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 ІГРОВОГО ДОДАТКУ ПЛАТФОРМЕРА «JUMPING SLIME 3D» ЗА ДОПОМОГОЮ UNITY МОВОЮ C#**»

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Овдієнко М. В.
(прізвище та ініціали)
Керівник Садовенко В. С.
(прізвище та ініціали)
Рецензент Зінченко О.В.
(прізвище та ініціали)
Нормоконтроль Трінтіна Н.А.
(прізвище та ініціали)

Київ – 2023

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

« »_____2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Овдієнко Марку Володимировичу
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка ігрового додатку платформера «Jumping Slime 3D за допомогою Unity мовою C#»
Керівник роботи к.е.н, доцент кафедри ІПЗ Садовенко В.С.
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «24» лютого 2023 року №26
2. Строк подання студентом роботи: 01.06.2023
3. Вихідні дані до роботи:

3.1 Положення побудови відеогри;

3.2Методи побудови відеоігор;

3.3Існуючи відеоігри жанру Tower defense;

3.4 Науково-технічна література

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

4.1 Аналіз предметної галузі

4.2 Засоби програмної реалізації

4.3 Проектування програмного забезпечення

4.4 Розробка програмного забезпечення

4.5 Тестування відеогри

4.6 Висновки

5 Перелік демонстраційного матеріалу (назва основних слайдів)

5.1 Титульний слайд

5.2 Мета, об’єкт та предмет дослідження

5.3 Задачі бакалаврської роботи

5.4 Аналіз аналогів

5.5 Концепт гри

5.6 Вимоги до додатку

5.7 Програмні засоби реалізації

5.8 Блок-схема ігрового процесу

5.9 Екранні форми інтерфейсу

5.10 Апробація результатів дослідження

5.11 Висновки

6 Дата видачі завдання «25» лютого 2023року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02-28.02	Виконано
2	Дослідження існуючих аналогів гоночних ігор	28.02-16.03	Виконано
3	Проектування архітектури системи	16.03-25.03	Виконано
4	Розробка гри	25.03-01.04	Виконано
5	Висновки, оформлення роботи	01.04-06.05	Виконано
6	Розробка демонстраційних матеріалів	06.05-15.05	Виконано
7	Попередній захист роботи	16.05-01.06	Виконано
8	Здача роботи	01.06.2023	Виконано

Студент Овдієнко М.В.

(підпис) (прізвище та ініціали)

Керівник роботи Садовенко В.С.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 68с., 13 рис., 16 джерел

Ключові слова: проектування програмного забезпечення, ігровий процес додатку, розробка програмного забезпечення, мови програмування, засоби.

Мета – розширення функціональних можливостей та основних програмних засобів для спрощення ігрової механіки ігрового додатку в жанрі платформер.

Об’єкт – ігровий процес додатку платформера «Jumping Slime 3D» з використанням рушія Unity.

Предмет – програмне забезпечення для реалізації ігрового додатку платформера «Jumping Slime 3D».

Методи дослідження – методи побудови відеоігор, методи структурного аналізу і проектування, методи розробки програмного забезпечення, методи тестування відеоігор, методи верифікації програмного забезпечення.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Провести огляд предметної області.
2. Визначити, що таке розробка відеоігор.
3. Визначити поняття платформера.
4. Оглянути процес створення відеоігор.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

VR - Virtual reality

MMORPG - Massively multiplayer online role-playing game

Roguelike - rogue-подібні ігри; також мандрівні ігри

NPC - Non-Player Character

RPG - Role-Playing Game

AR - augmentedreality

ПК -Персональний комп'ютер\

ФПС - frames per second

ВСТУП

В цифрову епоху комп’ютерні ігри стають не просто розважальним процесом, а використовуються як інструмент маркетингу для збільшення продажів та послуг, для набуття відповідних компетенцій працівників підприємств. Тому розробка ігрового додатку є актуальною та важливою задачею в галузі програмного забезпечення.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Провести огляд предметної області.
2. Визначити, що таке розробка відеоігор.
3. Визначити поняття платформера.
4. Оглянути процес створення відеоігор.

Обрати засоби реалізації програмного забезпечення:

1. Обрати рушій.
2. Обрати мову програмування.
3. Обрати середовище розробки
4. Спроектувати внутрішню будову.
5. Розробити графічний інтерфейс.
6. Розробити основні механіки.
7. Провести тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття відеогри

Бажання грати таке ж давнє, як і саме людство. Для дітей це життєво важливий інструмент, щоб підготувати їх до життя, а для дорослих це спосіб втекти від реальності та отримати кілька моментів розваги для себе. Як і майже всі аспекти життя, за останні десятиліття ігри також перемістилися в цифровий світ. У 2018 році одна третина населення світу регулярно проводила час за відеоіграми на мобільному телефоні, ігровій приставці чи комп'ютері. Тріумфальне зростання відеоігор почалося в 1970-х роках, але корінням воно сягає подій, що відбулися приблизно 20 років тому. Відеогра або комп'ютерна гра — це електронна гра, яка взаємодіє з інтерфейсом користувача для забезпечення візуального зворотн'ого зв'язку.

Відеоігри визначаються відповідною платформою та мають різні типи за типом гри та призначенням. Перші прототипи відеоігор 1950-х і 1960-х років використовували відеовихід мейнфрейма. Першою споживчою відео грою стала аркадна відеогра Computer Space у 1971 році. У 1972 році була випущена популярна аркадна гра Pong і перша домашня консоль Magnavox Odyssey.

Галузь швидко зростала під час золотої доби аркадних відеоігор наприкінці 1970-х і на початку 1980-х років, але ринок відеоігор у Північній Америці впав тому встановили методи і методику розробки та публікації відеоігор, щоб запобігти подібним збоям у подальшому.

Від простих крапок на блідому екрані до кольорових пікселів і гіперреалістичних 3D-пейзажів: завдяки технічним розробкам історія відеоігор переносить геймерів усе глибше у віртуальний світ. Як виглядатимуть відеоігри майбутнього? Одне можна сказати точно: історія триватиме в тому ж стилі. Вже зараз завдяки віртуальній реальності геймери можуть майже повністю зануритися в гру.

Краща роздільна здатність графіки, тактильні контролери та легші пристрої

визначать майбутнє ігрової індустрії.

Ранні приклади включають «Drafts» Крістофера Стрейчі, комп'ютер Nimrod на Британському кінофестивалі 1951 року; OXO, комп'ютерний кросворд, розроблений для EDSAC Олександром С. Дугласом у 1952 році; «Тенніс для двох» Вільямом Хіггінботамом у 1958 році. гру, розроблену в 1961 році, і Spacewar!, написану на DEC PDP-1 в 1961 році студентами MIT Мартіном Грецем, Стівом Расселом і Вейном Вітаненом. Кожна гра має свій інструмент відображення: NIMROD має панель з підсвічуванням для ігор Nima.

Попередні винаходи проклали шлях до виникнення сучасних відеоігор. Ральф Х. Байєр, працюючи в Sanders Associates у 1966 році, розробив систему керування для гри в настільний теніс на телебаченні. Зі схвалення Белла він побудував прототип «коричневої коробки». Сандерс запатентував винахід Бера і ліцензував його для Magnavox, який комерціалізував його як першу домашню консоль в 1972 році.

Термін «відеоігри» був розроблений для того, щоб розрізняти цей тип відеоігор, у які грають на деякому типі відеодисплея, на відміну від телепринтера або подібного пристрою. Це також відрізняється від багатьох портативних відеоігор, таких як Merlin, які часто використовують світлодіодні лампи як індикатори, але не поєднують їх для створення зображень [1].

«Комп’ютерні ігри» також можна використовувати для опису відеоігор, оскільки всі відеоігри за своєю природою вимагають комп’ютерного процесора і в деяких випадках можуть використовуватися як «відеоігри» [1]. Однак термін «комп’ютерні ігри» також може стосуватися ігор, у які грають переважно на персональних комп’ютерах або інших типах гнучких апаратних систем (також відомих як комп’ютерні ігри), щоб відрізняти їх від ігор, у які грають на стаціонарних пристроях.

Використовувалися в 1970-х і на початку 1980-х років, особливо для домашніх ігрових консолей, підключених до телевізорів [1].

В Японії ігрові консолі, такі як Odyssey, спочатку імпортували їх, а потім виготовляли в країні великі виробники телевізорів, такі як Toshiba і Sharp. Ці ігри

називалися «телевізійними іграми», або TV geemu або terebi geemu. «Електроніка» або «Гра» також може використовуватися для позначення відеоігор, але також включає ранні портативні електронні ігри, такі як пристрої без відеовиходу. Термін «відеогра» все ще широко використовується в 21 столітті [1].

Перший термін «відеогра» з'явився приблизно в 1973 році. Оксфордський словник англійської мови цитує статтю Business Week цього терміну від 10 листопаду 1973 року. Хоча Бушнелл вважає, що термін виник у 1971 році в огляді вендингового журналу Computer Space.

Назвати продукт «телевізійною грою» здавалося незручним, тому, щоб запозичити опис ігрового автомата з білбордом, Adlum дав назву розважальної машині «відео гра». Розважальні аркади були поширеними до початку 1970-х років Аркадні ігри без відео, наприклад, пінбол та електромеханічні ігри [1].

Однак ігри можуть бути розроблені для альтернативних платформ, ніж передбачалося, які описуються як порти або перетворення.

Також можуть бути переробки - більшість вихідного коду оригінальної гри використовується повторно, арт-активи, моделі та ігрові рівні оновлені для сучасних систем і переробки, на додаток до покращень активів [1].

Більшість комп'ютерних ігор — це комп'ютерні ігри, в яких гравець взаємодіє з персональним комп'ютером (ПК), підключеним до відеомонітора. ПК не є спеціалізованими ігровими платформами, тому під час запуску однієї гри на різному обладнанні можуть виникати відмінності.

Крім того, відкритість надає розробникам такі функції, як зниження вартості програмного забезпечення, підвищення гнучкості, збільшення інновацій, емуляції, створення модифікацій або адаптацій, відкритий хостинг онлайн-ігор (де одна людина грає у відеоігри з іншими) тощо.

Ігрові комп'ютери – це комп'ютери або ноутбуки, призначені для ігор, часто використовують високопродуктивні та дорогі компоненти. Окрім ігор на ПК, існують ігри, які запускаються на мейнфреймах та інших подібних системах, де користувачі можуть віддалено входити в систему, щоб використовувати

комп’ютер.

Консольні ігри відбуваються на домашній консолі, спеціалізованому електронному пристрої, підключеному до спільного телевізора або композитного відеомонітора. Домашні консолі розроблені для ігор, у яких використовується спеціальне апаратне середовище, яке надає розробникам конкретні цілі розробки обладнання та забезпечує доступні функції.

Портативні пристрої, як правило, не такі потужні, як ПК чи консольні пристрої.

В період 1990-х і 2000-х роках багато портативних ігор використовували картриджі, що дозволяло їм грати в багато різних ігор. У 2010-х роках портативні консолі зникли, оскільки ігри на мобільних пристроях стали більш домінуючими.

Аркадні відеоігри, як правило, відносяться до ігор, в які грають на більш професійних типах електронних пристроїв, зазвичай призначених лише для однієї гри і розміщених у спеціальному великому шафі для монет, який має ЕПТ-екрани, аудіопідсилювачі та колонки.

В аркадних іграх часто є яскраво намальовані зображення які пов’язані з темою гри.

Браузерні ігри використовують стандартизацію технологій, щоб забезпечити функціональність веб-браузера на кількох пристроях, забезпечуючи міжплатформне середовище. Ці ігри можна визначити за веб- сайтом, на якому вони з’являються, як-от гра Miniclip. Інші названі на честь програмних платформ, які використовуються для їх розробки, таких як Java та Flash ігри.

З появою смартфонів і планшетів, стандартизованих для iOS і Android, мобільні ігри стали важливою платформою.

Ігри починають використовувати унікальні функції мобільних пристроїв, які не обов’язково присутні на інших платформах, наприклад, акселерометри, інформація GPS та пристрої з камерою, щоб підтримувати ігри з доповненою реальністю.

Хмарні ігри потребують мінімум апаратних пристроїв, таких як базові комп’ютери, консолі, ноутбуки, мобільні телефони і навіть спеціальні апаратні пристрої. Ігри обчислюються та відтворюються на віддалених пристроях з використанням

кількох методів передбачення.

Зворотна сумісність за своєю природою схожа на емуляцію, оскільки старі ігри можна грати на нових платформах, але зазвичай безпосередньо через обладнання та мікропрограму на платформі. Наприклад, PlayStation 2 може грати в справжні ігри PlayStation, просто підключивши оригінальний ігровий носій до нової консолі, а Nintendo Wii може грати в ігри Nintendo GameCube таким же чином.

Цифрове розповсюдження через інтернет або інші засоби комунікації та хмарні ігри полегшують потребу в будь-якому фізичному носії. У деяких випадках носій зберігання є пам’яттю лише для читання гри, або це може бути носій інсталяції, який зберігає основні засоби в локальному сховищі платформи гравця для швидшого завантаження та пізніших оновлень.

Ігри можуть розширюватися за допомогою нового вмісту та програмного забезпечення за допомогою пакетів розширення, які зазвичай доступні як фізичні носії інформації або як вміст, який можна завантажити. Він є доступний через цифрове розповсюдження та безкоштовно або використовуються для монетизації після випуску гри в вільний доступ.

Деякі ігри дозволяють гравцям створювати створений користувачами вміст, яким можна поділитися з іншими. Інші ігри, здебільшого на ПК, можуть бути доповнені модифікаціями або модифікаціями, створеними користувачами, які змінюють або доповнюють гру. Зазвичай вони не є офінійними, розроблені гравцем у процесі зворотного проектування гри, але інші ігри надають офіційну підтримку для модифікацій гри [1].

Відеоігри можуть використовувати багато типів пристроїв введення, щоб перевести людські дії в ігри. Найпоширенішим є використання ігрових контролерів, таких як геймпади та джойстики, що використовуються в більшості консолей.

