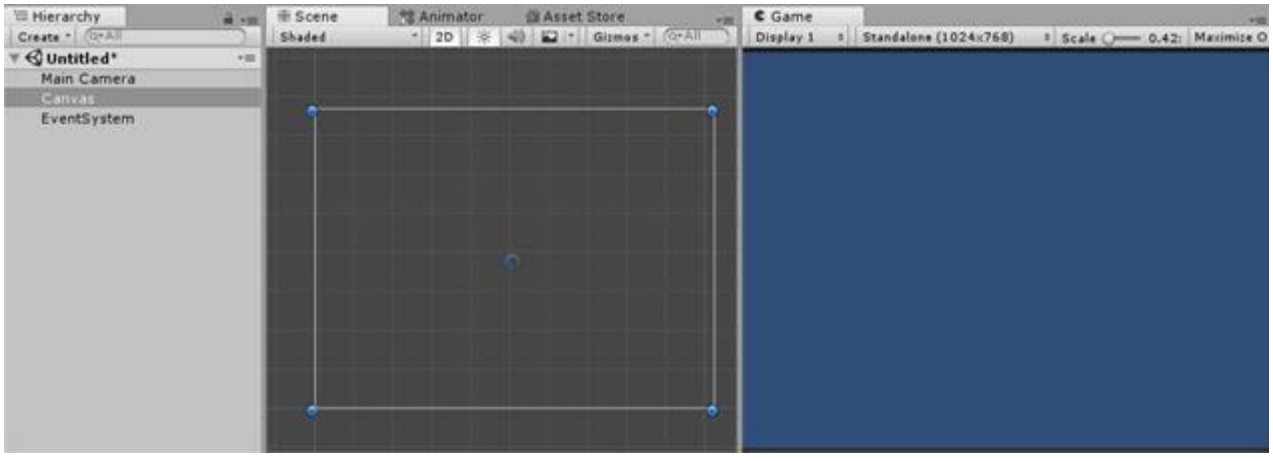


Рисунок 3.2 – Canvas

Canvas автоматично змінюється в залежності від дозволу екрану. Властивості Canvas зображені на рисунку 3.3.

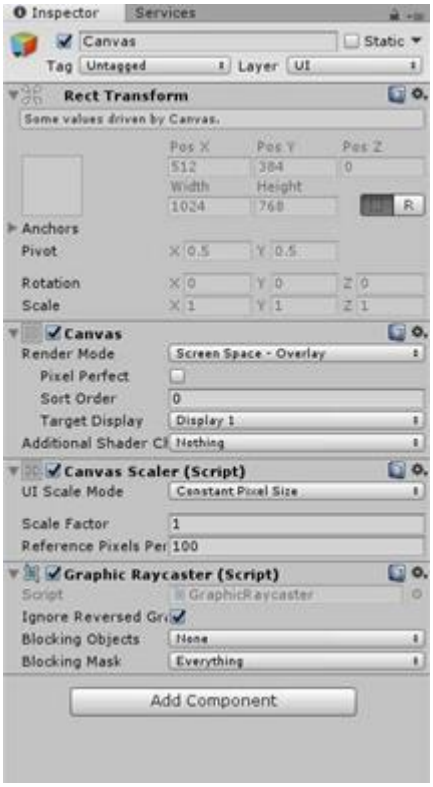


Рисунок 3.3 – Властивості Canvas

Render Mode (рис. 3.4) - своєрідність, яка дає можливість налаштувати, як графічні елементи, що знаходяться всередині Canvas, повинні відповідати.

- Screen Space - Overlay – всі графічні елементи відображаються поверх ігрової сцени. У цьому режимі розміри полотен автоматично підстраюються під розміри екрану.
- Screen Space - Camera - всі графічні елементи відображаються за допомогою камери. Для цього потрібно помістити елемент Camera в свій Render Camera. Змінюючи налаштування камери, можна впливати на зовнішній вигляд Canvas.
- World Space – всі UI знаходяться в 3D просторі і вважаються

звичайними 3D об'єктами.

-

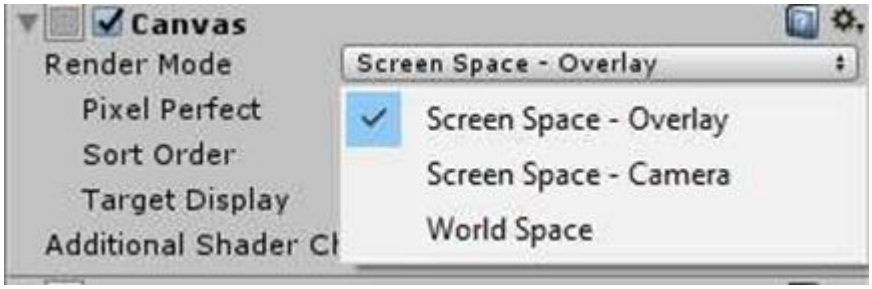


Рисунок 3.4 – Render Mode

Будь-який створений на Canvas графічний елемент можна прив'язати до однієї з дев'яти крапок (до кутів, серединам сторін, або до центру екрану). Для цього в панелі Rect Transform (рис. 3.5) використовується Anchor Presets (рис. 3.6).

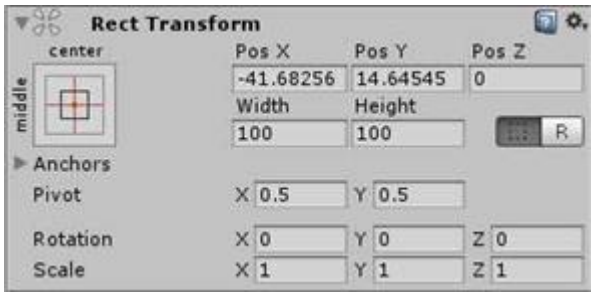


Рисунок 3.5 – Rect Transform

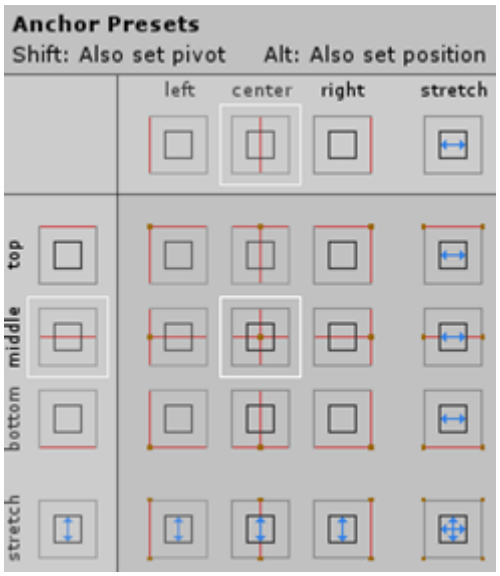


Рисунок 3.6 – Anchor Presets

Кординати розміщення елемента будуть відчитуватися від крапки, яка обрана як «якоря». Також у будь-якого елемента в панелі React Transform є своєрідність Pivot. Це та крапка на самому елементі, від якої буде відчитуватися її координати (рис. 3.5).

Text. Представляє собою елемент, що відображає текст, який можна поставити в спеціальному полі або в скриптах.

Image. За допомогою цього елемента можна розмістити на екрані будь-яке зображення. За замовчуванням об'єкт малюнки створюється завжди одного і того ж розміру, але за допомогою кнопки Set Native Size в інспекторі Image можна підігнати розміри зображення під розміри проекту.

Button. Кнопка є складовим елементом, тому текст всередині кнопки редагується в дочірньому до кнопки елементу Текст. На кнопки можна натискати і обробляти це натискання. На компоненті Button є властивість, що дозволяє наполягати ті функції, які будуть оброблятися за натисканням на кнопки. Щоб кнопка розуміла, яку функцію викликати, ця функція описується в окремому скрипті, при цьому функція повинна бути кермом. Події можна насторожувати як на натискання кнопки (On Click ()), так і на відпускання кнопки (додатковий компонент Event Trigger).

Toggle. Це складний елемент, який складається з кількох малюнків і тексту. Текст вказується в дочірньому елементі Текст, а зображення вказується в елементі Image. Даний елемент має тільки одне подія, яке спрацьовує при зміні стану цього об'єкта.

Scrollbar. Майже теж більше, що Slider, тільки використовуються в складах більше складних компонентів, щоб взаємодія з Scrollbar давала якийсь ефект.

Dropdown. Даний елемент є складним і представляє собою розкривний список. Елементи список задаються в налаштуваннях самого компонента.

Input Field. Представляє собою текстове поле, в яке користувач може вносити які-небудь дані. У цього елемента достатньо багато робіт, основні з них:

- зміна тексту про прохання ввести будь-яку інформацію (дочірній

елемент Placeholder);

- вміст type - своєрідність представляє собою розкривається список, який містить кілька елементів (рис. 3.7). Вибравши один з них можна поставити символи, які можна написати всередину Input Field (цілі числа, букви і т.д.);
- можливість коректування миготливого коркопа. Налаштувати можна часто миготіння, товщини, колір;
- елемент може бути одностроковим або багатостроковим (дається можливість при натисканні клавіші Enter перейти на інший рядок).

