ДОДАТОК А ГРАФІЧНА ЧАСТИНА

Харківський національний університет радіоелектроніки

Кафедра ЕОМ
Кваліфікаційна робота
Другий (магістерський) рівень

Методи та алгоритми портування ігор

Виконав:
Маматов А.О.,
ст. гр. СПм – 22 – 1

Керівник:
Фесенко Т.Г.,
проф. каф. ЕОМ

Об'єкт, предмет, мета і завдання

Метою кваліфікаційної роботи є дослідити існуючі методи портування розробленої гри в середовищі Unity з підвищенням продуктивності та ефективності, та удосконалити їх.

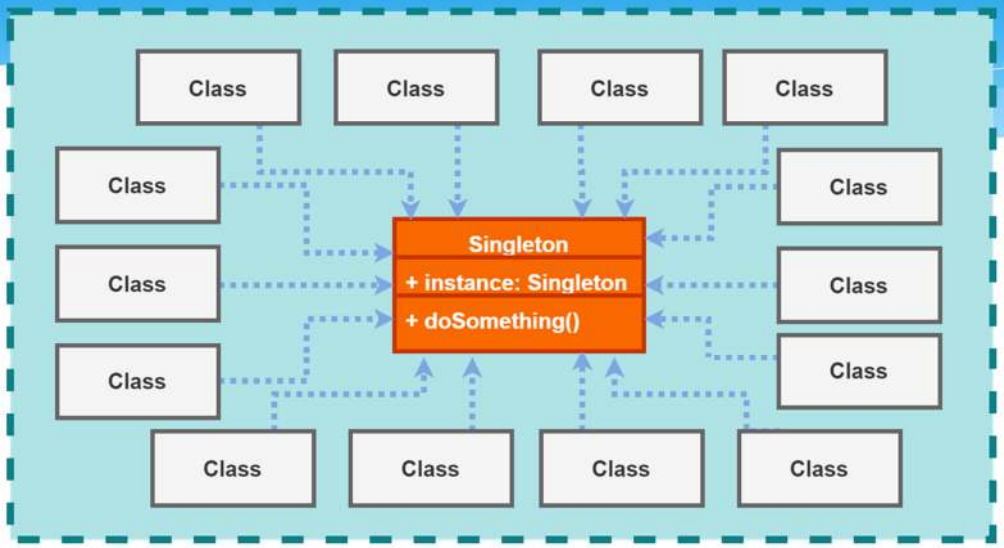
Основні завдання кваліфікаційної роботи наступні:

- Оглянути існуючі рішення для портування ігор;
- Дослідити вплив портування на продуктивність гри на платформі;
- Розробити та удосконалити метод портування Dependency Injection;
- Протестувати застосування удосконаленого методу оптимізації на прикладі ігрового додатку на платформі Unity.

Портування ігор

Портування ігор – це процес адаптації вже існуючої відеогри, розробленої для одної платформи, для ігрових консолей, комп'ютерів або мобільних пристроїв. Основна мета портування – забезпечити можливість грати в гру на різних платформах, розширюючи її аудиторію.

Метод для портування Zenject+DI+Singleton



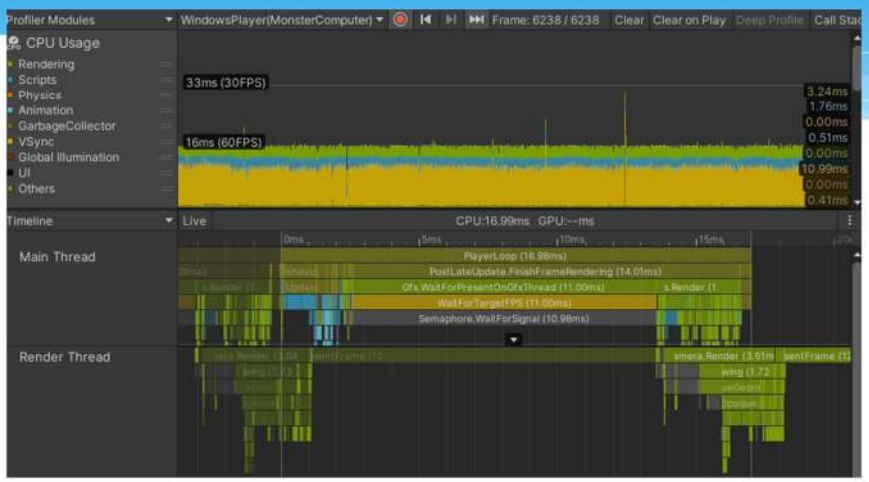
4

Оптимізація

Оптимізація відіграє важливу роль у багатьох сучасних іграх, і вона може суттєво відрізнятися на різних платформах. Вона є ключовим аспектом портування. Використання різних рівнів деталізації, адаптація текстур і роздільної здатності дозволяють досягти оптимального співвідношення якості та продуктивності. Оптимізація гри на Unity є необхідною складовою процесу створення гри. Просто створити гру недостатньо. Важливо забезпечити, щоб гра працювала на пристроях користувачів і вимагала мінімальну кількість ресурсів. Продуктивність гри прямо залежить від того, як багато ресурсів пристрою вона використовує.