Загальні елементи керування на останніх контролерах включають кнопки на обличчі, тригери на плечах, аналогові джойстики та панелі напрямків. Консолі зазвичай включають стандартні контролери, які знаходяться з самою консоллю,

тоді як периферійні контролери доступні окремо від виробника консолі або сторонніх постачальників [1].

Набори керування вбудовуються в портативні консолі та аркадні шафи. Нові технологічні вдосконалення включають додаткові технології в контролері або ігровій платформі, такі як сенсорні екрани та датчики руху, що дає гравцям більше можливостей для взаємодії з грою. Спеціальні контролери доступні для певних типів ігор, включаючи гоночні колеса, легкі гармати та танцювальні майданчики. Цифрові камери та детектори руху можуть фіксувати рухи гравців як вхідні дані для ігор, ефективно усуваючи елементи керування в деяких випадках, а в інших системах, таких як віртуальна реальність, для покращення занурення в гру.

Як і більшість інших засобів масової інформації, відеоігри можна розділити на різноманітні жанри. Однак, на відміну від фільму чи телебачення, де використовуються візуальні або оповідні елементи, відеоігри часто поділяють на жанри на основі їх взаємодії з ігровим процесом, оскільки це основний спосіб взаємодії з відеоіграми.

Сюжет не впливає на ігровий процес, шутер все одно залишається стрілком, чи то у фантастичних світах, чи в космосі.

Виняток становить жанр гри жахів, який використовується для ігор, заснованих на елементах фантастики та жахів, надприродних і психологічних історій жахів [1].

Назви жанрів часто самоописують відповідно до типу гри, наприклад, екшн, рольова гра чи шутер, хоча деякі джерела жанру налаштовують впливові роботи жанру, наприклад roguelikes у Rogue, клони Grand Theft Auto від Grand Theft Auto III і «Королівська битва» з фільму «Королівська битва».

Назви можуть змінюватися з часом, коли гравці, розробники та ЗМІ придумують нові терміни, наприклад, на основі гри 1993 року шутер від першої особи спочатку називався «клоном судного дня».

Ігрові жанри мають ієрархію з жанрами вищого рівня, такими як шутер та екшн, охоплюючи основні стилі конкретно реалізованих піджанрів та шутери від

третьої особи. Деякі міжжанрові жанри також існували, поки не з’явилися кілька вищих жанрів, наприклад, пригоди.

Ігри кількох гравців які засновані для змагання один з одним пропонують кооперативні та асиметричні ігрові процеси. Структура сервера, що використовується в онлайн-іграх, може дозволити багатокористувацькі онлайн-ігри (MMO) які підтримують одразу велика кількість гравців.

Деякі відеоігри є іграми з нульовими гравцями з дуже обмеженою взаємодією гравця з самою грою. Здебільшого це ігри-симулятори, де гравець скидає налаштування, а потім дозволяє грі грати самостійно, спостерігаючи за результатами як звичайний спостерігач, як і багато комп’ютерних симуляцій Conway's Life Game [1].

Більшість відеоігор створені для розважальних цілей, ця категорія також відома як «базові ігри». Деякі ігри створені не для розваг. Такі як казуальні ігри, навчальні, серйозні та художні.

Для легкого доступу та швидкого розуміння набору правил розроблені казуальні ігри. Вони часто підтримують можливість входити і виходити з гри на вимогу, наприклад, під час робочих або обідніх перерв.

Багато браузерних і мобільних ігор належать до категорії казуальних ігор, а казуальні ігри часто походять із типу ігрових елементів низької інтенсивності, таких як приховані об’єкти, управління часом та ігри- головоломки.

Причинно-наслідкові ігри часто використовують механіку соціальних мереж, де гравці можуть взаємодіяти з друзями та робити додаткові повороти або ходи щодня. Популярні казуальні ігри включають Tetris і Candy Crush Saga. Зовсім недавно, з кінця 2010-х років, з’явилися гіпер-казуальні ігри, які використовують простіші правила для створення коротких, але нескінченно повторюваних ігор, таких як Flappy Bird [1].

Освітнє програмне забезпечення використовується вдома та в класах для навчання дітей та студентів, а навчальні ігри адаптовані для цих причин, щоб забезпечити взаємодію та розваги, пов’язані з елементами ігрового дизайну. Існують різні відмінності в тому, як вони розроблені та навчають користувачів.

Прикладами навчальних ігор є The Oregon Trail та серія Carmen Sandiego.

Навчальні ігри є формою серйозних ігор, але інші типи серйозних ігор включають фітнес-ігри, які включають важливі фізичні вправи, щоб допомогти гравцям залишатися у формі, наприклад, (Wii Fit), симулятори польотів (Microsoft), які імітують пілотування комерційних і військових літаків.

Також існують серйозні ігри – це ігри, які додають, приховують або навіть видаляють елемент розваги через інші цілі гри. Ігри створені для нерозважальних цілей, щоб покращити гру, наприклад, використання технології відеоігор для інтерактивних ігрових світів або гейміфікація для покращення.

Симулятори, рекламні ігри на основі реклами продуктів (наприклад, Pepsi) та ігри новин, призначені для поширення певного рекламного повідомлення (наприклад, NarcoGuerra).

Хоча відеоігри вважаються окремою формою мистецтва, художні ігри також можуть бути розроблені так, щоб намагатися навмисно передати історію чи повідомлення.

Художні або схожі на мистецтво ігри створені для того, щоб викликати у гравців емоції та співпереживання, кидати виклик соціальним нормам та критикувати через інтерактивність середовища відеоігор.

Вони можуть не мати умов виграшу і призначені для того, щоб дозволити гравцям досліджувати ігрові світи та сценарії. Більшість художніх ігор — це інді-ігри, засновані на особистому досвіді або історіях, створених одним розробником або невеликою командою. Приклади художніх ігор включають Passage, Flower, That Dragon, Cancer [1].

1.2 Поняття платформеру

Гра-платформер (часто скорочено — платформер або гра зі стрибком і бігом) — це жанр відеоігор та піджанр екшн-ігор, головна мета яких — переміщення персонажа гравця між точками у відтвореному середовищі. Платформні ігри характеризуються дизайном рівнів з нерівним рельєфом і

підвісними платформами різної висоти, які вимагають використання здібностей персонажів гравців, таких як стрибки та лазіння, для навігації в середовищі гравця та досягнення цілей. Інші акробатичні рухи також впливають на ігровий процес, наприклад, розмахування з таких об'єктів, як ліани або гаки, стрибки зі стін, стрибки в повітрі, ковзання в повітрі, стрільба з гармат або стрибки з трампліна чи батута. Ігри з повністю автоматизованими стрибками, наприклад 3D-ігри з серії Legend of Zelda, перевершують цей жанр.

Хоча вони часто асоціюються з консольними іграми, багато відомих платформерів були випущені для відеоігор, а також портативних консолей і домашніх комп'ютерів.

Під час піку популярності платформних ігор наприкінці 1980-х і на початку 1990-х років платформери становили від чверті до третини всіх консольних ігор, але з тих пір були витіснені шутерами від першої особи. У 2006 році популярність жанру впала, становивши 2% ринку в порівнянні з 15% у 1998 році, однак, жанр все ще існує в комерційному середовищі, і кількість проданих ігор обчислюється мільйонами.

Концепцією є гра на платформах що вимагає від гравців маневрувати своїми персонажами між платформами, щоб досягати цілей, битися з ворогами та уникати перешкод на шляху. Ці ігри представлені або збоку з використанням 2D руху, або в 3D з камерою позаду головного героя, або з ізометричної перспективи. Звичайний ігровий процес платформних ігор часто дуже динамічний і кидає виклик рефлексам гравця, часу та спритності за допомогою елементів керування.

Найпоширенішими варіантами вправ у цьому жанрі є ходьба, біг, стрибки, атака та лазіння. Стрибки є основою жанру, але є винятки, наприклад, Popeye від Nintendo (1982). У деяких іграх траєкторія стрибка фіксована, але в інших вона може змінюватися в повітрі. Падіння зі значної висоти зазвичай призводить до травм або смерті. Багато платформерів мають екологічні бар'єри, які вбивають персонажа гравця при контакті, наприклад, лавові ями або прірва. У різних областях ігрового світу гравці можуть збирати предмети та нагороди, які стануть в нагоді в різних ситуаціях, а також дати головному герою нові здібності для

подолання труднощів.

Більшість ігор цього типу складаються з кількох рівнів і з проходженням кожного складність ігрового процесу зростає, це може перемежовуватися зі зустрічами з босами, коли персонаж повинен перемогти особливо небезпечного ворога, щоб підняти собі рівень. Зазвичай порядок рівнів заздалегідь визначений, але деякі ігри також дозволяють гравцеві вільно переміщатися в ігровому світі або можуть мати різні шляхи в певний час. Ще одним поширеним елементом жанру є прості логічні головоломки, які потрібно розв'язувати, і тести на навички.

Більш сучасний варіант платформера (зазвичай 3D-прокрутки), який називається «раннер», в якому головний герой завжди рухається на великій швидкості, і гравець повинен правильно керувати ним, щоб він не впав і не вдарився об перешкоди, і керувати ним для досягнення контрольних точок своєчасно. Цей тип ігор найкраще підходить для мобільних пристроїв через відносно просте керування і останнім часом став дуже популярним.

У роки після виходу першого визнаного імені в жанрі Donkey Kong (1981) використовувалися різні назви. Під час розробки Сігеру Міямото вперше назвав Donkey Kong «грою - бігти/стрибати/лазити». Міямото часто використовує термін «спортивна гра» для позначення Donkey Kong та пізніших ігор у цьому жанрі, таких як Super Mario Bros (1985) [2].

Donkey Kong породила безліч інших ігор, які поєднували біг, стрибки та вертикальні рухи, новий жанр, який не відповідав стилю попередніх ігор, дозволяючи журналістам та письменникам надавати власні умови. Комп'ютерні та відеоігри, серед іншого, називають жанр «Donkey Kong» або
«Donkey Kong style». «Climbing Games» використовувався в книзі Стіва Блума 1982 року «Video Invaders» і в Electronic Games (США) і TV Gamer (Великобританія) 1983 року. Блум визначає «гра в лазіння» як гру, в якій гравець повинен піднятися вгору з нижньої частини екрана, уникаючи та/або знищуючи перешкоди та ворогів, які часто зустрічаються на шляху. Виходячи з цього визначення, він назвав Space Panic (1980), Donkey Kong і Frogger (1981) іграми зі

скелелазінням [2].

У грудні 1982 року в огляді Creative Computing Beer Run Apple II рецензент використав інший термін: «Я називаю це драбиною, як у «типах сходів», які включають Apple Panic і Donkey Kong. Програвач відеоігор також використовував термін «драбина гра» у 1983 році, коли він присудив Donkey Kong's Coleco Port Ladder of the Year нагороду.

Іншим терміном, який зазвичай використовується для опису жанру, є «рольові ігри» з кінця 1980-х до 1990-х років, який використовувався для позначення ігор, таких як Super Mario Bros, Sonic the Hedgehog та Bubsy. Однак цей термін також ширше застосовується до відеоігор з боковою прокруткою, включаючи шутери, такі як Gunstar Heroes.

«Платформер» став стандартним терміном наприкінці 1980-х років, популярним у британській пресі. До 1989 року британські журнали використовували термін «платформер» для позначення жанру; приклади включають посилання на «форми Супер Маріо» (такі як Като-чан і Кен-чан) як платформер або Strider як для гри «Платформи та драбини» [2].

1.3 Розробка відеоігор

Розробка відеоігор – це процес або робота яку виконують розробники, починаючи від окремих осіб і закінчуючи міжнародними командами по всьому світу. Традиційні розробки комерційних комп’ютерних і консольних ігор фінансується видавцями та створюються роками. Інді-ігри, як правило, вимагають менше часу та грошей, і можуть бути створені окремими особами та невеликими розробниками. Інді-індустрія ігор процвітає завдяки доступному програмному забезпеченню для розробки ігор, такому як Unity і Unreal Engine, новим системам онлайн-розповсюдження, таким як Steam і Uplay, а також ринках Android і iOS [2].

Початок розробки відеоігр приходиться на 1960-х роки, яки на той час не комерціалізувалися. Для роботи їм потрібні були мейнфрейми яки були недоступні

для громадськості. Розробка комерційних ігор почалася в 1970-х роках з появою ігрових консолей та домашніх комп'ютерів. Через низьку вартість і низьку функціональність домашніх комп'ютерів програміст розробляв одну повну ігру.

Однак наприкінці 1980-х і 1990-х роках зростання обчислювальної потужності комп'ютерів і зростання очікувань геймерів ускладнювали роботу однієї людини. створити звичайну консольну або комп'ютерну гру. Середня вартість створення відеоігор у трьох жанрах повільно зросла з 1 мільйона доларів до 4 мільйонів доларів у 2000 році до понад 5 мільйонів доларів 2006 рік, у 2010 році до 20 мільйонів доларів.

Великі комерційні комп'ютерні та консольні ігри розробляються поетапно: на етапі попереднього виробництва пишуться презентації, прототипи, ігровий дизайн-документ після погодження схваленої ідеї починається фінансування та повномасштабна розробка. Проект повноцінної гри включає в себе команду від 20 до 100 людей. Ігри розробляються творчо та з метою отримання прибутку.

Ігрова індустрія потребує інновацій, оскільки видавці не можуть отримати прибуток від постійного випуску повторюваних сиквелів та імітацій. Щороку відкриваються нові інді-розробники, яки створюють популярні ігри, в той же час багато розробників закриваються, тому що їхнє виробництво нерентабельне [2].

На початку 1980-х років, коли з'явилися домашні комп'ютери та ігрові консолі, програміст міг впоратися з будь-яким завданням розробки ігор — програмуванням, графічним дизайном, звуковими ефектами тощо. Взагалі розробка гри може зайняти всього шість тижнів. Однак користувачі висувають високі очікування та вимоги до сучасних комерційних ігор, які виходять за межі можливостей окремого розробника, що вимагає поділу праці. У проекті може працювати команда з понад 100 людей на повний робочий день [2].