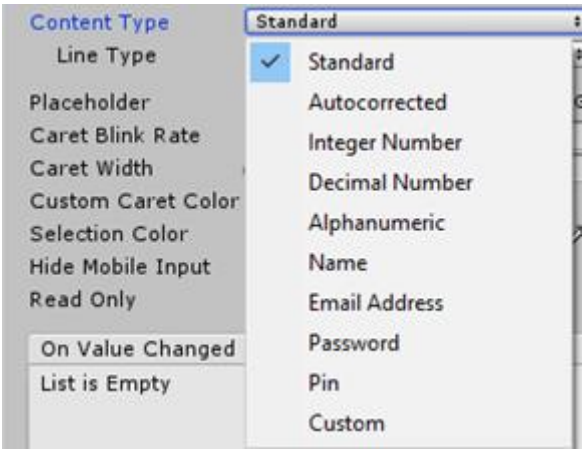


Рисунок 3.7 – Content type

Panel. Представляє собою підкладку, на яку можна помістити будь-якій інший UI.

3.2.2 Опис класу «BoardManager»

Для початку було розроблено клас BoardManager для відображення ігрової дошки та методів, які опрацьовують ігрові дії. Скелет класу BoardManager приведений в лістингу 3.1.

Лістинг 3.1. Скелет класу «BoardManager»

```
public class BoardManager : MonoBehaviour {
    public static BoardManager _Instance;
    public List<Sprite> _Characters = new List<Sprite>();
```

```
        public GameObject _Tile;
        public int _xS, _yS;
        private GameObject[,] _Tiles;
        public bool IsShiftings {
get {}; set{};
}
//Метод, який обов’язково буде викликаний
        void Start () {
                }
// Метод для створення дошки
        private void CreateBoards (float XOff, float YOff) {
   }
//Метод для знаходження порожніх плиток
        public IEnumerator FindNullTiles() {
        }
//Метод для зсуву всіх плиток вних
        private IEnumerator ShiftingsTileDown (int X, int YS, float ShiftDelay =
.04f) {
}
```

3.2.3 Створення додатку під ОС IOS

Після написання коду та детального його тестування можна приступити до побудови додатку під операційну систему IOS. Для цього перейшовши по меню «File» -> «Build & Run» середовища Unity 5.6.4 (в якому було розроблено ігру) відкриється вікно для побудови проекту під будь-яку платформу (рис. 3.8).

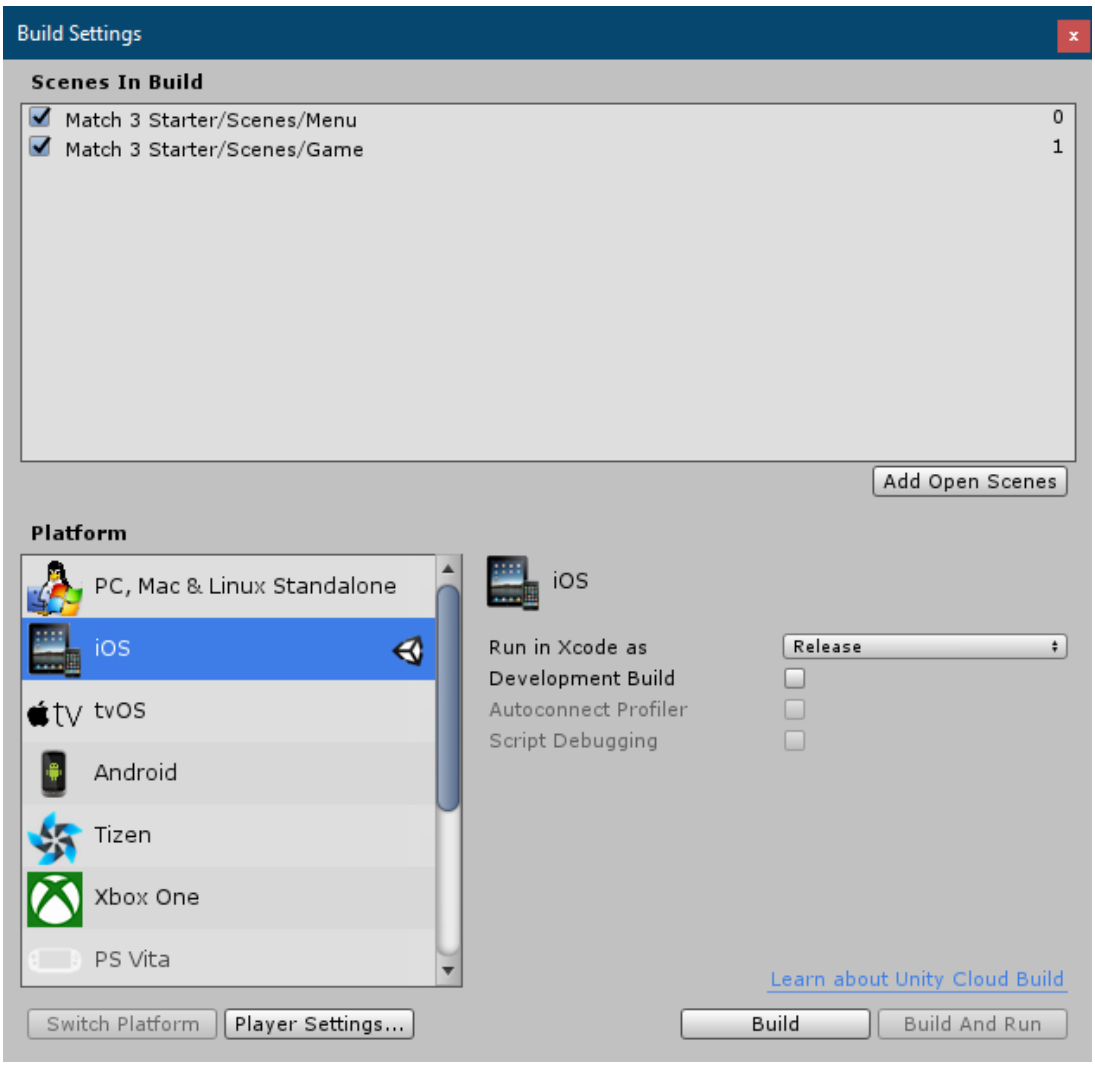


Рисунок 3.8 – Вікно для створення додатку під різні платформи

Ділі необхідно вибрати платформу, на якій буде створено білд та натиснути кнопку «Player Settings» для детального налаштування параметрів гри (рис. 3.9).