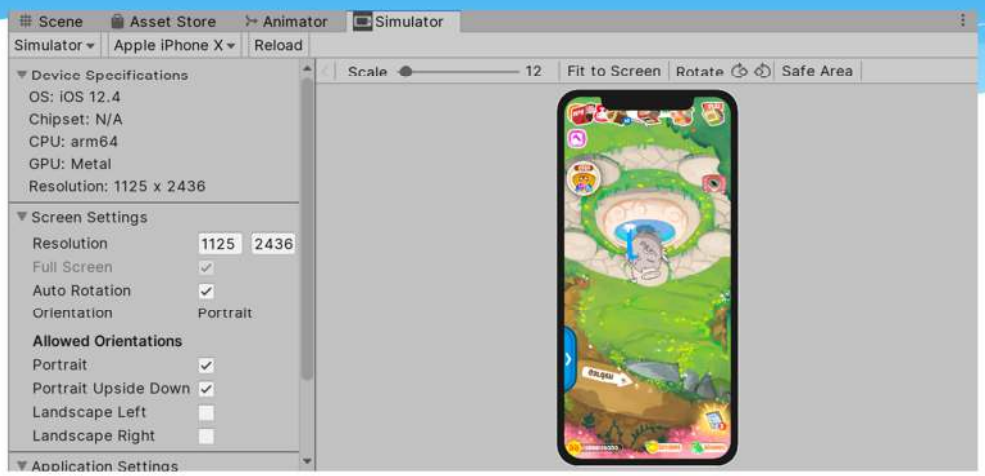
5

Unity Profiler



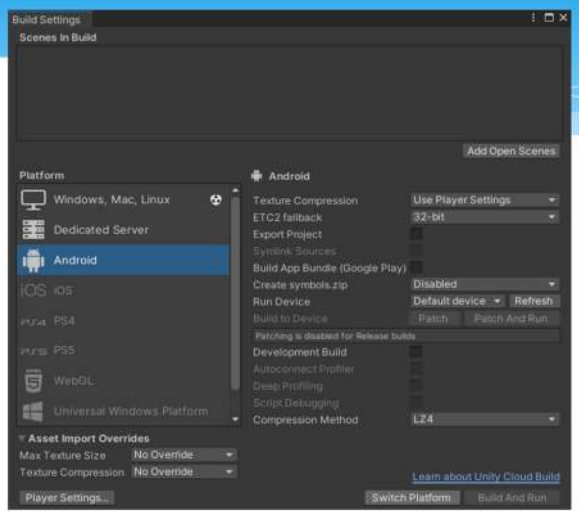
7

Моделювання сценаріїв використання



7

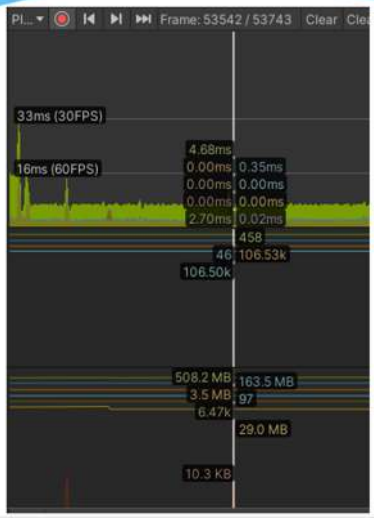
Unity Build Settings як інструмент портування ігор



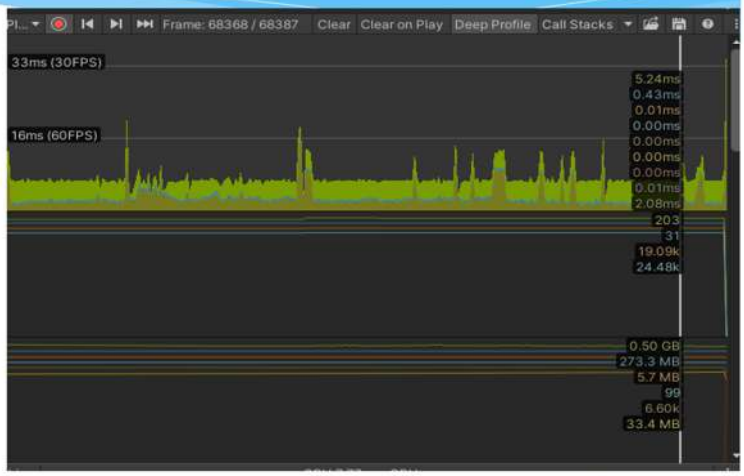
9

Результат роботи удосконалених методів

Результат на момент початку гри за малої кількості об'єктів(Time: 4.68ms)