Розробка, виробництво або дизайн гри – це процес, який починається з ідеї чи концепції. Часто ідея заснована на модифікації існуючої концепції гри. Ідея для гри може належати до одного або кількох жанрів. Дизайнери часто експериментують з різними жанровими комбінаціями. Як правило, ігрові дизайнери готують початковий документ пропозиції гри, який описує основні

концепції, ігровий процес, список функцій, налаштування та історію, цільову аудиторію, вимоги та терміни, персонал і бюджет. Різні компанії мають різні формальні процедури та філософії для проектування та розробки ігор. Стандартизованого методу розробки не існує, проте є деякі загальні особливості [2].

Розробники ігор можуть варіюватися від окремих осіб до великих транснаціональних корпорацій. Є незалежні студії та видавничі студії яким доводиться розробляти ігри. Офіційна пропозиція на гру подається видавцю, який фінансує гру до декількох років та зберігає за собою виключні права на публікацію та рекламування. Зазвичай видавцем є особа, яка володіє інтелектуальною власністю гри [2].

Розробка більшості сучасних комп'ютерних або консольних ігор займають декілька років вплив мають фактори: тип, активи, платформи, масштаби.

Для завершення деяких ігор може знадобитися більше десяти років. Прикладом є Duke Nukem Forever з 3D Realms, виробництво якого було оголошено в квітні 1997 року і випущено через 14 років у червні 2011 року. Планування Maxis Spore почалося наприкінці 1999 року, гра була випущена через дев'ять років у вересні 2008 року. Prey був коротко описаний у PC Gamer 1997 року, але був випущений лише в 2006 році, і лише потім у сильно зміненому вигляді. Нарешті, за словами Гейба Ньюелла, розробка Team Fortress 2 з 1998 року до її випуску в 2007 році була результатом заплутаного процесу розробки, який включав «ймовірно, три-чотири різні ігри» [2].

Дохід від роздрібної торгівлі ігор розподіляється між різними сторонами в ланцюжку розповсюдження, такими як розробники, видавці, роздрібні продавці, виробники та роялті з консолей. Багато розробників не змогли отримати від цього прибуток і збанкрутували. Багато розробників шукають альтернативні економічні моделі для збільшення віддачі через Інтернет-маркетинг і канали розповсюдження. Оскільки мобільне розповсюдження може становити 70% загального доходу [2].

Оскільки відеоігри — це програмне забезпечення з мистецтвом, аудіо та

ігровим процесом, формальні методи розробки програмного забезпечення часто ігноруються. Погано розроблена гра може перевищувати бюджет і час, а також містити багато помилок. Планування важливе для індивідуальних та групових проектів [2].

Загальна розробка ігор не вписується в типовий підхід життєвого циклу програмного забезпечення, такий як модель водоспаду.

Одним із методів розробки гри є agile development. Він заснований на ітеративному прототипуванні, підмножині програмного прототипування. Agile розробка покладається на зворотний зв'язок і вдосконалення ітерацій гри, оскільки набір функцій поступово збільшується.

Цей підхід працює, оскільки більшість проектів не мають чітких вимог. Популярним методом розробки гнучкого програмного забезпечення є Scrum.

Іншим успішним підходом є процес персонального програмного забезпечення (PSP), який вимагає додаткового навчання співробітників для підвищення обізнаності щодо планування проекту. Цей підхід є більш дорогим і вимагає від членів команди відданості. PSP можна поширити на процес командного програмного забезпечення, яким керує вся команда [2].

Розробка ігор часто передбачає застосування цих методів. Наприклад, створення активів можна здійснити за допомогою каскадної моделі, оскільки вимоги та специфікації чітко визначені, але дизайн ігрового процесу можна зробити за допомогою ітераційного прототипування [2].

З удосконаленням комп'ютерних технологій і розвитком індустрії відеоігор гравці хочуть переживати все більш реалістичні ігрові пригоди найвищої якості та якомога автентичніші . Від початку індустрії відеоігор до середини 1990-х років двовимірні ігри були нормою.

У той час навіть не можна було використовувати 3D-технології. Коли 2D-ігри вперше з'явилися, вони здавалися неймовірними, але 2D-ефекти перетворилися на 3D-ефекти ігор , набувши набагато більшого зростання завдяки цій індустрії, що розвивається. Нині 3D-технологія настільки складна, що сцени в деяких відеоіграх такі природні, майже як у фільмах. Це не означає, що 2D-

ігри залишилися в тіні. Кожен тип гри має свою аудиторію гравців. І 2D, і 3D мають плюси і мінуси.

Індустрія відеоігор зросла настільки, що розробники відеоігор щодня запускають ігри на різних платформах, і вони більше не обмежуються 2D і 3D іграми. Гравці люблять проводити час онлайн, граючи в інтерактивні відеоігри, як 2D, так і 3D. Згідно зі звітом «State of Online Gaming 2021», розробленим на замовлення Limelight Networks, Inc., геймери в усьому світі витрачають у середньому 8 годин 27 хвилин щотижня на відеоігри , тоді як індійські геймери витрачають більше 8 годин 36 хвилин на тиждень.

За десятиліття, що щойно закінчилося, відеоігри перетворилися на мільярдний бізнес, а прибутки перевищили прибутки кіно- та музичної індустрії.

Безліч незалежних ігрових студій розробили ігри для всіх типів платформ: комп'ютерів, консолей, планшетів і мобільних телефонів. У результаті ще більше людей регулярно грають у відеоігри. І минули ті часи, коли в ці ігри грали лише діти та підлітки. Літні люди також відкривають для себе головоломки та ігри на спритність на своїх мобільних телефонах. У поїзді, в очікуванні автобуса або перед сном: у будь-який вільний момент доступна відеогра. Хоча компульсивні ігри не є чимось новим, кількість залежних від ігор зростає з поширенням смартфонів. Однак для більшості тих, хто грає, гра є нешкідливою розвагою. Такі ігри як Red Dead Redemption 2 (2019), який завдяки витонченим діалогам і емоційним сюжетам не тільки забезпечує години ігрового задоволення, але й приймає інший курс відповідно до рішень гравця, і тому можна грати знову і знову, випущено на ПК , PlayStation і Xbox. Іншим явищем цього десятиліття є те, що відоме як «Let's play» відео. Геймери записують себе під час гри, коментуючи процес гри та даючи поради, а потім публікують відео на YouTube, де воно збирає мільйони кліків.

Бажання грати таке ж давнє, як і саме людство. Для дітей це життєво важливий інструмент, щоб підготувати їх до життя, а для дорослих це спосіб втекти від реальності. Тріумфальне зростання відеоігор почалося в 1970-х роках, але корінням воно сягає подій, що відбулися приблизно 20 років тому.

2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Огляд ігрового рушія

Unity — це платформа для розробки відеоігор. Ми досліджуватимемо Unity як платформу для розробки ігор і досвіду. Ігрові движки, також звані ігровими фреймворками, — це спеціальне програмне забезпечення, яке розробники ігор використовують для створення як 2D, так і 3D відеоігор. Як ігровий движок, Unity є чудовою платформою для проектування уявних світів, розрахунку фізики для об’єктів у цих світах, щоб рухатися, стрибати чи стикатися, відтворювати звуки тощо. Розробники використовують ігрові движки Unity, через те, що вони надають багаторазовий код, який знадобиться кожній грі. Це означає, що витрачається менше часу на відновлення колеса та більше часу на створення цікавого нового вмісту.

Як і більшість ігрових движків, Unity має п’ять компонентів. Основним компонентом є програма, яка містить логіку гри, механізм візуалізації, який генерує тривимірну анімовану графіку. Компонент аудіосистеми надає алгоритми для створення звуків і керування ними. Відповідно до фізичних законів фізичний механізм обробляє рух і зіткнення. Для дій і рішень, які не виконуються гравцями, є компонент штучного інтелекту.

Моделі для візуалізації графіки та освітлення світу легко доступні в ігрових движках. Створити персонажів із фізичною поведінкою можна швидко. Ці інструменти звільняють розробників і дизайнерів від зосередження на логіці та процесі гри.

Цей об’єктно-орієнтований і модульний підхід дає можливість розробникам робити внесок у продукт ігрового движка та ділитися своїм «проміжним» програмним забезпеченням. Це програмне забезпечення покращує та додає функції до ігрового двигуна. Компоненти проміжного ПЗ вирішують конкретні завдання. Unity використовувала концепцію об’єктно-орієнтованого програмування для створення ігрового движка з чітко визначеним і розширюваним набором об’єктів.

У 2013 році Facebook інтегрував набір для розробки ігрового програмного забезпечення за допомогою ігрового движка Unity. Це включає інструменти для відстеження рекламних кампаній і глибоких посилань, за допомогою яких користувачі отримують прямі посилання з публікацій у соціальних мережах на певні частини світу. Легкий обмін зображеннями в іграх і поза ними. У 2016 році Facebook використав Unity для розробки нової ігрової платформи для ПК. Unity підтримує ігрову платформу Facebook, дозволяючи розробникам Unity швидше експортувати та публікувати ігри на Facebook.

Ігрові движки, також звані ігровими фреймворками, — це спеціальне програмне забезпечення, яке розробники ігор використовують для створення як 2D, так і 3D відеоігор. Як ігровий движок, Unity є чудовою платформою для проектування уявних світів, розрахунку фізики для об’єктів у цих світах, щоб рухатися, стрибати чи стикатися, відтворювати звуки тощо. Розробники також люблять такі ігрові движки, як Unity, через те, що вони надають багаторазовий код, який знадобиться кожній грі. Це означає, що витрачається менше часу на відновлення колеса та більше часу на створення цікавого нового вмісту. Цей об’єктно-орієнтований і модульний підхід дає можливість розробникам робити внесок у продукт ігрового движка та ділитися своїм «проміжним» програмним забезпеченням.

Це програмне забезпечення покращує та додає функції до ігрового двигуна. Компоненти проміжного ПЗ вирішують конкретні завдання. Розглянемо звуковий механізм Wwise від Audiokinetic. Wwise — це багатофункціональне інтерактивне аудіорішення, яке використовується для централізованого створення аудіо. Фактично, ігровий процес може використовувати Wwise для взаємодії та зміни звуку в грі! Або Physx, наданий Nvidia . Ця фізична система з відкритим кодом покращує час обчислень завдяки підтримці багатопоточності.

Unity використовувала концепцію об’єктно-орієнтованого програмування для створення ігрового движка з чітко визначеним і розширюваним набором об’єктів. Деякі з цих об’єктів спрямовані на конкретну галузь, а інші є більш загальними. Користувачі можуть змішувати та поєднувати об’єкти з різними візуальними та

фізичними атрибутами, наданими Unity або створеними розробником.

2.2 Огляд мови програмування

C# є мовою програмування загального призначення, яка дозволяє розробникам створювати програми для різних цілей, включаючи веб-розробку, розробку додатків для настільних комп'ютерів та мобільних пристроїв, розробку ігор та різноманітних програмних продуктів.

Синтаксис C# схожий на синтаксис інших мов програмування з родини мов C, таких як C++ та Java, тому для програмістів, які вже володіють досвідом роботи з цими мовами, вивчення C# не буде складним. Однак, C# має свої особливості та можливості, які роблять його унікальним.

Основні особливості C#:

Об'єктно-орієнтований підхід: C# дозволяє програмістам створювати класи, які описують об'єкти, що використовуються в програмі. Це дає можливість створювати більш складні програми з більшою кількістю функцій та операцій.

Система керування пам'яттю: C# використовує систему автоматичного керування пам'яттю, що дозволяє програмістам не звертати увагу на процеси виділення та звільнення пам'яті.

Бібліотека класів .NET: C# має доступ до широкого спектру класів, що забезпечують виконання різноманітних завдань, таких як робота з мережевим протоколами, робота з файлами та базами даних, взаємодія з операційною системою.

Ці класи збираються в бібліотеку класів .NET, яка є частиною платформи .NET Framework.

Розширені можливості безпеки: C# має розширені можливості безпеки, що дозволяють програмістам розробляти програми, які захищені від вразливостей та атак.

Підтримка мультиплатформеності: C# може бути виконана на різних є основною мовою для розробки програм для платформи .NET Core, що дозволяє

створювати програми, які можуть бути виконані на різних пристроях та операційних системах.

Підтримка паралельності: C# має вбудовану підтримку паралельного програмування, що дозволяє програмістам розробляти програми, які можуть виконуватись паралельно на багатьох процесорах або ядрах.

Строга типізація: C# має строгу типізацію, що дозволяє програмістам визначати типи даних, що використовуються у програмі. Це дозволяє запобігти багам, що пов'язані з неправильним використанням даних.

У загальному, C# є мовою програмування з великою кількістю можливостей та переваг. Це дозволяє програмістам розробляти програми для різних платформ та завдань, забезпечуючи високу продуктивність та безпеку.

C# — строго типізована об’єктно-орієнтована мова програмування. C# є відкритим кодом, простим, сучасним, гнучким і універсальним.

C# — це універсальна мова програмування з багатьма парадигмами, яка включає в себе строго типізовані дисципліни програмування з лексичною областю, імперативні, декларативні, функціональні, загальні об’єктно-орієнтовані (на основі класів) і компонентно-орієнтовані дисципліни.

Методи під час виконання прив’язують динамічні типи, що дозволяють викликати методи, схожі на JavaScript, і створювати об’єкти.

Як і псевдосигнали та слоти фреймворку Qt, C# має семантику навколо подій стилю публікації-підписки, хоча C# використовує для цього делегатів.

C# надає Java-подібні синхронізовані виклики методів через атрибут (MethodImpl(MethodImplOptions.Synchronized) і підтримує блокування мьютексів із блокуваннями за ключовими словами.

Найбільшою перевагою є те, скільки часу ви можете заощадити, використовуючи C# замість іншої мови програмування.

Оскільки C# має статичний тип і його легко читати, користувачі можуть витрачати менше часу на пошук у сценаріях дрібних помилок, які порушують роботу програми. Згідно з опитуванням Stack Overflow 2022 року, програмісти оцінили C# як одну з найпривабливіших мов програмування, доступних сьогодні,

одразу після Python.