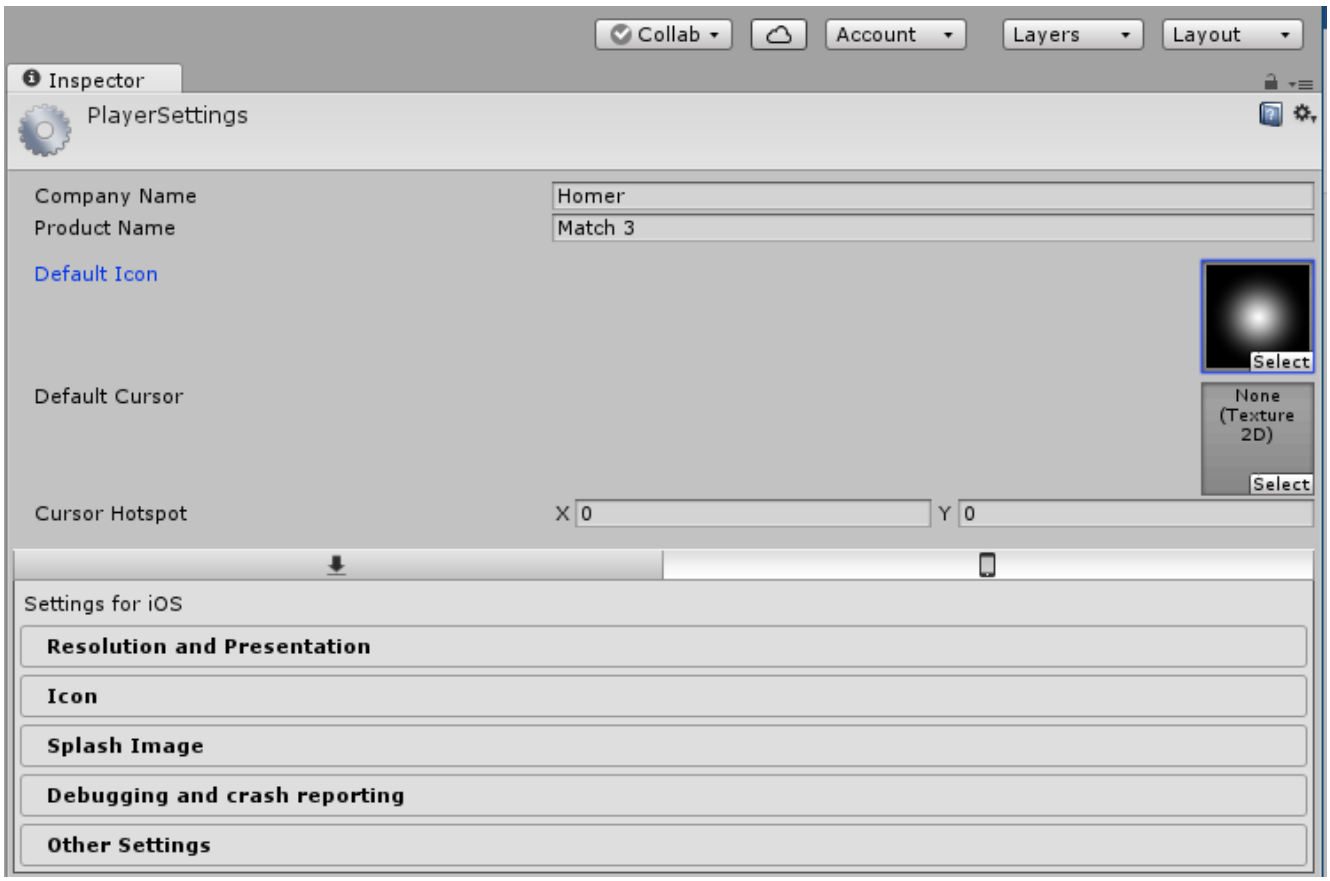


Рисунок 3.9 – Вікно «Player Settings

В поле «Product Name» – необхідно задати назву гри, а в поле «Company Name» – задати назву компанії, якщо потрібно. Також тут можна задавати і інші параметри, такі як: іконку гри, курсор по замовчуванню та багато іншого.

Після задання всіх параметрів необхідно натиснути кнопку «Bild», після чого з’явиться вікно для вибору папки, де буде створено гру під вибрану платформу (рис. 3.10).

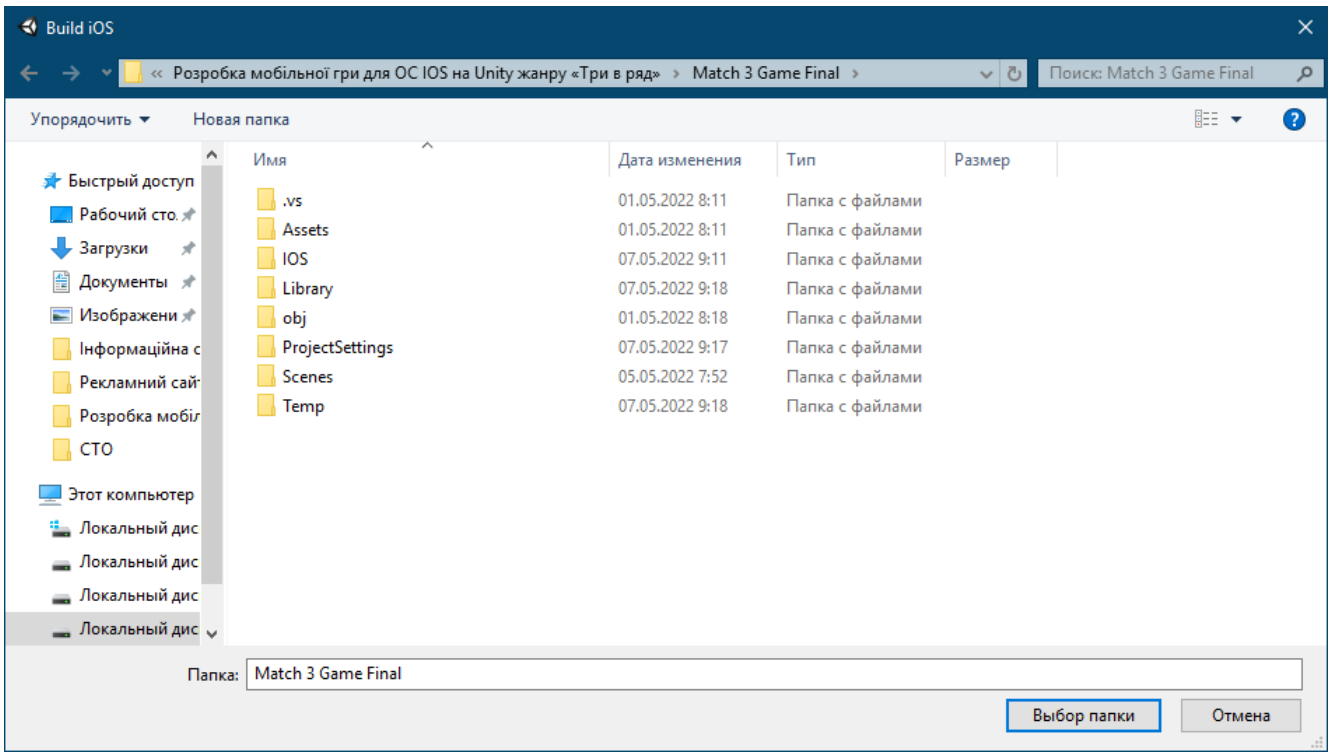


Рисунок 3.10 – Вибір папки для створення білду

Після вибору папки, буде створено необхідний білд.

3.3 Опис функціоналу

Заходячи до гри, перед нами з’являється головне меню гри (рис. 3.11)

Рисунок 3.11 - Початок гри

Після натиснення кнопки «Play» розпочнеться гра (рис. 3.12).

Рисунок 3.12 – Початок гри

У лівій нижній частині екрану розташована статистика кількості спроб перестановок плитки, а в правій нижній частині – кількість набраних очок гравцем. За одну знищену плитку нараховується 50 очок, та кількість однакових плиток повинна бути не менше трьох штук, кількість спроб -99.

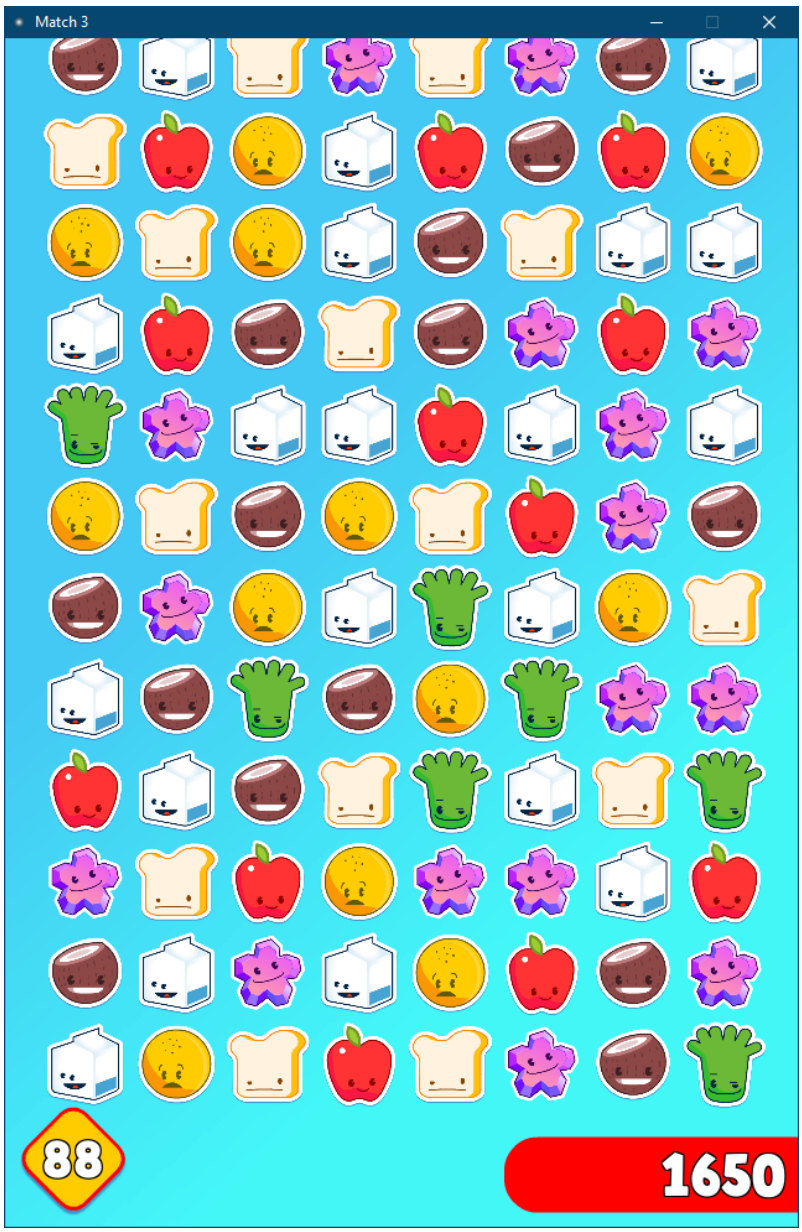


Рисунок 3.13 – Нарахування кількості очок за знищені плитки

Як можна побачити кількість спроб зменшується, а кількість набраних очок гравцем збільшується. Якщо ж спроби дійдуть до 0, система виведе вікно з результатом, яке показано на рисунку 3.14.

Рисунок 3.14 – Виведення вікна із результатом

Після виведення результату, можна розпочати гру заново, або ж перейти на початок гри та натиснути вихід (рисунок 3.11).

3.4 Тестування

Тестування — це процес аналізу та дослідження, який надає змогу виявити інформацію про якість продукту відносно умов, в яких він буде застосовуватись. Методика тестування також включає в себе процес пошуку дефектів, помилок, несправностей. Також це є випробуванням програмних складових з метою оцінити готовність програмного продукту до використання. Результат тестування оцінюється за наступними критеріями:

— відповідність вимогам, які надавалися розробниками та проектувальниками;

— відповідність вихідних даних;

— прийнятний час виконання функцій;

— практичність;

— відповідність вимогам замовника.

Кількість тестів навіть для простих програмних компонентів може бути ледь не нескінченним, тому тактика тестування має полягати в тому, що будуть проведені тільки необхідні тести з урахуванням доступного часу та ресурсів. Як результат, програмні засоби тестуються звичайним виконанням програми з метою виявити баги, помилки або інші дефекти.

Існує багато видів тестування: деякі види тестування виконуються самими розробниками, а інші види тестування виконують спеціальні групи. В нашому ж випадку буде використовуватись тестування системи.

Тестування системи — це процес виконання запущеного програмного забезпечення в його остаточній конфігурації.

Одним із способів вивчення поставленого питання є дослідження методики «чорної скриньки», Ці тести допоможуть зрозуміти:

— Виконання всіх функцій розробленого продукту.

— Прийом вихідних даних.

— Формування результатів.

— Зберігання цілісності зовнішньої інформації.

Тестування програмних засобів буде доречно проводити використовуючи методику «чорної скриньки».

Вона базується на використанні шаблонів тестування, бо ж тест-кейсів. Це означає що буде створено декілька ситуацій у яких перевіряється працездатність додатку, коректності основних функцій.

Розроблені тест-кейси описано в таблиці 3.1.

Таблиця 3.1 – Тест-кейси

<table>
<tr><td rowspan="3">Код тест-кейса</td><td colspan="3">Опис тест-кейса</td></tr>
<tr><td colspan="2">Хід тестування</td><td>Очікуваний результат</td></tr>
<tr><td>Дата тестування</td><td>Результат</td><td>Примітка</td></tr>
<tr><td rowspan="3">001</td><td colspan="3">Перевірка запуску гри «3 match»</td></tr>
<tr><td colspan="2">1. Запустити додаток</td><td>Вивід відповідного вікна із функціоналом для запуску гри</td></tr>
<tr><td>07.05.2022</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">002</td><td colspan="3">Розпочати ігровий процес гри «3 match»</td></tr>
<tr><td colspan="2">1. Запустити додаток;
2. Натиснути кнопку «Play»;</td><td>Після натискання кнопки «Play» відображається головна дошка гри, на якій розташовані плитки.</td></tr>
<tr><td>07.05.2022</td><td>Пройдено</td><td>-</td></tr>
</table>

Продовження таблиці 3.1

003	Поміняти місцями плитки у грі «3 match»		
	1. Запустити додаток; 2. Натиснути кнопку «Play» ; 3. Поміняти місцями плитки	Плитки успішно міняються місцями, якщо їхня кількість рівна 3 або більше, вони зникають та нараховуються кількість набраних очок.	
	07.05.2022	Пройдено	-
004	Пройти гру «3 match» до кінця		
	1. Запустити додаток; 2. Натиснути кнопку «Play»; 3. Використати всі спроби, яких є 99	Після використання всіх спроб перестановок, гра успішно завершується, після чого виводяться всі її результати.	
	07.05.2022	Пройдено	-
005	Вийти із гри «3 match»		
	1. Запустити додаток; 2. Натиснути кнопку «Play»; 3. Використати всі спроби, яких є 99; 4. Вийти в головне меню; 5. Вийти із програми	Після виходу в головне меню та натисканні кнопки «Exit» програма успішно закривається.	
	07.05.2022	Пройдено	-

3.5 Інструкція користувачу

Дана гра «3 match» є досить непоганою як для свого жанру, оскільки дозволяє гравцю розслабитись.

На початку необхідно запустити додаток «3 match». Після запуску додатку буде виведено вікно, в якому можна буде розпочати гру натиснувши кнопку «Play» або «Exit» для виходу із гри..

Після запуску гри, гравцю дається 99 спроб на перестановку плиток. Плитки необхідно розташувати таким чином, щоб кількість однакових була не менше 3-ох штук. Після використання 99 спроб буде виведено результат гри у вигляді кількості набраних очок та можливості розпочати гру заново, щоб покращити свій результат.

3.6 Висновки до розділу 3

В цьому розділі розповідається про етапи розробки та створення програмного додатку. Описується приклад одного із розробленого алгоритму за яким відбувалося написання програмного коду. Розповідається про розробку структури даних та описується функціонал програми. Також розповідається як відбувалося тестування, та які помилки виникали під час тестування. Закінчується розділ написанням інструкції для користувача програми.

ВИСНОВКИ

Побудова сучасних ігор займає дуже багато часу. Вона починається з аналізу предметної області, детального планування системи, описання вимог, моделювання її поведінки за допомогою різних інструментів. Визначається шаблони, які будуть використовуватись при розробці.

Після планування починається стадія розробки. Розробка починається із вибору інструментів розробки та закінчується тестуванням розробленого продукту.

При розробці слід враховувати, що вимоги змінюються швидко і потрібно будувати систему так, щоб вона була гнучкою.

Розробка системи виконується по окремим компонентам. Кожний створений компонент потрібно детально тестувати, щоб мінімізувати помилки на наступних етапах розробки.

Якщо дотримуватися всіх вищеперерахованих вимог, то можна побудувати гнучку і стійку систему.

У першій частині роботи було детальне вивчення предметної області проведено аналітичний огляд існуючих аналогів ігор в жанрі «3 в ряд». Перед розробкою гри було проведено огляд існуючих рушіїв, розглянуто їх переваг та виявлення недоліків. Після чого, процес розробки гри було розбито на етапи, а також проаналізовано дві основні мови програмування, які найкраще підходять для розробки ігор.

У другій частині роботи насамперед було описано технічне завдання для розробленого продукту та описано середовище, яке було вибрано для розробки гри. Проведено опис засобів розробки та вимог до сучасних ігор. Після розробки гри було зроблено опис архітектури проекту.

У третій частині проведено опис етапів розробки та створення програмного додатку. Проведено опис функціоналу та тестування додатку. Після цього розроблено інструкцію користувача програми Результати тестування додатку були успішними, помилок не було виявлено.

В результаті проведеної роботи було розроблено гру жанру «три в ряд» під операційну систему IOS. У процесі вирішення завдання розроблено інженерну методику для розробки ігор даного жанру.

У ході роботи отримано такі основні наукові та практичні результати:

1. Розроблено гру, призначену для того, щоб гравець зміг розслабитись та отримати позитивні емоції.

2. Дана гра поки що ніде не використовується. Розроблені методи для гри жанру «три в ряд» можуть бути використані для широкого класу завдань, тому можливий подальший розвиток розробленого продукту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Why the Match 3 games are so popular [Електронний ресурс]. - Режим доступу: https://gamingonphone.com/editorial/why-the-match-3-puzzle-games-are-so-popular/

2. Chain shot game [Електронний ресурс]. - Режим доступу: https://en.wikipedia.org/wiki/SameGame

3. Sensor Tower, King's Candy Crush [Електронний ресурс]. - Режим доступу: https://app.sensortower.com/android/us/king/app/candy-crush-saga/com.king.candycrushsaga/overview

4. Silver Tale [Електронний ресурс]. - Режим доступу: https://www.gametop.com/news/10-best-match-3-games-in-2020/#/silver-tale/

5. Гра Fishdom: Depths[Електронний ресурс]. - Режим доступу: https://www.playoholic.com/fishdom-deep-dive-tips-tricks/

6. Гра Call of Atlantis Game [Електронний ресурс]. - Режим доступу: https://www.gameyum.com/other-puzzle-games/72240-call-of-atlantis-game-helpful-hints-and-tips/

7. Game engine Unity [Електронний ресурс]. - Режим доступу: https://unity.com/our-company

8. Рушій Unreal Engine [Електронний ресурс]. - Режим доступу: https://ru.education-wiki.com/5231353-what-is-unreal-engine

9. Ігровий рушій CryEngine [Електронний ресурс]. - Режим доступу: https://vlab.fandom.com/ua/wiki/CryEngine

10. Етапи розробки ігор [Електронний ресурс]. - Режим доступу: https://moluch.ru/archive/242/55992/

11. Мова програмування C# [Електронний ресурс]. - Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/

12. Мова програмування C++ [Електронний ресурс]. - Режим доступу: https://ravesli.com/uroki-cpp/

13. Єдність у дії. Багатоплатформна розробка на C#. - М .: Петро, 2018 .-- 608 с.