2.3 Огляд середовища розробки

Visual Studio являє собою інтегроване середовище розробки (IDE).

Visual Studio 2022 - це комплексне програмне середовище, розроблене компанією Microsoft, яке дозволяє програмістам ефективно розробляти програми для платформи Windows. Це високоефективний інструмент, який надає широкі можливості для створення високоякісного програмного забезпечення.

Visual Studio 2022 має декілька важливих функцій та особливостей, які дозволяють зручно та швидко розробляти програмне забезпечення. Деякі з них:

- Керування проектами: Visual Studio 2022 надає можливість керувати проектами та їх залежностями. Це дозволяє зручно керувати структурою проекту та його компонентами.
- Розширені можливості відлагодження: Visual Studio 2022 має розширені можливості відлагодження, що дозволяють програмістам зручно відлагоджувати код та знаходити помилки.
- Вбудована документація: Visual Studio 2022 має вбудовану документацію, що дозволяє програмістам швидко знайти необхідну інформацію про функції, класи та методи.
- Розширені можливості роботи з Git: Visual Studio 2022 має розширені можливості роботи з Git, що дозволяють зручно керувати версіями коду та співпрацювати з іншими членами команди.
- Підтримка віддаленої розробки: Visual Studio 2022 має підтримку віддаленої розробки, що дозволяє програмістам розробляти програмне забезпечення на віддалених серверах та віртуальних машинах.
- Розширені можливості тестування: Visual Studio 2022 має розширені можливості тестування, що дозволяють програмістам зручно тестувати свій код та виявляти помилки.
- Вбудований редактор графічного інтерфейсу: Visual Studio 2022 має

вбудований редактор графічного інтерфейсу, що дозволяє програмістам швидко та зручно створювати графічний інтерфейс для свого програмного забезпечення.

– Розширені можливості підключення додаткових інструментів: Visual Studio 2022 має розширені можливості підключення додаткових інструментів та плагінів, що дозволяє програмістам зручно розширювати можливості IDE та підлаштовувати його до своїх потреб.

Visual Studio розроблене Microsoft для GUI (графічного інтерфейсу користувача), консолі, веб-програм, веб-програм, мобільних програм, хмарних і веб-служб.

За допомогою IDE можемо створювати керований код, а також рідний код. Він використовує різні платформи програмного забезпечення для Microsoft, Microsoft Silverlight і Windows API.

Еволюція Visual Studio: перша версія VS випущена в 1997 році під назвою Visual Studio 97 з номером версії 5.0. Її також називають Visual Studio 2017. Підтримувані версії .Net Framework в останній версії Visual Studio — від 3.5 до 4.7. Переваги використання Visual Studio IDE:

1.Доступна повнофункціональна платформа програмування для кількох операційних систем, Інтернету та хмари, Visual Studio IDE. Користувачі можуть легко переглядати інтерфейс користувача, щоб вони могли швидко й точно написати свій код.

2.Щоб допомогти розробникам швидко визначити потенційні помилки в коді, Visual Studio пропонує надійний інструмент налагодження.

3.Розробники можуть з упевненістю розмістити свою програму на сервері, оскільки вони усунули все, що могло призвести до проблем із продуктивністю.

4.Незалежно від того, яку мову програмування використовують розробники, користувачі Visual Studio можуть отримати живу підтримку програмування. Для швидшої розробки Платформа пропонує опцію автозаповнення.

5.Вбудована інтелектуальна система пропонує описи та поради для API. За допомогою Visual Studio IDE ви можете легко співпрацювати з колегами в

одному проекті. Ця IDE допомагає розробникам ділитися, надсилати та витягувати свій код зі своїми товаришами по команді.

6.Кожен користувач Visual Studio має можливість налаштувати його. Вони мають можливість додавати функції відповідно до своїх потреб.

Наприклад, вони можуть завантажувати доповнення та встановлювати розширення у своїй IDE. Навіть програмісти можуть надсилати власні розширення.

.NET Standard - це специфікація, що визначає набір API (Application Programming Interface) для платформі .NET. Ця специфікація встановлює мінімальний набір API, які повинні бути підтримані всіма імплементаціями .NET, що підписались на цю специфікацію.

Основною метою створення .NET Standard є забезпечення сумісності між різними імплементаціями .NET. Завдяки цьому, розробники можуть створювати бібліотеки класів, які можуть використовуватися на різних імплементаціях .NET без потреби їх переписування.

.NET Standard встановлює загальну базу класів, яка містить типи, що повинні бути підтримані всіма імплементаціями .NET. Ці типи включають базові класи та інтерфейси, такі як System.Object, System.String, System.Collections.Generic та інші.

Крім того, .NET Standard визначає спеціальні набори API, які повинні бути підтримані для різних версій імплементацій .NET. Наприклад, .NET Standard 2.0 містить API, які підтримуються в .NET Framework 4.6.1, .NET Core 2.0, Xamarin.iOS 10.14, Xamarin.Android 7.5 та інших.

.NET Standard використовується для розробки бібліотек класів, які можуть використовуватися на різних платформах .NET, таких як .NET Framework, .NET Core та Xamarin. Розробники можуть створювати бібліотеки класів з використанням .NET Standard, які будуть працювати на всіх платформах, що підтримують цю специфікацію.

Однією з переваг використання .NET Standard є забезпечення сумісності між різними версіями імплементацій .NET. Розробники можуть створювати

бібліотеки класів з використанням .NET Standard, які підтримують різні версії .NET Framework, .NET Core, Xamarin і т.д. Це дозволяє зменшити залежність від конкретної версії платформи .NET і забезпечує більш широку сумісність бібліотек.

Крім того, .NET Standard дозволяє розробникам створювати бібліотеки класів, які будуть працювати на різних платформах .NET, що дозволяє створювати більш уніфіковану кодову базу. Це дозволяє збільшити швидкість розробки і підтримки коду, що покращує якість розробки програмного забезпечення.

Також, .NET Standard дозволяє розробникам використовувати більш широкий набір функцій, що реалізовані на різних імплементаціях .NET, в тому числі .NET Framework, .NET Core, Xamarin і т.д. Це дозволяє розробникам використовувати бібліотеки класів з використанням більш широкого набору функцій та API, що забезпечує більш гнучкий підхід до розробки.

Більш того, .NET Standard підтримує автоматичне завантаження залежностей, що дозволяє забезпечити більш просту та зручну розробку програмного забезпечення. Крім того, ця специфікація підтримує пакування та розповсюдження бібліотек класів, що забезпечує більш простий та зручний процес розгортання.

3 ПРОЕКТУВАННЯ ГРИ

3.1 Створення сценарію гри

Створення сценарію гри - це важливий процес, який може забезпечити успішність. Перед початком розробки сценарію потрібно визначити жанр гри. Чи буде це аркадна гра, головоломка або інша казуальна гра? Це допоможе сконцентруватися на тематиці, механіці та інших аспектах гри.

Спочатку потрібно визначити мету гри. Визначивши мету гри, з’являється змога більш чітко визначити сценарій. Сюжет може бути простим або складним, але він повинен бути захопливим для гравців. Використовуйте діалоги, щоб розкрити характери персонажів та розкрити сюжет.

Механіка гри повинна відповідати жанру та меті гри. Визначення ігрових механік на початковому етапі значно спрощують подальшу розробку. Рівні повинні бути прогресивними, цікавими та захоплюючими означаючи зростання складності гри та відповідая меті.

Після створення сценарію гри потрібно протестувати гру та переконатись що вона захоплююча та викликає цікавість у гравців. Виправлення помилок та додавання нових елементів покращить гру.

3.2 Взаємодія з користувачем

Взаємодія гри з користувачем відбувається за допомогою сенсорного екрану та звуку. Гравець завдяки сенсорному екрану включає різні методи інтерактивності: дотик, свайп, тапання, мультитач жестів, та інші. Методи використовуються для контролю над персонажами гри, руху камери, стрільби та інших дій в грі.

Сенсорний екран є важливою складовою мобільних ігор, оскільки дозволяє гравцеві взаємодіяти за допомогою простих жестів та дотиків. Він забезпечує більш прямий та інтуїтивний спосіб керування грою, що зазвичай використовуються для

ігор на комп'ютерах.

Залежно від жанру гри, сенсорний екран може мати різні функції. У іграх з жанру "Match-3" гравець може переміщати елементи на екрані шляхом перетягування їх пальцем, а у іграх з жанру "Runner" гравець може уникати перешкод та стрибати.

Для мобільних ігор складовою яка керує, відкриває та створює нові та цікаві ігрові елементи є сенсорний екран. Він дозволяє гравцям: більш прямо та інтуїтивно керувати грою, відкривати багато можливостей для творців ігор у створенні нових та цікавих ігрових елементів, надаючи гравцям більше контролю над діями в грі, взаємодіяти з різними ігровими об'єктами, такими як інтерфейсні елементи, текстові поля та зробити ігровий процес більш інтуїтивним та зручним для гравців.

Взаємодія з гравцем через сенсорний екран підвищує ігровий досвід, сприяючи більшій інтерактивності та залученню гравців до гри.

У деяких іграх дотик використовуватися для створення враження присутності, дозволяючи гравцям взаємодіяти з об'єктами в грі, використовуючи жести, які дуже нагадують дії в реальному житті. Наприклад, в грі-головоломці гравець може перетворювати кубики, розташовані на екрані, за допомогою жестів, що надає враження прямої взаємодії з об'єктами.

Використання звуку відбувається коли гравець не може або не повинен дивитися на екран та для повідомлення гравця про події в грі, таких як зміна рівня, отримання повідомлень або попередження про небезпеку. Звукові ефекти вказують де знаходяться предмети, які гравець повинен зібрати або де знаходиться вихід з рівня.

Музикальні ефекти використовуються для створення настрою гри та забезпечення більш іммерсивного досвіду для гравця. Використання музики створює певний настрій, а звукові ефекти допомогають більш наочно уявити ігровий світ. Деякі ігри можуть використовувати голосові команди або голосове керування для взаємодії з гравцем. Створення незабутнього досвіду для гравців та покращення геймплею досягаються використанням звуку у грі.

3.3 Розробка архітектури гри

Початковий етап розробки архітектури гри, потрібно розпочинати з визначення головних компонентів гри: геймплей, графіка, звук, фізика, штучний інтелект, мережевий геймплей. Далі визначають, як компоненти взаємодіють між собою та які є залежності між ними.

Один із підходів до розробки архітектури гри - це розбиття гри на модулі або компоненти, які можна розробляти окремо та об'єднувати в одну гру. Цей підхід дозволяє розробникам працювати над різними частинами гри одночасно та зменшує ризик конфліктів між різними частинами гри.

Наступний підхід використання шаблону проектування для розробки архітектури гри. Шаблони проектування - це загальні рішення для типових проблем, які можуть виникнути під час розробки програмного забезпечення. Використання шаблонів дозволяє розробникам швидко та ефективно вирішувати типові проблеми та зменшувати кількість помилок під час розробки гри. Вони розробляються та застосовуються для вирішення повторюваних проблем під час проектування програмного забезпечення.

Шаблони проектування застосовані в різних областях та мовах програмування, включаючи розробку ігор. Шаблон проектування у програмуванні називають патерном.

Найпоширеніші шаблони проектування включають:

- Observer - дозволяє одним об'єктам слідкувати та реагувати на зміни в інших об'єктах.
- Decorator - дозволяє динамічно додавати нову функціональність до існуючих об'єктів.
- Adapter - це шаблон, який дозволяє використовувати існуючий клас з іншим інтерфейсом. При проектуванні та реалізації програмного забезпечення зменшення помилок виникає за допомогою патернів проектування та дозволяє зробити код більш зрозумілим та ефективним.

При проектуванні та реалізації програмного забезпечення зменшення помилок виникає за допомогою патернів проектування та дозволяє зробити код більш зрозумілим та ефективним.

Наведемо категорії шаблонного проектування:

- Шаблон створення об'єктів (Creational Patterns) - шаблони що допомагають вирішувати проблеми, пов'язані з створенням об'єктів. Наприклад, шаблон Фабрика допомагає створювати об'єкти без прив'язки до конкретних класів.
- Шаблон структури (Structural Patterns) - ці шаблони допомагають організовувати об'єкти та класи в більш складні структури. Наприклад, паттерн Декоратор допомагає додавати нову функціональність до об'єкту, не змінюючи його основної структури.
- Шаблон поведінки (Behavioral Patterns) - допомагають вирішувати проблеми, пов'язані зі співпрацею між об'єктами.

Наприклад, паттерн Стратегія допомагає вибирати різні алгоритми в залежності від контексту.

Для розробки ігор використовують відомий у веб-розробці шаблон проектування MVC який дозволяє розділити логіку програми на три компоненти: модель (що відповідає за даних і бізнес-логіку), представлення (яке відображає дані користувачу) та контролер (який приймає вхідні дані та взаємодіє з моделлю для оновлення представлення).

Шаблон проектування MVC допомагає відокремити головні компоненти гри, такі як логіку вводу, гравець та інтерфейс користувача, та забезпечує легкість розробки та зберігання коду в хорошому порядку. Він також забезпечує зручний спосіб тестування окремих компонентів гри.

MVC є одним з найпопулярніших шаблонів проектування, який виник у світі розробки програмного забезпечення ще в 1970-х роках. Він створений для розробки графічних оболонок мови програмування Smalltalk.

Розробники гри використовують стандартизовані фреймворки та бібліотеки, які містять реалізації різних паттернів проектування та інших компонентів, зменшуя час, необхідний для розробки гри, дозволяючи швидше переходити до

розробки головної логіки гри.

Castlevania : Castlevania була однією з перших великих платформних ігор для консолі Nintendo (називається NES). Цей двовимірний платформер розповідає про мисливця на вампірів, який шукає вампірів у замку Дракули. Він розділений на 18 етапів, де персонаж повинен пройти кожен рівень, перш ніж закінчиться здоров'я, щоб дійти до кінця та зіткнутися з фінальним босом.