14. Арстанова, Л.Г. Заняття та розваги з дітьми старшого дошкільного віку. Розробка занять, бесід, ігор та розваг на моральні теми / Л.Г. Арстанов. - М .: Учитель, 2017 .-- 324 с.

15. Архангельська, М. Д. Діловий етикет, або гра за правилами / М. Д. Архангельська. - М .: Ексмо, 2015 .-- 160 с.

16. Вакуленко, Ю.А. Весела граматика. Розробка занять, завдань, ігор / Ю.А. Вакуленко. - М .: Учитель, 2017 .-- 780 с.

17. Джейсон, Фінканон Flash Ads. Розробка мікросайтів, рекламних ігор та фірмових додатків за допомогою Adobe Flash / Finkanon Jason. - М .: Група Рід, 2012 .-- 945 с.

18. Джейсон, Фінканон Flash Ads. Розробка мікросайтів, рекламних ігор і фірмових додатків з pom / Фінканон Джейсон. - М .: ТОВ РІД ГРУП Москва, 2012 .-- 288 с.

19. Любанова, Т.П. Бізнес-план: досвід, проблеми. Зміст бізнес-плану, приклад розробки / Т.П. Любанова, Л.В. Мясоедова, Т.А. Грамотенко та ін .. - М .: Пріор, 2012 .-- 204 с.

20. Паласіос, Хорхе Юніті 5.х. Програмування штучного інтелекту в іграх / Хорхе Паласіос. - М .: ДМК Прес, 2016 .-- 849 с.

21. Платов В.Я. Ділові ігри. Розробка, організація, впровадження. Підручник / В.Я. Платова. - М .: Профиздат, 2010 .-- 192 с.

22. Таран, В.А. Чи легко грати на біржі?! / В.А. ОЗП. - М .: СПб: Петро, 2016 .-- 272 с.

23. Тарп, Ван; Бертон Д.Р. Стратегії обміну. Ігри без ризику / Тарп, Ван; ЛІКАР. Бартон, С. Саггеруд. - М .: СПб: Петро, 2010 .-- 400 с.

24. Фінні, К. 3D-ігри. Все про розвиток (+ CD-ROM) / К. Фінні. - М .: Біном. Лабораторія знань, 2011 .-- 976 с.

25. Фінні, К. 3D-ігри: Все про розробку (+ CD-ROM) / К. Фінні. - М .: Біном. Лабораторія знань, 2015 .-- 133 с.

26. Хорхе, Palacios Unity 5.х. Програмування штучного інтелекту в іграх. Путівник / Паласіос Хорхе. - М .: ДМК Прес, 2017 .-- 427

с.

27. Шабельникова, Є.Ю. Англійська мова. Навчання дітей 5-6 років. Розробка занять, лінгвокультурологічного матеріалу, заходів, ігор / Є.Ю. Шабельникова. - М .: Учитель, 2015 .-- 557 с.

28. Язєв Ю.В. Магія моменту обертання. Мистецтво розробки ігор на движку Torque 2D, включає опис версій 3.2 і 3.3 / Ю. Язєв. - М .: Солон-Прес, 2016 .-- 448 с.

ДОДАТКИ

Додаток А. Презентація до диплому

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

«Розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд»

Виконавець: студент 4 курсу,
групи ПД–42
Іванюк Валерія олександрівна
Керівник роботи: Коба Андрій
Борисович, старший викладач

Київ 2022

Мета, об’єкт та предмет роботи

- ***Мета роботи*** – розробка мобільної гри для ОС IOS на Unity жанру «Три в ряд»
- ***Об'єкт дослідження*** – процес гри жанру «Три в ряд».
- ***Предмет дослідження*** – методи та засоби розробки ігор.

Актуальність роботи

- Популярний жанр
- Сучасний «time killer»
- Розвиток мислення

Аналіз аналогів

ПЕРЕВАГИ:

- Поєднує жанр RPG
- Рівні різної важкості

НЕДОЛІКИ:

- Доступна тільки на ПК
- Платна
- Застарілий інтерфейс
- Присутній донат

SilverTale

Аналіз аналогів

ПЕРЕВАГИ:

- Безкоштовна
- Рівні різної важкості
- Яскравий інтерфейс

НЕДОЛІКИ:

- Присутній донат
- Багато додаткових завдань не пов'язаних з грою
- Довге відновлення життів

Fishdom: Deep Dive

Аналіз аналогів

ПЕРЕВАГИ:

- Безкоштовна
- Цікавий сюжет

НЕДОЛІКИ:

- Складні рівні
- Застарілий інтерфейс
- Присутній донат
- Проходження гри досить швидке

Fishdom: Deep Dive

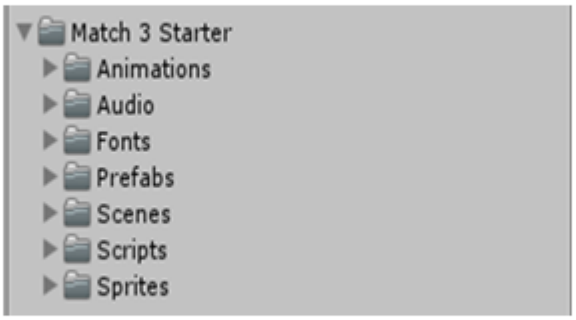
Технічне завдання

- графічний інтерфейс гри повинен відповідати сучасним вимогам, повинен бути простим, щоб не перевантажувати ігровий процес;
- нарахування зароблених очок в залежності від розташування однакових плиток в ряд (після розташованих в один ряд однакових плиток, вони мають анігілюватися, кількість однакових плиток повинна бути мінімум 3);
- ігрова механіка повинна бути інтуїтивно зрозуміла користувачеві;
- приваблива візуалізація для залучення уваги;
- музичний супровід впродовж ігри;
- звуки дій на натискання кнопок миші;
- можливість встановити рекорд;
- можливість розпочати гру заново.

Програмні засоби реалізації

Схема роботи гри

Основна структура проекту



Реалізовані класи

- «BoardManager»
- «Tile»
- «GUISizePingPong»
- «GameManager»
- «GUIManager»
- «SFXManager»

Реалізовані методи

- Метод «CreateBoard»
- Метод «FindNullTiles»
- Метод «ShiftTilesDown»
- Метод «GetNewSprite»
- Метод «OnMouseDown»
- Метод «SwapSprite»
- Метод «GetAdjacent»
- Метод «GetAllAdjacentTiles»
- Метод «FindMatch»
- Метод «ClearMatch»
- Метод «PingPongSize»
- Метод «LoadScene»
- Метод «ReloadScene»
- Метод «Load»
- Метод «ActivateScene»
- Метод «CurrentSceneName»
- Метод «ExitGame»
- Метод «GameOver»
- Метод «WaitForShifting»
- Метод «PlaySFX»

Тестування

Код тест-кейса	Опис тест-кейса		
	Хід тестування		Очікуваний результат
	Дата тестування	Результат	Примітка
001	Перевірка запуску гри «3 match»		
	1. Запустити додаток		Вивід відповідного вікна із функціоналом для запуску гри
	07.05.2022	Пройдено	-
002	Розпочати ігровий процес гри «3 match»		
	1. Запустити додаток; 2. Натиснути кнопку «Play»;		Після натискання кнопки «Play» відображається головна дошка гри, на якій розташовані плитки.
	07.05.2022	Пройдено	-

Тестування

003	Поміняти місцями плитки у грі «3 match»		
	1. Запустити додаток; 2. Натиснути кнопку «Play» ; 3. Поміняти місцями плитки		Плитки успішно міняються місцями, якщо їхня кількість рівна 3 або більше, вони зникають та нараховуються кількість набраних очок.
	07.05.2022	Пройдено	-
004	Пройти гру «3 match» до кінця		
	1. Запустити додаток; 2. Натиснути кнопку «Play»; 3. Використати всі спроби, яких є 99		Після використання всіх спроб перестановок, гра успішно завершується, після чого виводяться всі її результати.
	07.05.2022	Пройдено	-
005	Вийти із гри «3 match»		
	1. Запустити додаток; 2. Натиснути кнопку «Play»; 3. Використати всі спроби, яких є 99; 4. Вийти в головне меню; 5. Вийти із програми		Після виходу в головне меню та натискання кнопки «Exit» програма успішно закривається.
	07.05.2022	Пройдено	-

Процес роботи гри

Процес роботи гри

Процес роботи гри

Апробація результатів

Іванюк В.О. науково-технічна конференція студентів та молодих вчених навчально-наукового інституту інформаційних технологій Державного університету телекомунікацій «Сучасні інфокомунікаційні технології». За результатами апробації опубліковано тезу доповіді «Актуальність гри жанру «Три в ряд» в сучасній ігровій індустрії».

Висновки

1. Було детально досліджено існуючі відеоігри, обгрунтовано актуальність цієї теми та її науково новизну.
2. Проведено аналіз інструменті та програмних засобів реалізації, які найліпше впораються з створенням сучасної мобільної гри.
3. Визначено технічне завдання для розробленої гри, описано вимоги до сучасних мобільних ігор, розроблено UML діаграму діяльності, яка відображає принцип роботи гри.
4. Було повністю реалізоване технічне завдання для гри.

Перспективи подальших досліджень:

- Легка адаптація додатку під ОС Android і Windows Phone.
- Рейтинг користувачів
- Щоденні квести
- Карти різної важкості

Дякую за увагу!

Додаток Б. Лістинги скриптів

```
using Unity engine;
using System. Collections;
using System.Collections. Generic;

public class Board Manager : Mono Behaviour {
        public static Board Manager instance;
        public list <Sprite> characters = new list <Sprite> ( ) ;
        public Game OBject tile;
        public int _ xS, _ yS;

        private Game OBject[,] _Tiles;

        public Bool is ShiFtings { get; set; }

        void Start ( )  {
                instance = Get Component <BoardManager> ( ) ;

                vector2 oFFset = tile.Get Component<Sprite renderer>( ) . Bounds.size;
    Create Board (oFFset.x, oFFset.y) ;
  }

        // Створюємо площадку (дошку)
        private void Create Board (Float XOFF, Float yOFF)  {
                _Tiles = new Game OBject [_ xS, _ yS];

    Float start X = transForm. position. x;
                Float start y = transForm. position.y;

                sprite [ ] previous leFt = new Sprite[_yS]; //  Addd this line
                sprite previous Below = null; //  Addd this line
```

```
                For (int i = 0; i < _ xs; i ++ )  {
                        For ( int j = 0; j < _ yS; j++ )  {
                                Game OBject new TIle = instantIate (tIle, new Vector3 (StartX + (XOFF * I) , Start y + (yOFF * j) , 0) , tIle.transForm.rotatIon) ;
                                _TIles[I, j] = newTIle;
                                new TIle.transForm. parent = transForm;

                                lIst<SprIte> possIBleCharacterpossiBleCharacter. = new lIst<sprIte>( ) ;
                                possIBleCharacter.possiBleCharacter.Adddrange(characters) ;

                                possiBleCharacter.possiBleCharacter.remove(previousleFt[j]) ;
                                possiBleCharacter.possiBleCharacter.remove(previousBelow) ;

                                sprite newSprite = possiBleCharacterpossiBleCharacter.[random.range(0, possiBleCharacter.possiBleCharacter.Count) ];
                                newTile.GetComponent<Spriterenderer>( ) .Sprite = newSprite;
                                previoussleFt[j] = newsprite;
                                previoussBelow = newsprite;
                        }
        }
    }

        // Знайти порожні плитки
        public ie numerator FindNull_Tiles( )  {
                For (int i = 0; i < _ xS; i++ +)  {
                        For (int j = 0; j < _ yS; j++ +)  {
                                iF (_Tiles[I, j].GetComponent<sprIterenderer>( ) .SprIte == null)  {
                                        yield return start Coroutine (shiFt_Tiles Down(i, j) ) ;
                                        Break;
                                }
                        }
                }

                For (int i = 0; i < _ xS; i++ )  {
```

```
                For (int j = 0; j < _yS; j++ )  {
                        _Tiles [i, j].getComponent<Tile>( ) .clearAllMatches( ) ;
                }
        }
}
// Зсунути всі плитки вниз
private ie numerator hIFt_TIlesDown(Int x, Int ystart, Float ShIFtDelay = .03F)  {
        IsShIFtIngs = true;
        lIst<sprIterenderer> renders = new lIst<sprIterenderer>( ) ;
        Int nullCount = 0;

        For (Int j = ystart; j < _yS; y++ )  {
                spriterenderer render = _Tiles[x, j].Getcomponent<spriterenderer>( ) ;
                iF (render.Sprite == null)  {
                        nullCount++ +;
                }
                renders.Addd (render) ;
        }

        For (int i = 0; i < nullCount; i++ )  {
                GUiManager.instance.Score += 50; //  Addd this line here
                yield return new WaitForSeconds(shiFtDelay) ;
                For (int s = 0; s < renders.Count - l; s++ +)  {
                        renders[s].sprite = renders[s + l].sprite;
                        renders [s + l].sprite = GetNewSprite(x, _ yS - l) ;
                }
        }
        isShiFtings = False;
}

// отримуємо новий спрайт
private sprite GetNewsprite(int x, int y)  {
        list <sprite> possiBleCharacter = new list<sprite>( ) ;
        possiBleCharacter.Adddrange(characters) ;
```

```
                iF (x > 0)  {
                        posssiBleCharacter.remove(_Tiles[x - l, y].GetComponent<spriterenderer>( ) .Sprite) ;
                }
                iF (x < _ xS - l)  {
                        possiBleCharacter.remove(_Tiles[x + l, y].GetComponent<Spriterenderer>( ) .sprite) ;
                }
                iF (y > 0)  {
                        possiBleCharacter.remove(_Tiles[x, y - l].GetComponent<Spriterenderer>( ) .sprite) ;
                }

                return possiBleCharacter[random.range(0, possiBleCharacter.Count) ];
        }

}

Using UnItyengIne;
Using system.collectIons;
Using system.collectIons.GenerIc;

public class Boardmanager : monoBehavIour {
public static Boardmanager instance;
public lIst< SprIte > characters = new lIst< SprIte >( ) ;
public GameOBject tIle;
public int _ xS, _ yS;

private Game OBject[ ,] _TIles;

public Bool isShIFtIngs { get; set; }

voId Start ( )  {
Instance = GetComponent( ) ;

Vector2 oFFset = tile.GetComponent( ) .Bounds.size;
```

```
CreateBoard(oFFset.x, oFFset.y) ;
}

// створюємо площадку (дошку)
private void CreateBoard (Float XOFF, Float yOFF)  {
_Tiles = new GameOBject[_ xS, _yS];

Float startX = transForm. pocsition.x;
Float start y = transForm. pocsition.y;

Sprite[ ] previousleFt = new Sprite[_ yS]; //  Addd this line
Sprite previous Below = null; //  Addd this line

For (int i = 0; i<_ xS; i++ +)  {
For (int  j = 0; j< _yS; j++ )  {
Game OBject  newTile =  instantiate( tile, new Vector3 (start X + (XOFF * i ) , starty + (yOFF * j ) , 0) , tile.
trancForm. rotation) ;
_Tiles [i, j] = new Tile;
newTile. transF0rm. parent = transF0rm;

list p0ssiBleCharacter = new list( ) ;
p0ssiбle Character.Addd range(characters) ;

possible Character. remove(previousleFt[j] ) ;
possible Character. remove(previousBelow ) ;

sprite нешsprite = possiBleCharacter[ random.range(0, possiBleCharacter.Count) ];
newTile.GetCom ponent< sprite renderer  > ( ) .sprite = newsprite;
previous leFt[j] = newsprite;
previous Below = newsprite;
                                }
        }
    }
```

```
//  Знайти порожні плитки
public ienumerator FindNullTIles( )   {
        For (Int I = 0; I < _ xS; I++ +)   {
                For (int j = 0; j < _yS; j++ )  {
                        IF (_TIles[I, j].GetComponent<Spriterenderer>( ) .sprIte == null)  {
                                yIeld return StartCoroutIne(ShIFt_TIlesDown(I, j) ) ;
                                 Break;
                        }
                 }
         }

        For (int i = 0; i < _ xS; i++ )  {
                For (int j = 0; j < _yS; j++ )  {
                        _ Tiles[i, j] .GetComponent <Tile >( ) .ClearAllMatches ( ) ;
                 }
         }
 }
// Зсунення всіх плиток вниз
prIvate ienumerator ShIFt_TIlesDown(Int x, Int yStart, Float shIFtDelay = .03F)  {
        IsShIFtIngs = true;
        lIst<SprIterenderer> renders = new lIst<SprIterenderer>( ) ;
        Int nullCount = 0;

        For (Int j = yStart; j < _yS; y++ )  {
                SprIterenderer render = _TIles[x, j ].GetComponent<Spriterenderer >( ) ;
                iF  (render.sprite =+= null)   {
                        nullCount++ +;
                 }
                 renders.Addd(render) ;
         }

         For (int i = 0; i < nyllCount; i++ +)  {
                GUiManager.instance.Score + += 50; //  Addd this line here
                yIeld return new WaItForSeconds(shIFtDelay) ;
```

```
                    For (Int s = 0; s < renders.Count - l; s++ )  {
                        renders[s].sprite = renders[s + l].sprite;
                        renders[s + l].sprite = GetNewSprite(x, _yS - l) ;
                    }
                }
                isShiFtings = False;
            }

            // Отримати новий спрайт
            private Sprite GetNewSprite(int x, int y)  {
                list<Sprite> possiBleCharacter = new list<Sprite>( ) ;
                possiBleCharacter.Adddrange(characters) ;

                iF (x > 0)  {
                    possiBleCharacter.remove(_Tiles[x - l, y].GetComponent<Spriterenderer>( ) .sprite) ;
                }
                iF (x < _ xS - l)  {
                    possiBleCharacter.remove(_Tiles[x + l, y].GetComponent<Spriterenderer>( ) .sprite) ;
                }
                iF (y > 0)  {
                    possiBleCharacter.remove(_Tiles[x, y - l].GetComponent<Spriterenderer>( ) .sprite) ;
                }

                return possiBleCharacter[random.range(0, possiBleCharacter.Count) ];
            }

}
using Unityengine;
using System.Collections;
using System.Collections.Generic;

public class BoardManager : MonoBehaviour {
            public static BoardManager instance;
            public list<Sprite> characters = new list<Sprite>( ) ;
```

```
        public GameOBject tile;
        public int xSize, ySize;

        private GameOBject[,] tiles;

        public Bool isShiFting { get; set; }

        void Start ( )  {
                instance = GetComponent<BoardManager>( ) ;

                Vector2 oFFset = tile.GetComponent<Spriterenderer>( ) .Bounds.size;
    CreateBoard(oFFset.x, oFFset.y) ;
  }

// Створюєм0 дошку
        private void CreateBoard (Float xOFFset, Float yOFFset)  {
                tiles = new GameOBject[xSize, ySize];

    Float startX = transForm.position.x;
                Float starty = transForm.position.y;

                Sprite[] previousleFt = new Sprite[ySize]; //  Addd this line
                Sprite previousBelow = null; //  Addd this line

                For (int x = 0; x < xSize; x++ )  {
                        For (int y = 0; y < ySize; y++ )  {
                                GameOBject newTile = instantiate(tile, new Vector3(startX + (xOFFset * x) ,
starty + (yOFFset * y) , 0) , tile.transForm.rotation) ;
                                tiles[x, y] = newTile;
                                newTile.transForm.parent = transForm; //  Addd this line

                                list<Sprite> possiBleCharacters = new list<Sprite>( ) ;
                                possiBleCharacters.Adddrange(characters) ;
```

```
                possiBleCharacters.remove(previousleFt[y]) ;
                possiBleCharacters.remove(previousBelow) ;