Donkey Kong : Donkey Kong спочатку був аркадною грою, перш ніж перейти на ігрові консолі. У грі персонаж Маріо намагається піднятися по драбині, повній перешкод, і врятувати принцесу Піч, а мавпа Донкі Конг кидає на нього бочки згори. Це була перша гра на платформі, яка містила стрибки.

Mirror's Edge : Mirror's Edge — це 3D-платформер, який надає гравцеві перспективу від першої особи, дозволяючи йому дивитися очима персонажа, який маневрує через запаморочливі висоти та складні перешкоди.

3.4 Розробка діаграми класів продукту

У програмній інженерії діаграма класів уніфікованої мови моделювання (UML) — це статична структурна діаграма, яка описує структуру системи, показуючи системні класи, їх властивості, операції (або методи) і зв'язки між об'єктами.

Діаграми класів є основним будівельним блоком об'єктно-орієнтованого моделювання. Використовується для загального концептуального моделювання структури програми та детального моделювання для перетворення моделей у програмний код. Діаграми класів також можна використовувати для моделювання даних. Класи на діаграмі класів представляють основні елементи програми, взаємодії та класи, що програмуються.

На схемі ці класи представлені вікнами, що містять три відділення:

У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.

Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а

перша буква мала.

У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи багато класів ідентифікуються і групуються в схему класів, що допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проектування часто поділяють на підкласи.

Залежність — це семантичний зв’язок між залежними елементами моделі та незалежними елементами. Він існує між двома елементами, якщо зміна, визначена одним елементом (сервером або цільовим), може спричинити зміну іншого елемента (клієнта чи джерела). Це об’єднання одностороннє. Залежності показані пунктирними лініями з порожнистою стрілкою, що вказує від замовника до постачальника.

Для подальшого опису поведінки системи ці діаграми класів можуть бути доповнені діаграмами станів або автоматами станів UML.

Асоціація являє собою серію посилань. Бінарні асоціації (з двома кінцями) зазвичай представляють у вигляді рядків. Асоціація може пов'язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціації можна назвати, а кінці асоціації можна прикрасити іменами ролей, індикаторами власності, множинністю, видимістю та іншими атрибутами.

Є чотири різні типи асоціацій: дво направлені, односпрямовані, агрегатні (включаючи комбінаторні агрегати) і рефлексивні. Найбільш поширеними є дво направлені та односпрямовані асоціації.

Наприклад, категорія польоту пов’язана з категорією літака з двостороннім рухом. Асоціація являє собою статичні відносини, спільні між об'єктами двох класів.

Агрегації є різновидом відношення «has»; агрегації є більш конкретними, ніж асоціації. Це асоціація, яка представляє частину мети або частину відносин.

Як тип асоціації, агрегати можуть бути названі й мати те саме прикраси, що й асоціації.

Однак агрегат не може включати більше двох класів; це має бути бінарна асоціація. Крім того, у процесі впровадження існує невелика різниця між агрегацією та асоціацією, і графіки можуть повністю опускати зв’язки агрегації.

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але класи, що містяться, не мають сильної залежності від часу життя контейнера. Коли контейнер знищено, вміст контейнера все ще існує.

В UML він представлений графічно у вигляді порожнистого ромба на класі хоста, з'єднаного з класом хоста лінією. Агрегат є семантично розширеним об’єктом і вважається одиницею в багатьох операціях, навіть якщо він фізично складається з кількох менших об’єктів.

Приклад: бібліотека та студенти. Тут студенти можуть існувати без бібліотеки, а зв’язок між студентами та бібліотекою є сукупністю.

Це вказує на те, що один із двох споріднених класів (підкласів) вважається особливою формою іншого (супертипу), а суперклас вважається узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу також є екземпляром суперкласу. Типове дерево узагальнень цієї форми можна знайти в біологічних класифікаціях: людина — це підклас людиноподібних мавп, людиноподібні мавпи — підклас ссавців тощо. Найкраще цей зв’язок можна зрозуміти за допомогою фрази «А є Б» (люди — ссавці, а ссавці — тварини).

Графічне представлення узагальнення UML — це форма порожнистого трикутника в кінці суперкласу рядка (або дерева рядків), що з’єднує його з одним або кількома підтипами.

Відносини узагальнення також відомі як відносини успадкування або відносини «є».

Суперклас (базовий клас) у відносинах узагальнення також називають «батьківським класом», суперкласом, базовим класом або базовим класом.

Підтип у відносинах спеціалізації також називається "дочірнім" підкласом, похідним класом, похідним типом, успадкованим класом або успадкованим типом.

Відносини зовсім не схожі на біологічні відносини між батьком і дитиною: використання цих термінів дуже поширене, але воно може ввести в оману.

А є типом В

Наприклад, «дуб — це дерево», «автомобіль — транспортний засіб»

Узагальнення можна відобразити лише в діаграмах класів і діаграмах використання.

У моделюванні UML відносини реалізації — це відносини між двома елементами моделі, де один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, задану іншим елементом моделі (постачальником).

Графічне представлення реалізації UML — це порожнистий трикутник на кінці штрихового інтерфейсу (або дерева ліній), який з'єднує його з одним або кількома реалізаторами. Проста стрілка використовується в кінці інтерфейсу з пунктирами, щоб підключити його до користувача. На діаграмах компонентів використовується графічна умова «м'яч і розетка» (реалізатор розміщує кульку або льодяник, а користувач показує сокет). Реалізації можна показати лише на діаграмах класів або компонентів. Реалізації – це зв'язки між класами, інтерфейсами, компонентами та пакетами, які з'єднують елементи замовника з елементами постачальника. Відношення реалізації між класом/компонентом та інтерфейсом вказує на те, що клас/компонент реалізує операції, запропоновані інтерфейсом.

Залежність — це слабкіша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Клас залежить від іншого класу, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації властивостей залежних класів, які є екземплярами незалежних класів. Іноді відносини між двома класами слабкі. Вони взагалі не реалізуються зі змінними-членами. Натомість вони можуть бути реалізовані як аргументи для функцій-членів.

Ці відносини часто описують як «А має Б» (мама з кошенятами, кошенята з мамою).

Представлення UML асоціації - це лінія, що з'єднує два пов'язані класи. На кожному кінці рядка є додаткові позначення.

Наприклад, ми можемо вказати, використовуючи наконечник стрілки, що загострений кінець видно з хвоста стрілки. Ми можемо вказати власність шляхом розміщення кульки, ролі, яку відіграють елементи цього кінця, вказавши ім'я ролі та множинність екземплярів цієї сутності (діапазон кількості об'єктів, які беруть участь в асоціації з точки зору іншого кінця).

Діаграма класів системи наведено на рис. 3.1.

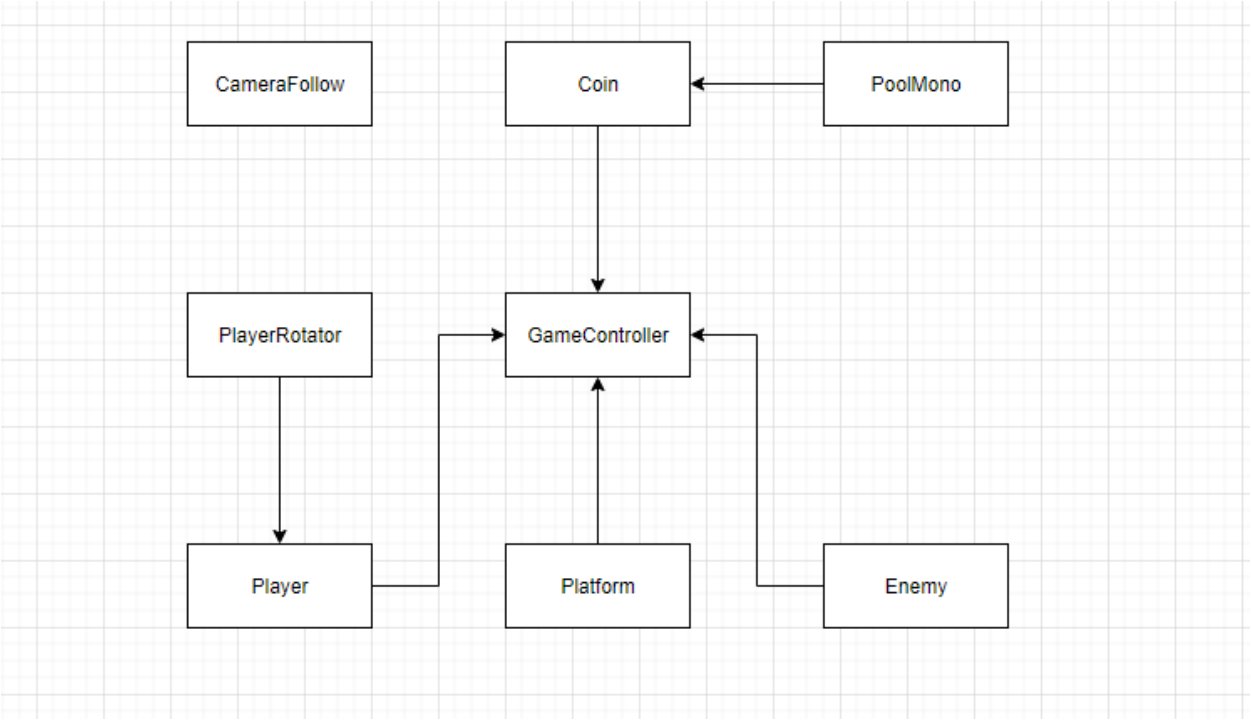


Рисунок 3.1 — Діаграма класів системи

Класи сутності моделюють довготривалу інформацію, якою обробляє система, а іноді і поведінку, пов'язану з цією інформацією. Їх не слід ідентифікувати як таблиці баз даних чи інших сховищ даних.

Вони намальовані як кола з короткою лінією, прикріпленою до нижньої частини кола. Як варіант, їх можна намалювати як звичайні класи із позначенням стереотипу «сутність» над назвою класу.

4 ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

4.1 Розробка діаграми предметної галузі

Діаграма предметної галузі наведено на рис.4.1, відома як діаграма сутностей-зв'язків та є інструментом моделювання, що використовується для візуалізації та опису структури і зв'язків в предметній галузі. Вона допомагає розуміти потреби та вимоги до системи або проекту.

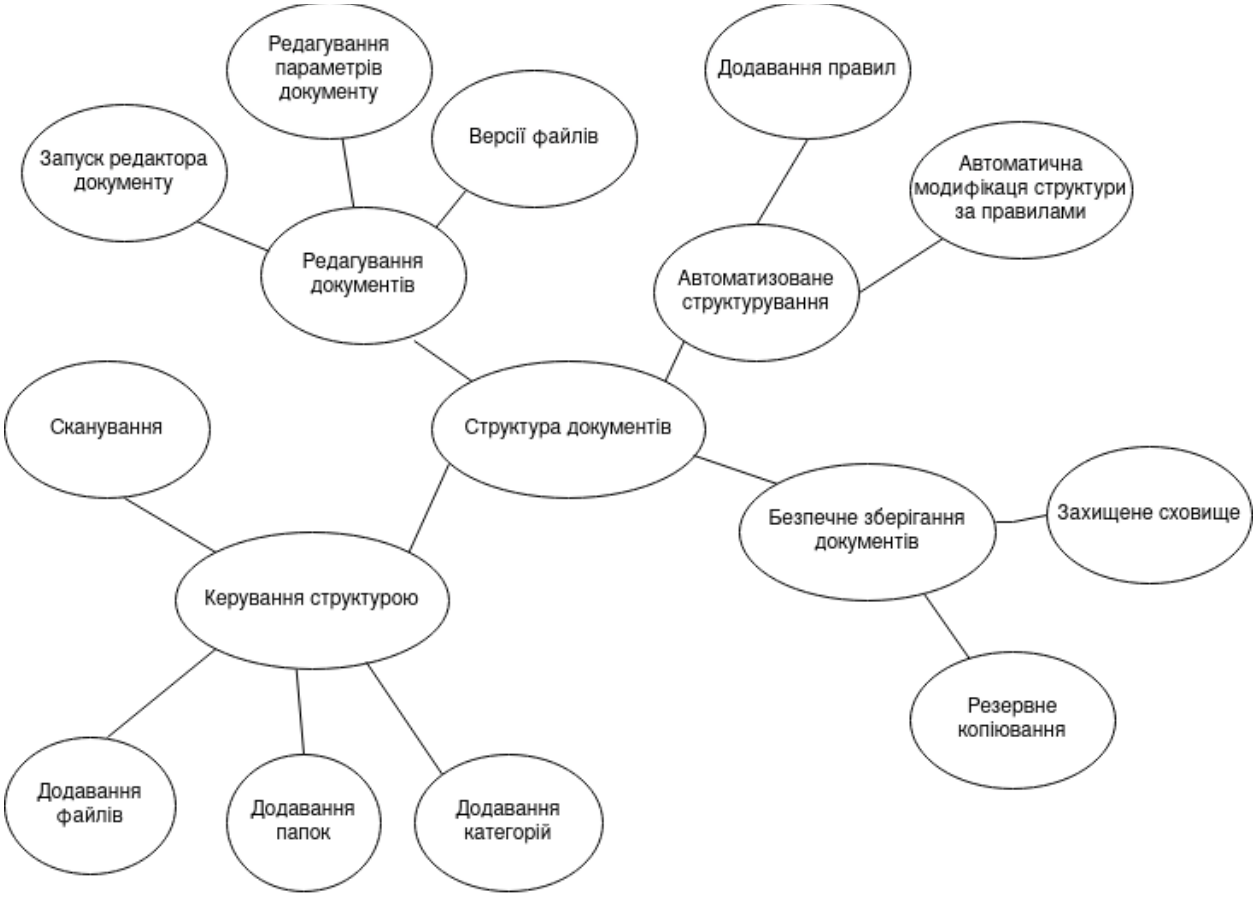


Рисунок. 4.1. – Діаграма предметної галузі

За даною діаграмою, можна помітити що функція «Структура документів» є головною, оскільки на ній базується вся робота з документами. Вона має похідні функції, які забезпечують продуктивну роботу з структурою документів.
«Керування структурою» є одною з найважливіших функцій оскільки відповідає за

коригування структури що напряму впливає на ефективність роботи з документами.