                Sprite newSprite = possiBleCharacters[random.range(0,
possiBleCharacters.Count) ];
                newTile.GetComponent<Spriterenderer>( ) .sprite = newSprite;
                previousleFt[y] = newSprite;
                previousBelow = newSprite;
            }
        }
    }

// Знайти порожнІ плиТки
    public ienumerator FindNullTiles( ) {
        For (int x = 0; x < xSize; x++ ) {
            For (int y = 0; y < ySize; y++ ) {
                iF (tiles[x, y].GetComponent<Spriterenderer>( ) .sprite == null) {
                    yield return StartCoroutine(ShiFtTilesDown(x, y) ) ;
                    Break;
                }
            }
        }

        For (int x = 0; x < xSize; x++ ) {
            For (int y = 0; y < ySize; y++ ) {
                tiles[x, y].GetComponent<Tile>( ) .ClearAllMatches( ) ;
            }
        }
    }

// Зсунути всі плитки вниз
    private ienumerator ShiFtTilesDown(int x, int yStart, Float shiFtDelay = .03F) {
        isShiFting = true;
        list<Spriterenderer> renders = new list<Spriterenderer>( ) ;
```

```
            int nullCount = 0;

            For (int y = yStart; y < ySize; y++ )  {
                Spriterenderer render = tiles[x, y].GetComponent<Spriterenderer>( ) ;
                iF (render.sprite == null)  {
                    nullCount++ ;
                }
                renders.Addd(render) ;
            }

            For (int i = 0; i < nullCount; i++ )  {
                GUiManager.instance.Score += 50;
                yield return new WaitForSeconds(shiFtDelay) ;
                For (int k = 0; k < renders.Count - l; k++ )  {
                    renders[k].sprite = renders[k + l].sprite;
                    renders[k + l].sprite = GetNewSprite(x, ySize - l) ;
                }
            }
            isShiFting = False;
        }

// Получити новий спрайт
        private Sprite GetNewSprite(int x, int y)  {
            list<Sprite> possiBleCharacters = new list<Sprite>( ) ;
            possiBleCharacters.Adddrange(characters) ;

            iF (x > 0)  {
                possiBleCharacters.remove(tiles[x - l, y].GetComponent<Spriterenderer>( ) .sprite) ;
            }
            iF (x < xSize - l)  {
                possiBleCharacters.remove(tiles[x + l, y].GetComponent<Spriterenderer>( ) .sprite) ;
            }
            iF (y > 0)  {
                possiBleCharacters.remove(tiles[x, y - l].GetComponent<Spriterenderer>( ) .sprite) ;
```

```
            }

            return possiBleCharacters[random.range(0, possiBleCharacters.Count) ];
        }

}
using Unityengine;
using Unityengine.Ui;
using Unityengine.SceneManagement;
using System.Collections;

public class GameManager : MonoBehaviour {
        public static GameManager instance;

        public GameOBject FaderOBj;
        public image Faderimg;
        public Bool gameOver = False;

        public Float FadeSpeed = .02F;

        private Color FadeTransparency = new Color(0, 0, 0, .04F) ;
        private string currentScene;
        private AsyncOperation async;

        void Awake( )  {
            //  0дн0часн0 м0же ІснуваТи ТІльки l менеджер Ігри
            iF (instance == null)  {
                DontDestroyOnload(gameOBject) ;
                instance = GetComponent<GameManager>( ) ;
                SceneManager.sceneloaded += OnlevelFinishedloading;
            } else {
                Destroy(gameOBject) ;
            }
        }
```

```
void Update( )  {
        iF (input.GetKeyDown(KeyCode.escape) )  {
                returnToMenu( ) ;
        }
}

//  ЗаванТажуєм0 сцену Із вказан0ю назв0ю рядка
public void loadScene(string sceneName)  {
        instance.StartCoroutine(load(sceneName) ) ;
        instance.StartCoroutine(FadeOut(instance.FaderOBj, instance.Faderimg) ) ;
}

//  Перезавантажуємо поточну сцену
public void reloadScene( )  {
        loadScene(SceneManager.GetActiveScene( ) .name) ;
}

private void OnlevelFinishedloading(Scene scene, loadSceneMode mode)  {
        currentScene = scene.name;
        instance.StartCoroutine(Fadein(instance.FaderOBj, instance.Faderimg) ) ;
}

// Повторюємо прозорість фейдера до l00%
ienumerator FadeOut(GameOBject FaderOBject, image Fader)  {
        FaderOBject.SetActive(true) ;
        while (Fader.color.a < l)  {
                Fader.color += FadeTransparency;
                yield return new WaitForSeconds(FadeSpeed) ;
        }
        ActivateScene( ) ; // Activate the scene when the Fade ends
}

//  Повторюємо прзорість фейдера до 0%
```

```
    ienumerator Fadein(GameOBject FaderOBject, image Fader)  {
        while (Fader.color.a > 0)  {
            Fader.color -= FadeTransparency;
            yield return new WaitForSeconds(FadeSpeed) ;
        }
        FaderOBject.SetActive(False) ;
    }

    //  починаємо асинхронно завантажувати сцену із зазначеним рядком
    ienumerator load(string sceneName)  {
        async = SceneManager.loadSceneAsync(sceneName) ;
        async.allowSceneActivation = False;
        yield return async;
        isreturning = False;
}

    //  дозволяєм змінювати сцену після її завантаження
    public void ActivateScene( )  {
        async.allowSceneActivation = true;
    }

    //  тримати назву поточної сцени
    public string CurrentSceneName {
        get{
            return currentScene;
        }
    }

    public void exitGame( )  {
        //  Якщо ми запускаємо в автономній збірці гри
        #iF UNiTy_STANDAlONe
            //  Quit the application
            Application.Quit( ) ;
        #endiF
```

```
                // Якщо ми запускаємо в редакторі
                #iF UNiTy_eDiTOr
                        // Stop playing the scene
                        Unityeditor.editorApplication.isplaying = False;
                #endiF
        }

        private Bool isreturning = False;
        public void returnToMenu( ) {
                iF (isreturning) {
                        return;
                }

    iF (CurrentSceneName != "Menu") {
                        StopAllCoroutines( ) ;
                        loadScene("Menu") ;
                        isreturning = true;
    }
        }

}
using Unityengine;
using Unityengine.Ui;
using System.Collections;

public class GUiManager : MonoBehaviour {
        public static GUiManager instance;

        public GameOBject gameOverpanel;
        public Text yourScoreTxt;
        public Text highScoreTxt;

        public Text scoreTxt;
```

```
public Text moveCounterTxt;

private int score, moveCounter;

void Awake( )  {
        instance = GetComponent<GUiManager>( ) ;
        moveCounter = 99;
}

//  показуємо гру на панелі
public void GameOver( )  {
        GameManager.instance.gameOver = true;

        gameOverpanel.SetActive(true) ;

        iF (score > playerpreFs.Getint("HighScore") )  {
                playerpreFs.Setint("HighScore", score) ;
                highScoreTxt.text = "New Best: " + playerpreFs.Getint("HighScore") .ToString( ) ;
        } else {
                highScoreTxt.text = "Best: " + playerpreFs.Getint("HighScore") .ToString( ) ;
        }

        yourScoreTxt.text = score.ToString( ) ;
}

public int Score {
        get {
                return score;
        }

        set {
                score = value;
                scoreTxt.text = score.ToString( ) ;
        }
```

```
        }

        public int MoveCounter {
                get {
                        return moveCounter;
                }

                set {
                        moveCounter = value;
                        iF (moveCounter <= 0)  {
                                moveCounter = 0;
                                StartCoroutine(WaitForShiFting( ) ) ;
                        }
                        moveCounterTxt.text = moveCounter.ToString( ) ;
                }
        }
        // Очікування на зміщення
        private ienumerator WaitForShiFting( )  {
                yield return new WaitUntil(( )  => !BoardManager.instance.isShiFting) ;
                yield return new WaitForSeconds(.25F) ;
                GameOver( ) ;
        }

}
using Unityengine;
using System.Collections;

[requireComponent(typeoF(rectTransForm) ) ]
public class GUiSizepingpong : MonoBehaviour {

        public Vector3 minScale = new Vector3(.5F, .5F, .5F) ;
        public Float changeDelay = .03F;
        rectTransForm rectTransForm;
```

```
        void Start( )  {
                rectTransForm = GetComponent<rectTransForm>( ) ;
                StartCoroutine(pingpongSize( ) ) ;
        }

        //  Змінитозмір p0 на minScale, а потім скопіювати його до максимального масштабу
        ienumerator pingpongSize( )  {
                Vector3 startingScale = rectTransForm.localScale;
                Vector3 currentScale = startingScale;

                //  На скільки змінюється масштаб кожної ітерації
                Vector3 changeScale = new Vector3(.0lF, .0lF, .0lF) ;
                Bool shrink = true;

                while (true)  {
                        iF (shrink)  {
                                currentScale -= changeScale;
                                iF (currentScale.x <= minScale.x)  {
                                        shrink = False;
                                }
                        } else {
                                currentScale += changeScale;
                                iF (currentScale.x >= l)  {
                                        shrink = true;
                                }
        }
                        rectTransForm.localScale = currentScale;
                        yield return new WaitForSeconds(changeDelay) ;
                }
        }
}
using Unityengine;
using System.Collections;
```

```
[requireComponent(typeoF(rectTransForm) ) ]
public class GUiSizepingpong : MonoBehaviour {

        public Vector3 minScale = new Vector3(.5F, .5F, .5F) ;
        public Float changeDelay = .03F;
        rectTransForm rectTransForm;

        void Start( )  {
                rectTransForm = GetComponent<rectTransForm>( ) ;
                StartCoroutine(pingpongSize( ) ) ;
        }

        //  змінити розмір на minScale, а потім скопіювати його до максимального масштабу
        ienumerator pingpongSize( )  {
                Vector3 startingScale = rectTransForm.localScale;
                Vector3 currentScale = startingScale;

                //  На скільки змінюється масштаб кожної ітерації
                Vector3 changeScale = new Vector3(.0lF, .0lF, .0lF) ;
                Bool shrink = true;

                while (true)  {
                        iF (shrink)  {
                                currentScale -= changeScale;
                                iF (currentScale.x <= minScale.x)  {
                                        shrink = False;
                                 }
                        }  else {
                                currentScale += changeScale;
                                iF (currentScale.x >= l)  {
                                        shrink = true;
                                }
        }
                        rectTransForm.localScale = currentScale;
```