Містить наступний спектр підфункцій:

Додавання файлів;

Додавання категорій;

Додавання папок;

Сканування комп’ютеру;

«Безпечне зберігання документів» визначає спектр підфункцій, які відповідають за безпечне зберігання документів. Містить наступні пункти:

Резервне копіювання;

Захищене сховище;

Функція «Редагування документів» також є одною з визначних, оскільки дозволяє реалізувати весь потенціал структури документів за рахунок швидкого доступу до редактору документів. Таким чином визначений мінімальний набір підфункцій:

Запуск редактора документу;

Редагування параметрів документу;

Версії документу;

«Автоматизоване структурування» є функцією котра забезпечує автоматизацію організації великої кількості документів, містить наступні підфункції:

Додавання правил;

Автоматична модифікація структури за правилами;

4.2 Модель прецедентів

UML-діаграма прецедентів, також відома як Use-Case Diagram, моделює поведінку системи. Ця діаграма відображає взаємодії між системою та її учасниками, які називаються акторами. На діаграмах варіантів використання та дійових осіб описується, що система робить і як учасники її використовують, але

не відображається внутрішня робота системи.

Діаграма прецедентів наведено на рисунку.4.2.

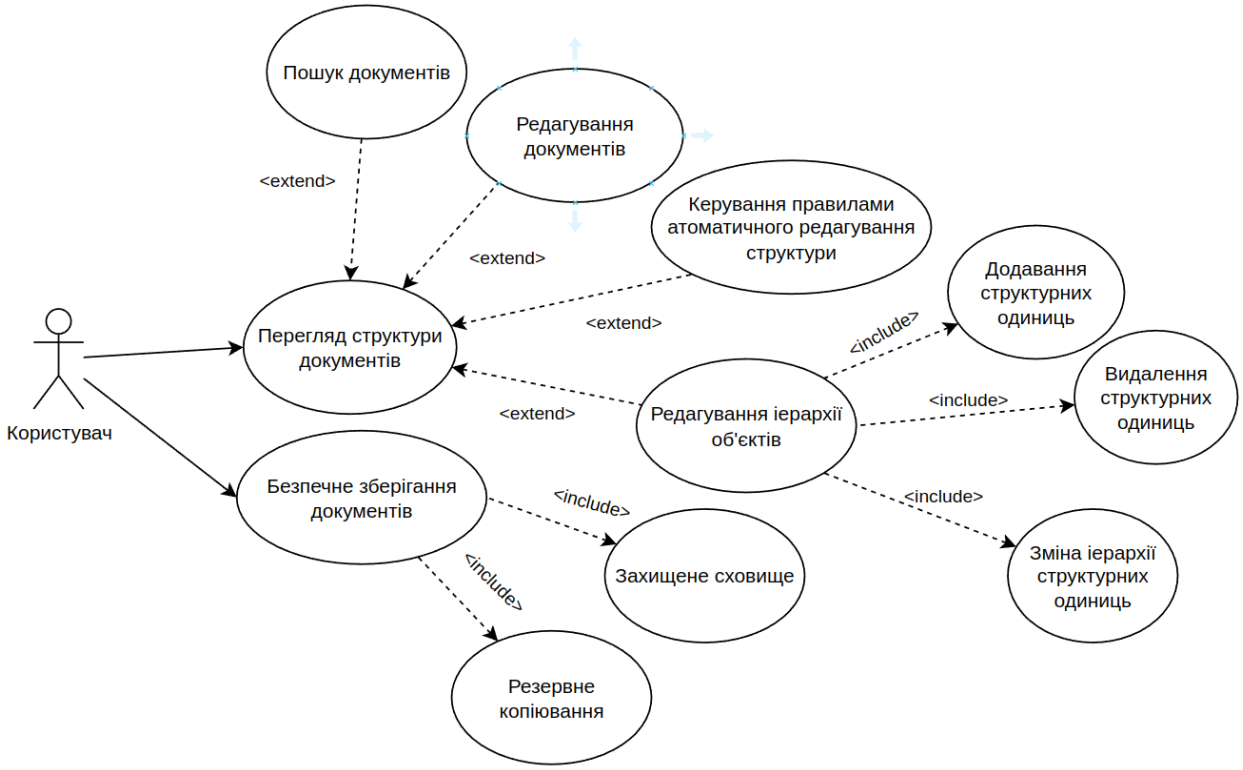


Рисунок. 4.2. – Діаграма прецедентів

За даною діаграмою можна побачити, що для сутності користувача основні сценарії використання це «Безпечне зберігання документів» та «Перегляд структури документів».

Сценарій «Безпечне зберігання документів» котрий включає в собі такі можливості як «Резервне копіювання» та «Захищене сховище».

Головним сценарієм є «Перегляд структури документів», він містить в собі можливість «Редагування структури документів» яка має сценарії з модифікації «Додавання структурних одиниць», «Видалення структурних одиниць», «Додавання структурних одиниць» та можливість «Керування правилами автоматичного редагування» що містить в собі звичайні сценарії модифікації. Також головний сценарій містить в собі допоміжні можливості «Пошуку документів» та «Редагування документів».

В UML 2 існує дві основні категорії діаграм: діаграми структури та діаграми поведінки. Кожна діаграма UML належить до однієї з цих двох категорій діаграм. Призначення структурних діаграм — показати статичну структуру системи, що моделюється. Вони включають діаграми класів, компонентів і/або об’єктів.

З іншого боку, діаграми поведінки показують динамічну поведінку між об’єктами в системі, включаючи такі речі, як їхні методи, співпраця та дії. Прикладами діаграм поведінки є діаграми діяльності, варіантів використання та послідовності.

Структурні діаграми показують статичну структуру системи, що моделюється. зосередження на елементах системи, незалежно від часу. Статичну структуру передають, показуючи типи та їх екземпляри в системі. Окрім відображення типів систем та їх екземплярів, діаграми структури також показують принаймні деякі зв’язки між цими елементами та потенційно навіть показують їхню внутрішню структуру.

Протягом життєвого циклу програмного забезпечення структурні діаграми корисні для різних членів команди. Загалом, ці діаграми дозволяють перевіряти проект і спілкуватися між окремими особами та командами. Наприклад, бізнес-аналітики можуть використовувати діаграми класів або об’єктів для моделювання поточних активів і ресурсів підприємства, таких як облікові книги, продукти або географічна ієрархія.

Архітектори можуть використовувати діаграми компонентів і розгортання, щоб перевірити/підтвердити, що їхній проект правильний. Розробники можуть використовувати діаграми класів для розробки та документування закодованих (або незабаром) класів системи.

UML 2 розглядає структурні діаграми як класифікацію; не існує самої діаграми, яка називається "схемою структури". Однак діаграма класів пропонує яскравий приклад типу структурної діаграми та надає нам початковий набір елементів нотації, які використовують усі інші структурні діаграми.

Мета діаграми класів полягає в тому, щоб показати типи, що моделюються в системі. У більшості моделей UML ці типи включають: UML використовує спеціальну назву для цих типів: «класифікатори».

Загалом ви можете думати про класифікатор як про клас, але технічно класифікатор є більш загальним терміном, який також відноситься до трьох інших типів.

Однак корисність діаграми компонентів охоплює весь термін служби системи. Діаграми компонентів є безцінними, оскільки вони моделюють і документують архітектуру системи. Оскільки діаграми компонентів документують архітектуру системи, розробники та потенційні системні адміністратори системи вважають цей робочий продукт критично важливим для того, щоб допомогти їм зрозуміти систему.

4.3 Розробка основних механік

Однією з основних механік можна вважати механіку роботи загального контроллера. Він відповідає за вивід всіх даних на екран, переміщення персонажа та спавн платформ, ворогів і монет.

Його код приведено в додатку В.

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using Random = UnityEngine.Random;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
   [Header("Generatio
```

```
[SerializeField]
private Platform platformPrefab;
[SerializeField]
private Platform backPlatform;
[SerializeField]
private Platform currentPlatform;
public Platform CurrentPlatform => currentPlatform;
[SerializeField]
private Platform nextPlatform;
public Platform NextPlatform => nextPlatform;
[SerializeField]
private float minDist;
[SerializeField]
```

4.4 Розробка графічного інтерфейсу.

Гра починається з меню, в якому вказано найкращий рахунок та показано, якими кнопками виконується управління рисунок. 4.4.

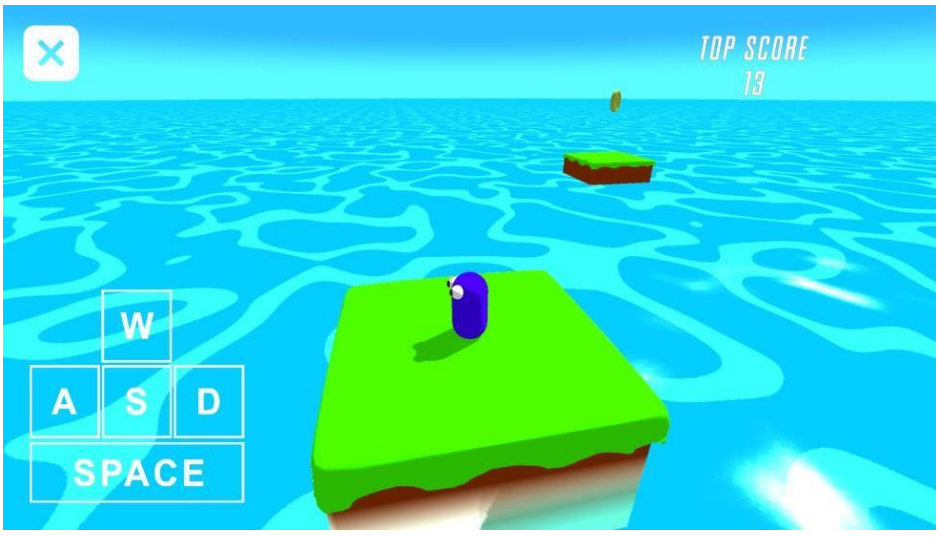


Рисунок. 4.4 Меню

Ігровий процес проходить у вигляді перестрибання з одної платформи на іншу, які випадково генеруються навколо персонажа, уникання ворогів та збирання монет з метою підвищення рахунку. Ігровий процес зображено на рисунку 4.5.

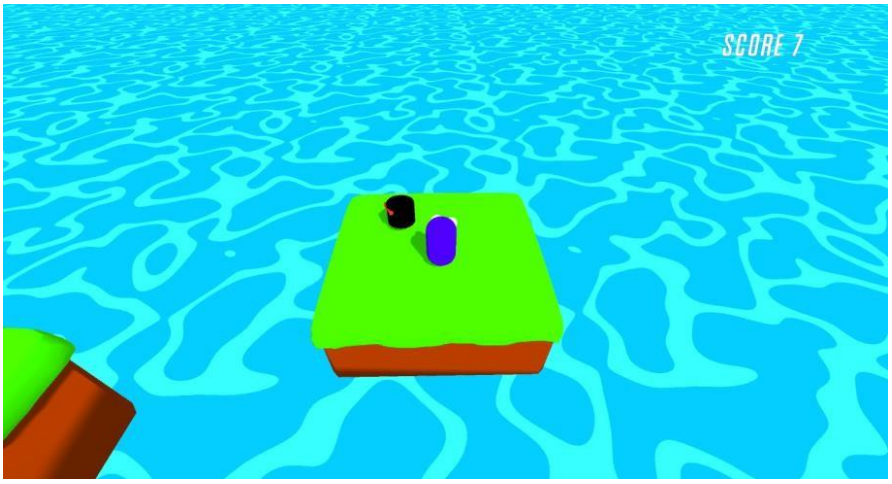


Рисунок. 4.5 Ігровий процес

При програші користувачеві показується екран програшу наведений на рисунку. 4.6.

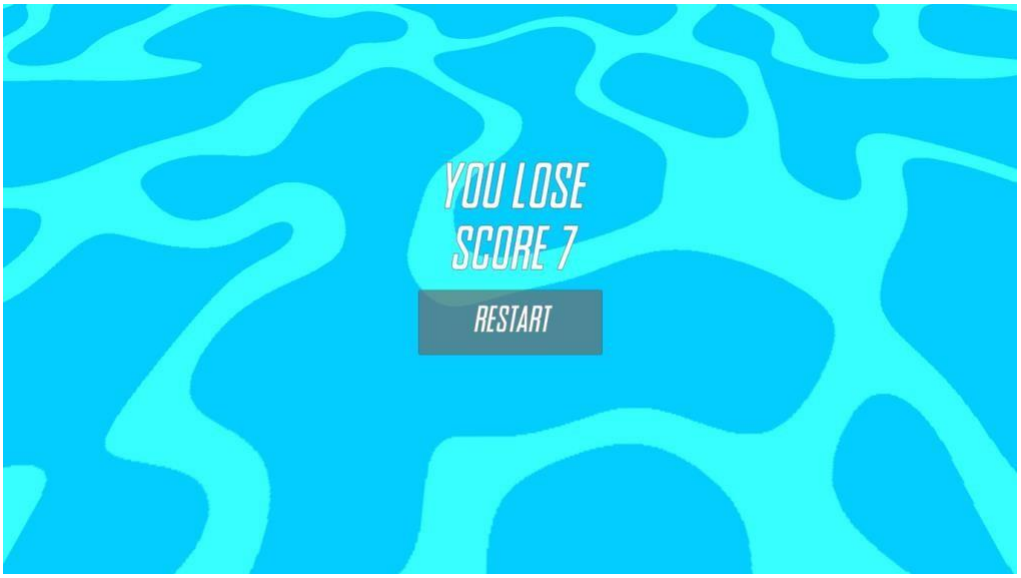


Рисунок. 4.6 Екран програшу

4.5 Тестування гри

У цьому розділі наведено огляд багатьох підходів до тестування програмного забезпечення, які зараз доступні. Нижче наведено дві категорії, які використовуються для класифікації методів тестування програмного забезпечення:

Ручне тестування, часто відоме як статичне тестування: процес тестування є тривалим і трудомістким. З точки зору статистики, це здійснюється на більш ранній стадії життєвого циклу. У деяких колах це також називають «статичним тестуванням». Його виконують аналітики, розробники та тестувальники. Наведемо деякі з різноманітних підходів ручного тестування, які доступні: перегляд інформації, екскурсія об’єктом, вивчення технічних аспектів та іспит.

Тестування, яке є динамічним (також відоме як автоматизоване тестування): цей тестер проводить тестування, виконуючи сценарій на інструменті тестування та виконуючи процедури.

Існує кілька різних типів автоматизованого тестування, і один із них називається динамічним тестуванням.

Додаткові типи автоматизованого тестування включають такі категорії та підтипи: тестування, щоб визначити, чи є воно точним, як швидко його можна виконати, наскільки воно безпечне та як довго воно може тривати. Тестування на правильність, продуктивність, безпеку та надійність — це назви цих різних видів.

Тестування на коректність вимагає, щоб програмне забезпечення принаймні відповідало фундаментальному рівню очікуваної точності. При перевірці правильності необхідно буде звернутися до оракула, щоб відрізнити допустимі і небажані форми поведінки. Тестер міг раніше знати про внутрішню роботу програмного модуля, який зараз тестується, хоча це не гарантовано.

Тестування білого ящика перевіряє не лише те, наскільки добре програма виконує свої призначені функції, але й те, наскільки добре вона відповідає специфікаціям дизайну. Тестування в середовищі білого ящика вимагає глибокого розуміння програмування. Тести білого ящика також часто називають тестуванням «прозорих ящиків» або «скляних ящиків».

Цю методологію можна використовувати для тестування на рівні пристрою, інтеграції та системи. Іншими словами, він встановлює, чи захищені дані інформаційними системами та чи працюють вони ефективно. У результаті він може перевірити окремі шляхи, якими може йти модуль. Усі цикли та структури даних перевіряються в процесі тестування.

Тестування білого ящика – це складний метод тестування, який також вимагає програмування. Його проводять професіонали-тестувальники. Він здійснює ретельний огляд і перевірку коду шляхом вивчення модуля.

Якщо вихідні дані не відповідають попереднім вимогам, їх буде повторно скомпільовано та перевірено повторно.

Оцінюється чистий код програми, а не її інтерфейс користувача чи візуальне представлення.

Цей вид тестування перевіряє, як програмне забезпечення зібрано всередині та передбачає надання вхідних даних системі для створення виходу.

Тестер повинен бути знайомий з вихідним кодом. Тестування в середовищі білого ящика є важливим для всіх інших типів тестування, включаючи інтеграційне, модульне та системне тестування.

Він гарантує, що всі тестові об’єкти та компоненти виконуються належним чином. Тестування чорної скриньки: воно базується на аналізі специфікацій частини програмного забезпечення, а не на тому, як програма насправді працює всередині.

Метою цього тесту є визначення того, наскільки ефективно компонент задовольняє критерії, встановлені для компонента. Він приділяє базовій логічній структурі системи або дуже мало, або взагалі не приділяє уваги, натомість повністю зосереджується на найважливіших характеристиках системи. Це гарантує успішне отримання вхідних даних і правильне формування вихідних даних. Під час тестування чорної скриньки зберігається цілісність зовнішніх даних.

Тестування «чорного ящика» може стосуватися різноманітних типів тестування, включаючи дослідницьке тестування, функціональне тестування, стрес-тестування, тестування навантаження, дослідницьке тестування, тестування

зручності використання, тестування диму, тестування відновлення та тестування обсягу.

Тести та оцінювання (альфа- та бета-версія) Деякі альтернативи традиційному тестуванню чорної скриньки включають тестування на основі графів і розподіл еквівалентності, порівняльне тестування, ортогональний масив і порівняльне тестування, спеціалізоване тестування, тестування фазз та метрики простежуваності.

Тестування за допомогою сірих ящиків Тестування сірих ящиків — це змішаний підхід, який включає тестування як білого, так і чорного ящиків.

Мета тестування сірого ящика полягає в тому, щоб переконатися, що програмне забезпечення відповідає його специфікаціям, а також надати розуміння його внутрішньої роботи. Це передбачає використання зворотного проектування для виявлення повідомлень про помилки або граничних значень.

Метод оцінки програмного забезпечення, усвідомлюючи код або логіку, що лежить під поверхнею, відомий як тестування сірого ящика. Тестування у сірій скриньці вимагає більш повного знання внутрішньої роботи програми. Тести, проведені в чорному або сірому ящику, не забезпечують перевірку всіх внутрішніх компонентів системи.

Незважаючи на те, що інфраструктура для створення та тестування програмного забезпечення постійно вдосконалюється, її важливість для процесу неможливо переоцінити.

Тестування - це процес перевірки, валідації і оцінки програмного продукту чи системи з метою виявлення дефектів, помилок, недоліків та забезпечення їх якості, надійності та відповідності заданим вимогам. В контексті відеоігор, тестування гри є важливою складовою процесу розробки, що включає перевірку різних аспектів гри, їх функціональності та якості. Основна мета тестування гри полягає в забезпеченні якості геймплею та задоволення користувачів. Воно включає в себе виконання різноманітних дій та сценаріїв гри з метою перевірки правильності функціонування, виявлення помилок та недоліків у грі, а також перевірку її стабільності, продуктивності та взаємодії з гравцем.

Процес тестування може включати ручне тестування, коли тестувальник вручну виконує різні дії та сценарії гри, а також автоматизоване тестування, коли використовуються спеціальні програми і скрипти для автоматичного виконання тестів та перевірки різних аспектів гри.

Тестування гри включає такі аспекти, як:

Функціональність

Перевірка роботи всіх функцій, правил та механік гри. Включає перевірку головних ігрових елементів, таких як керування, рух персонажів, взаємодія з об'єктами та інші геймплейні аспекти.

Стабільність

Перевірка стійкості гри та її здатності працювати без збоїв, відвалів або падінь фреймрейту. Включає тестування на різних платформах і пристроях, забезпечуючи, що гра працює стабільно і безперервно.

Взаємодія та інтерфейс

Перевірка взаємодії гравця з грою, включаючи елементи управління, інтерфейс користувача, навігацію та інші аспекти взаємодії. Тестування цього аспекту включає перевірку реагування гри на введення гравця, відображення інформації на екрані, коректність меню та інтерфейсних елементів.

Мережева функціональність

Якщо гра має режим мультиплеєра або мережевий режим, вона підлягає спеціальному тестуванню, щоб перевірити функціональність, стабільність та продуктивність мережевого з'єднання. Тестування включає перевірку синхронізації гравців, передачу даних, тестування серверів та виявлення можливих проблем з мережевим з'єднанням.

Тестування продуктивності

Перевірка продуктивності гри, такої як швидкість завантаження, час реакції, оптимізація пам'яті та ресурсів.

Тестування сумісності

Перевірка сумісності гри з різними операційними системами, пристроями та обладнанням. Це включає тестування на різних версіях операційної системи, різних

пристроях, розширеннях та інших факторах, які можуть впливати на сумісність гри.

Тестування витривалості

Перевірка здатності гри протягом тривалого періоду часу або в умовах постійного використання. Це включає проведення довготривалих сеансів гри, тестування стійкості гри під високим навантаженням та виявлення можливих проблем, пов'язаних зі збоєм, перегріванням або іншими чинниками, які можуть впливати на витривалість гри.

Локалізаційне тестування

Перевірка гри на наявність і правильність перекладів, локалізованого контенту, відповідності місцевим культурним нормам та вимогам. Це включає перевірку тексту, графіки, озвучення та інших аспектів, що піддаються локалізації, для забезпечення якості гри на різних мовах та ринках.

Тестування відмовостійкості

Перевірка поведінки гри в умовах відмови або непередбачуваних ситуацій, таких як відключення мережі, збої у зв'язку, несправність обладнання тощо. Це включає перевірку реакції гри на такі ситуації, відновлення гри після відмови та забезпечення надійності гри в умовах, коли стаються непередбачувані події.

Тестування безпеки

Перевірка захищеності гри від можливих атак, вразливостей та витоку конфіденційної інформації. Це включає перевірку безпеки мережевого з'єднання, перевірку наявності захисту від шахрайства, зламу або незаконних дій. Усі ці етапи тестування вимагають детального планування, систематичного підходу та досвіду. Тестування гри допомагає виявити і усунути помилки, забезпечити високу якість гри, зробити її стабільною та задовольняючою для користувачів.

В процесітестування використовуються різні методи і техніки, такі як функціональне тестування, тестування користувацького досвіду, регресійне тестування, тестування завантаження та інші. Під час тестування гри використовуються різні засоби і інструменти, такі як спеціалізовані тестувальні програми, автоматизація тестування, збор даних про виконання гри (логи), засоби для відстежування дефектів (баг-трекери) та інші.

Також можуть використовуватись спеціально наймані тестувальники або команди тестування, які працюють у співпраці з розробниками гри для виявлення й виправлення проблем.

Тестування гри відбувається на різних етапах розробки, починаючи з тестування окремих функцій та компонентів гри, до повного тестування всієї гри перед її випуском. Це дозволяє виявити та виправити помилки на ранніх етапах, забезпечуючи більшу якість та задоволення від гри. Тестування гри також враховує різні платформи, на яких вона буде запускатись, такі як комп'ютери, консолі, мобільні пристрої тощо.

Кожна платформа може мати свої особливості, обмеження та вимоги, тому тестування включає перевірку сумісності гри зрізними платформами та впевненість, що гра працює на них належним чином. У підсумку, тестування гри єважливим етапом процесу розробки і випуску відеоігор, спрямованим на забезпечення якості, стабільності та задоволення користувачів. Воно допомагає виявити та виправити помилки, проблеми з функціональністю та взаємодією, а також забезпечити відповідність гри вимогам та очікуванням гравців.

Додаткові аспекти, які можна включити до опису тестування гри:

Тестування штучного інтелекту (AI)

Перевірка поведінки і реакцій ігрових персонажів, керованих комп'ютером, і їх здатності адаптуватись до різних ситуацій у грі. Тестування AI включає перевірку алгоритмів прийняття рішень, реалістичності поведінки персонажів та балансу гри.

Тестування мультимедійного контенту

Перевірка візуальних ефектів, анімації, звукового супроводу та інших аспектів мультимедійного вмісту гри. Тестування мультимедіа включає перевірку якості графіки, плавності анімації, наявності інтерактивних звукових ефектів та інших аспектів, які створюють атмосферу гри.

Тестування ігрового середовища

Перевірка локацій, рівнів, карт, об'єктів та інших елементів, які створюють ігрове середовище. Тестування середовища включає перевірку наявності та

коректності колізій (зіткнень), взаємодії з об'єктами та поведінки ігрових елементів у різних ситуаціях.

Тестування збереження даних

Перевірка функції збереження прогресу гри, налаштувань, досягнень і статистики гравця. Тестування збереження даних включає перевірку коректності збереження та завантаження даних, роботи різних слотів збереження, можливості синхронізації даних між різними пристроями та інших аспектів, пов'язаних із збереженням ігрових даних.

Загалом, тестування є невід'ємною частиною процесу розробки програмного забезпечення і допомагає забезпечити якість і надійність продукту, зниження ризиків та поліпшення користувацького досвіду.

Було здійснено тестування коректного натискання на кнопку виходи з гри.

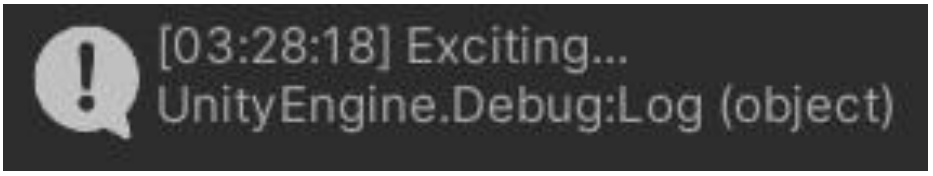


Рисунок 4.7 – Приклад коректного натискання на кнопку виходу

ВИСНОВКИ

У роботі проведено дослідження та розробка ігрового додатку за допомогою платформи Unity.

- Проведено огляд предметної області.
- Визначено, що таке розробка відеоігор.
- Визначено поняття платформера.
- Досліджено процес створення відеоігор.
- Обрано рушій Unity для створення ігрового додатку.
- Обрано мову програмування C#.
- Обрано середовище розробки Visual Studuo 2019.
- Спроектовано внутрішню будову.
- Розроблено основні механіки ігрового додатку.
- Проведено тестування.

Завдяки виконанню завдань отримано повноцінний ігровий додаток в жанрі платформера.

Виходячи з проведеного тестування, можна вважати, що додаток повністю функціональний і може використовуватися.

Цікавою ідеєю для майбутнього розширення можна вважати додавання більшої кількості рівнів з різноманітним дизайном, додавання нових ворогів і персонажів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Відеогра [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Відеогра
2. Історія відеоігор [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Історія_відеоігор
3. Media sapiens [Електронний ресурс]. – Режим доступу до ресурсу: https://ms.detector.media/internet/post/24912/2020-06-20-videoigry-tse-mystetstvo/
4. Korydor [Електронний ресурс]. – Режим доступу до ресурсу: http://korydor.in.ua/ua/opinions/iak-videoihry-staly-mystetstvom.html
5. Жанри відеоігор [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Жанри_відеоігор#:~:text=Британський%20експерт%20Стів%20Пул%20у,Пригоди%2C%20Рольова%20гра%20та%20Головоломка.
6. Tower Defense [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Tower_Defense
7. Wikiwand [Електронний ресурс]. – Режим доступу до ресурсу: https://www.wikiwand.com/simple/Tower_defense
8. MasterClass [Електронний ресурс]. – Режим доступу до ресурсу: https://www.masterclass.com/articles/tower-defense-game-video-game-guide
9. Plarium [Електронний ресурс]. – Режим доступу до ресурсу: https://plarium.com/en/blog/rpg-elements/
10. Unity [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Unity_(ігровий_рушій)
11. Unity Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://docs.unity.com
12. C# Documentation [Електронний ресурс]. – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/
13. Visual Studio Documentation [Електронний ресурс]. – Режим доступу до

ресурсу:

https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022

14. Blende [Електронний ресурс]. – Режим доступу до ресурсу: https://wiki.blender.org/wiki/Main_Page

15. Діаграма класів [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_класів

16. Діаграма діяльності [Електронний ресурс]. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_діяльності

ДОДАТКИ

ДОДАТОК А

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка ігрового додатку платформера «Jumping Slime 3D» за допомогою Unity мовою C#

Виконав студент 4 курсу
групи ПД-43
Овдієнко Марк Володимирович
Керівник роботи
К.ф-м наук, доц, доцент кафедри Садовенко Володимир Сергійович

Київ – 2023

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета – розширення функціональних можливостей та основних програмних засобів для спрощення ігрової механіки ігрового додатку в жанрі платформер.

Об’єкт – ігровий процес додатку платформера «Jumping Slime 3D» з використанням рушія Unity.

Предмет – програмне забезпечення для реалізації ігрового додатку платформера «Jumping Slime 3D».

.

ЗАДАЧІ БАКАЛАВРСЬКОЇ РОБОТИ

1. Проаналізувати предметну область ігрової індустрії.
2 .Обрати засоби реалізації програмного забезпечення.
3. Розробити структуру ігрового додатку за допомогою обраного програмного забезпечення .
4. Розробити ігровий додаток Jumping Slime 3D.
5. Провести тестування гри.

АНАЛІЗ АНАЛОГІВ

Назва гри	NinJump	Katana Zero	The Messenger	Cyber Shadow	Jumping Slime 3D
На мобільні платформи	+	-	-	+	+
Нескінченна гра	+	-	-	-	+
Переваги	Низький поріг входу, можна грати будь-де і будь-коли	Висока динаміка, різноманітні рівні, цікавий сюжет	Велика кількість ефектів, присутня механіка покращення ігрового спорядження	Цікавий сюжет, гра з логічним завершенням, низький поріг входу	Низький поріг входу, висока динаміка, нескінчений геймплей
Недоліки	Однотипна гра, швидко набридає, бракує різноманітності ігрових елементів	Гра орієнтована на досвідчених гравців, відсутня на мобільних пристроях	Відсутня на мобільних пристроях, занадто коротка	Мала кількість рівнів та ігрових елементів	Відсутній логічний кінець

КОНЦЕПТ ГРИ

1. Основний ігровий сюжет полягає в перестрибуванні з однієї платформи на іншу.
2. Гравець за допомогою клавіатури обирає траєкторію руху свого персонажа, щоб переміститись до іншого місця.
3. При проходженні платформ, потрібно уникати ворогів, які заважатимуть просуватися вперед.
4. За проходження кожної платформи гравцю нараховуються ігрові бали.
5. Швидкість генерації рівня і ворогів зростає з часом, що підвищує складність ігрового процесу.
6. Збирання розміщених на платформах монет, підвищує рахунок гравця та дає можливість перейти на вищій рівень складності.
7. Гравець може бити власні рекорди.

ВИМОГИ ДО ІГРОВОГО ДОДАТКУ

Функціональні

- Реалізувати основні механіки додатку платформера «Jumping Slime 3D», перестрибувати з однієї платформи на іншу.
- Забезпечити можливість управління ігровим персонажем, включаючи встановлення його позиції та вибір цілей.
- Реалізувати систему генерації платформ збільшенням складності, різноманітними ворогами та їхніми атаками.

Нефункціональні

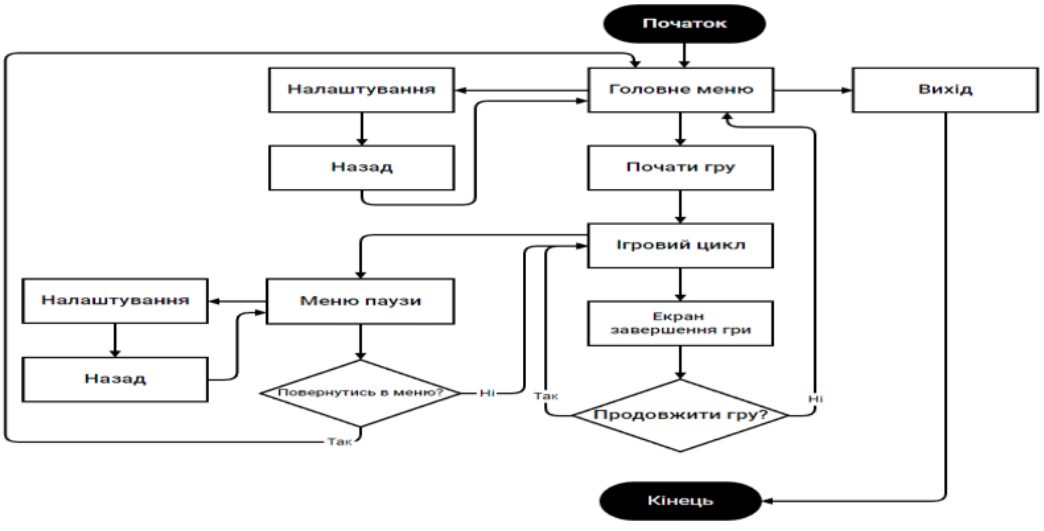
- Сумісність с пілатформою Віндус
- Мінімальний ФПС 30 кадрів
- Мінімальні вимоги:
- ОС: Windows Vista SP1+

Процесор: SSE2 instruction set support Оперативна пам’ять: 4 GB ОП
Відеокарта: DX10 (shader model 4.0) capabilities
Direct X: версії 10
Місце на диску: 282 MB доступного місця

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

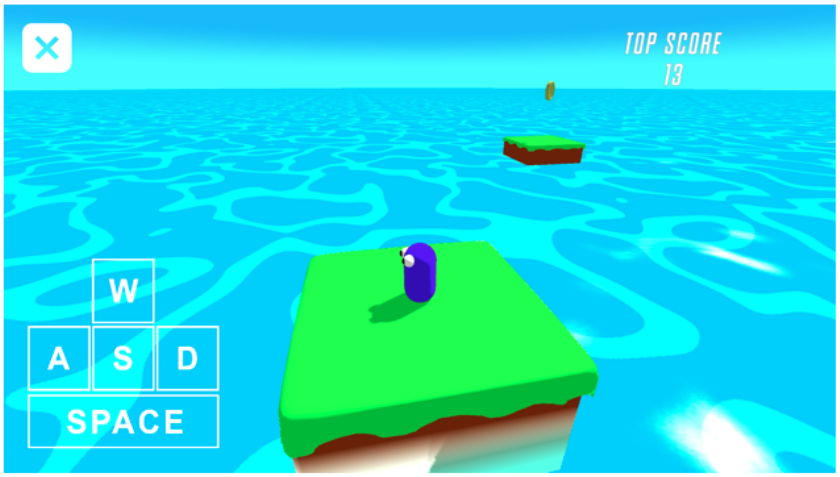
7

БЛОК-СХЕМА ІГРОВОГО ПРОЦЕСУ



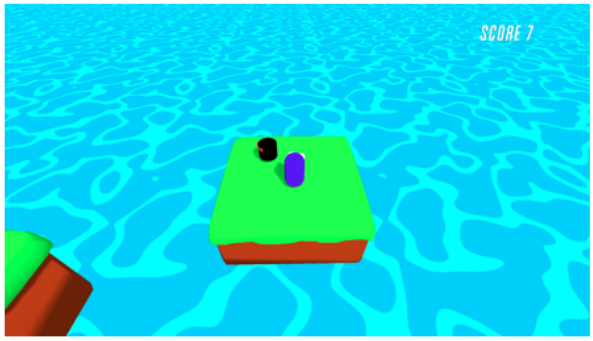
8

ЕКРАННІ ФОРМИ ІНТЕРФЕЙСУ

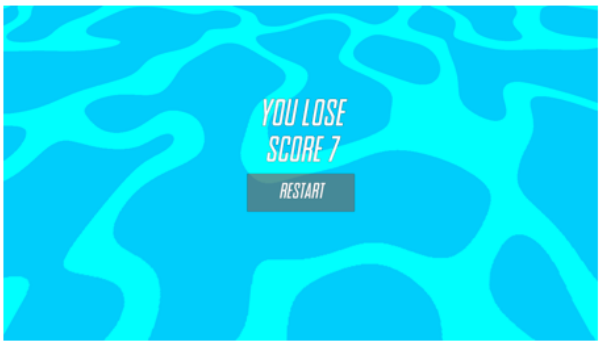


Меню ігрового додатку

ЕКРАННІ ФОРМИ



Ігровий процес



Екран програшу

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Садовенко В.С., Овдієнко М.В. Огляд середовища розробки «Microsoft Visual Studio»// Всеукраїнська науково-технічна конференція “Застосування програмного забезпечення в інфокомунікаційних технологіях” - Київ: ДУТ, 2023 - 128-129 с.;

2.Садовенко В.С., Овдієнко М.В. Особливості розробки ігрових додатків на Unity// Всеукраїнська науково-технічна конференція “Застосування програмного забезпечення в інфокомунікаційних технологіях” - Київ: ДУТ, 2023 - 130-131 с с..

ВИСНОВКИ

1. Проведено аналіз предметної області.
2. Досліджено роль динамічного геймплею в інтерактивних іграх, вирішено використовувати рушія Unity, мову C# та Visual Studio.
3. Розроблено структуру ігрового додатку за допомогою обраних програмних засобів.
4. Розроблено ігровий додаток Jumping Slime 3D.
5. Проведено тестування гри.

ДОДАТОК В

Розробка основних механік

Однією з основних механік можна вважати механіку роботи загального контроллера. Він відповідає за вивід всіх даних на екран, переміщення персонажа та спавн платформ, ворогів і монет.

Його код приведено нижче.