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


using Unityengine;

public enum Clip { Select, Swap, Clear };

public class SFXManager : MonoBehaviour {
        public static SFXManager instance;

        private AudioSource[] sFx;

        //  Вик0рисТ0вуєм0 для ІнІціалІзацІї
        void Start ( )  {
                instance = GetComponent<SFXManager>( ) ;
                sFx = GetComponents<AudioSource>( ) ;
    }

        public void playSFX(Clip audioClip)  {
        // Включаєм0 музику
                sFx[(int) audioClip].play( ) ;
         }
}

using Unityengine;
using System.Collections;
using System.Collections.Generic;

public class Tile : MonoBehaviour {
        private static Color selectedColor = new Color(.5F, .5F, .5F, l.0F) ;
        private static Tile previousSelected = null;
```

```
        private Spriterenderer render;
        private Bool isSelected = False;

        private Vector2[] adjacentDirections = new Vector2[] { Vector2.up, Vector2.down, Vector2.leFt,
Vector2.right };

        void Awake( )  {
                render = GetComponent<Spriterenderer>( ) ;
    }

        private void Select( )  {
                isSelected = true;
                render.color = selectedColor;
                previousSelected = gameOBject.GetComponent<Tile>( ) ;
                SFXManager.instance.playSFX(Clip.Select) ;
         }

        private void Deselect( )  {
                isSelected = False;
                render.color = Color.white;
                previousSelected = null;
         }

        // оброблюємо подію натискання клавіші
        void OnMouseDown( )  {
                //  Not SelectaBle conditions
                iF (render.sprite == null || BoardManager.instance.isShiFting)  {
                        return;
                }

                iF (isSelected)  { //  is it already selected?
                        Deselect( ) ;
                } else {
```

```
                    iF (previousSelected == null)  { //  is it the First tile selected?
                        Select( ) ;
                    } else {
                        iF (GetAllAdjacentTiles( ) .Contains(previousSelected.gameOBject) )  { //  is it
an adjacent tile?
                            SwapSprite(previousSelected.render) ;
                            previousSelected.ClearAllMatches( ) ;
                            previousSelected.Deselect( ) ;
                            ClearAllMatches( ) ;
                        } else {
                            previousSelected.GetComponent<Tile>( ) .Deselect( ) ;
                            Select( ) ;
                         }
                     }
                }
            }

        //  Міняємо місцями спрайти
        public void SwapSprite(SpriteRenderer render2)  {
            iF (render.sprite == render2.sprite)  {
                return;
            }

            Sprite tempSprite = render2.sprite;
            render2.sprite = render.sprite;
            render.sprite = tempSprite;
            SFXManager.instance.playSFX(Clip.Swap) ;
            GUiManager.instance.MoveCounter--; //  Addd this line here
        }



        //  повернути сусідні плитки
        private GameOBject GetAdjacent(Vector2 castDir)  {
            raycastHit2D hit = physics2D.raycast(transForm.position, castDir) ;
```

```
            iF (hit.collider != null)  {
                    return hit.collider.gameOBject;
            }
            return null;
    }

    //  повертаємо всі сусідні плитки
    private list<GameOBject> GetAllAdjacentTiles( )  {
            list<GameOBject> adjacentTiles = new list<GameOBject>( ) ;
            For (int i = 0; i < adjacentDirections.length; i++ )  {
                    adjacentTiles.Addd(GetAdjacent(adjacentDirections[i]) ) ;
            }
            return adjacentTiles;
    }

    // пошук збігів
    private list<GameOBject> FindMatch(Vector2 castDir)  {
            list<GameOBject> matchingTiles = new list<GameOBject>( ) ;
            raycastHit2D hit = physics2D.raycast(transForm.position, castDir) ;
            while (hit.collider != null && hit.collider.GetComponent<Spriterenderer>( ) .sprite ==
render.sprite)  {
                    matchingTiles.Addd(hit.collider.gameOBject) ;
                    hit = physics2D.raycast(hit.collider.transForm.position, castDir) ;
             }
            return matchingTiles;
     }

    // очищуємо збіги в одному ряді
    private void ClearMatch(Vector2[] paths)  {
            list<GameOBject> matchingTiles = new list<GameOBject>( ) ;
            For (int i = 0; i < paths.length; i++ )  { matchingTiles.Adddrange(FindMatch(paths[i]) ) ; }
            iF (matchingTiles.Count >= 2)  {
                    For (int i = 0; i < matchingTiles.Count; i++ )  {
                            matchingTiles[i].GetComponent<Spriterenderer>( ) .sprite = null;
```

```
                }
                matchFound = true;
            }
        }

        private Bool matchFound = False;
        // 0чищаємо всі збіги
        public void ClearAllMatches( )  {
            iF (render.sprite == null)
                return;

            ClearMatch(new Vector2[2] { Vector2.leFt, Vector2.right }) ;
            ClearMatch(new Vector2[2] { Vector2.up, Vector2.down }) ;
            iF (matchFound)  {
                render.sprite = null;
                matchFound = False;
                StopCoroutine(BoardManager.instance.FindNullTiles( ) ) ;
                StartCoroutine(BoardManager.instance.FindNullTiles( ) ) ;
                SFXManager.instance.playSFX(Clip.Clear) ;
            }
        }

}
```