```
using System;
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using Random = UnityEngine.Random;
using UnityEngine.SceneManagement;
public class GameController : MonoBehaviour
{
    [Header("Generation")]
    [SerializeField]
    private Platform platformPrefab;
    [SerializeField]
    private Platform backPlatform;
    [SerializeField]
    private Platform currentPlatform;
    public Platform CurrentPlatform => currentPlatform;
    [SerializeField]
    private Platform nextPlatform;
    public Platform NextPlatform => nextPlatform;
    [SerializeField]
    private float minDist;
    [SerializeField]
```

```
private float maxDist;
[SerializeField]
private int minY;
[SerializeField]
private int maxY;
[Header("Ui")]
public bool isStartGame = true;
[SerializeField]
private GameObject gamePanel;
[SerializeField]
private GameObject startPanel;
[SerializeField]
private GameObject losePanel;
[SerializeField]
private TMP_Text textTopScore;
[SerializeField]
private TMP_Text textLoseScore;
private int score;
public int Score => score;
[SerializeField]
private TMP_Text textScore;
public static GameController Instance { get; private set; }

private void Awake()
{
    Application.targetFrameRate = 60;

    if (Instance == null)
    {
        Instance = this;
```

```
        }
        else
        {
            Destroy(gameObject);
        }
    }

    private void Start()
    {
        UpdateTopScoreText();
    }
    private void Update()
    {
        CheckStartUi();
    }

    private void CheckStartUi()
    {
        if (isStartGame)
        {
            if (Input.GetAxisRaw("Horizontal") != 0 || Input.GetAxisRaw("Vertical") != 0 ||
Input.GetAxis("Jump") != 0)
            {
                gamePanel.SetActive(true);
                startPanel.SetActive(false);
                losePanel.SetActive(false);
                isStartGame = false;
            }
        }
    }
```

```
private void UpdateTopScoreText()
{
    startPanel.SetActive(true);
    int topScore = PlayerPrefs.GetInt("TopScore",0);
    textTopScore.text = "TOP SCORE" + "\n" + topScore.ToString();
}

public void SpawnPlatform()
{
    DestroyPlatform();
    backPlatform = currentPlatform;
    Platform platform = Instantiate(platformPrefab
                , Vector3.zero
                , Quaternion.identity);

    Vector3 size = platform.SetRandomSize();
    platform.transform.position = GetRandomSpawnPos((int) size.x);
    platform.transform.localScale = size;
    platform.AcitavatePlatform();
    nextPlatform = platform;
    currentPlatform = nextPlatform;
}

public void ExitGame()
{
    Application.Quit();
}
private void DestroyPlatform()
{
```

```
    if (backPlatform != null)
    {
        Destroy(backPlatform.gameObject);
        UpdateScore();
    }
}

public void UpdateScore()
{
    score++;
    textScore.text = "SCORE "+score.ToString();
}

public void ActivateLosePanel()
{
    gamePanel.SetActive(false);
    startPanel.SetActive(false);
    losePanel.SetActive(true);
    textLoseScore.text = "SCORE "+score.ToString();
}

public void RestartGame()
{
    int topScore = PlayerPrefs.GetInt("TopScore", 0);
    isStartGame = true;
    if (GameController.Instance.Score > topScore)
    {
        PlayerPrefs.SetInt("TopScore", GameController.Instance.Score);
    }
    SceneManager.LoadScene(0);
```

```
    }
    private Vector3 GetRandomSpawnPos(int sizePlatform)
    {
        Vector3 rndVect3 = new Vector3(Random.Range(minDist,
maxDist),0f,Random.Range(minDist, maxDist));

        if (Random.Range(0, 2) == 1)
        {
            rndVect3.x *= -1;
        }
        if (Random.Range(0, 2) == 1)
        {
            rndVect3.z *= -1;
        }

        int rndY = Random.Range(minY, maxY);

        var pos = new Vector3(rndVect3.x, currentPlatform.transform.localScale.y + rndY,
rndVect3.z);

        pos += new Vector3(sizePlatform * Mathf.Sign(rndVect3.x), 0, sizePlatform *
Mathf.Sign(rndVect3.z)) / 2f;
        pos += new Vector3(currentPlatform.Size * Mathf.Sign(rndVect3.x), 0,
currentPlatform.Size * Mathf.Sign(rndVect3.z)) / 2f;
        pos += rndVect3;
        pos += currentPlatform.transform.position;

        return pos;
    }
}
```