Результат на момент кінця гри за великої кількості об'єктів(Time: 5.24ms)



9

Експериментальна частина

10

Тестування

Характеристика	Рекомендовані вимоги	Мінімальні вимоги
Операційна система	Windows 10, 11, Android	Windows 10, 11, Android 5.0.2+
Процесор	Intel Core i5-12400H або AMD Ryzen 5 5600H	MediaTek Helio X10або Core i3-3240 або AMD Athlon II X3 4150
Оперативна пам'ять	16 ГБ	3 ГБ
Відеокарта	NVIDIA GeForce RTX 3050ti або AMD Radeon RX 6600	PowerVR G6200 або GTX 1030 або RX 460
Звук	Звукова карта, сумісна з DirectX 9.0c	Звукова карта, сумісна з DirectX 9.0c

Результат тестуванні гри на мінімальних вимогах

Висновки

- Роглянуто існуючі рішення для портування ігор.
- Досліджено вплив портування на продуктивність гри на платформі. Методи портування були спрямовані на зменшення впливу на ресурсоємність гри з метою підтримки гри на конкретних платформах.
- Реалізовано та удосконалено метод Dependency Injection у програмному застосунку та удосконалено за допомогою Zenject.
- Проведено експериментальні дослідження. Результати тестування застосування удосконаленого методу портування на прикладі ігрового додатку на платформі Unity підтвердили що продуктивність і комфорт гри зберігається при її перенесенні на іншу платформу.

Апробація результатів кваліфікаційної роботи

Маматов А.О. Методи і алгоритми портування ігор. *Modernization of science and its influence on global processes: collection of scientific papers «SCIENTIA» with Proceedings of the IV International Scientific and Theoretical Conference, November 3, 2023*. Bern, Swiss Confederation: International Center of Scientific Research. 2023, P. 121–126.

13

ДОДАТОК Б

November 3, 2023 • Bern, Swiss Confederation • Collection of scientific papers «SCIENTIA»

Маматов Артем Олексійович
здобувач вищої освіти технічного факультету
Харківський національний університет радіоелектроніки, Україна

Науковий керівник: Фесенко Тетяна Григорівна
професор кафедри електронних обчислювальних машин, доктор технічних наук,
Харківський національний університет радіоелектроніки, Україна

МЕТОДИ ТА АЛГОРИТМИ ПОРТУВАННЯ ІГОР

Анотація. *Предметом дослідження у статті є методи та алгоритми портування ігор для різних платформ. Мета роботи – вивчення відомих технологій портування ігор та їх особливостей застосування.*
У ході аналізу існуючих рішень зроблено висновок, що це складний та відповідальний процес, який вимагає глибокого розуміння різних платформ та ефективних стратегій адаптації ігор для різних платформ.

Вступ

Портування ігор — це процес адаптації відеоігор для роботи на різних платформах або операційних системах. Ця задача може виявитися викликливою через різницю у архітектурі, апаратних можливостях та особливостях програмного забезпечення різних пристроїв. Для успішного портування ігор використовуються різні методи і алгоритми.

Основною метою статті є розбір та опис технології портування ігор. Стаття оповідатиме більше про умови, в які поставлені розробники, і найуніверсальніші практики, які використовуються в індустрії протягом десятиліть.

Виклад основного матеріалу

1. Аналіз особливостей платформи

Першим і найважливішим кроком у процесі портування є аналіз характеристик цільової платформи. Важливо враховувати апаратні параметри, специфічні функції та обмеження платформи. Цей етап дозволяє розробникам належним чином врахувати всі нюанси та визначити стратегію портування.

Щоб зробити портування відеогри, потрібно:

– Переписати код мовою, прийнятною для нової платформи. Гра повинна працювати і почуватися добре в новому середовищі. Якщо рушій не підтримує платформу, на яку ви переносите гру, розробники переписують тисячі файлів під графічні драйвери для ПК NVIDIA або AMD замість унікальних API, розроблених для консолей та в інших випадках портування. Якщо рушій підтримує нову платформу, він інтегрує нові API та бібліотеки, призначені для цієї платформи.

– Робота над графікою. Графіка повинна виглядати однаково на всіх платформах. (Насправді вона часто не однакова, але команда портування повинна зробити так, щоб вона виглядала саме так).

– Прагніть до гідної продуктивності. Хороша продуктивність гри означає високу частоту кадрів за секунду, якісну підтримку графіки та плавність роботи гравця - на ПК, або тільки останнє - для консолей і мобільних платформ.

– Адаптуйте елементи керування та інтерфейс для нової платформи. Гра має виглядати "рідною" для цільової платформи. Отже, весь код гри, включно з елементами керування та графікою, має бути налаштований так, щоб він добре працював і відповідав

процесору/графічному процесору нового пристрою, його роздільній здатності та користувацькому інтерфейсу.

2. Використання кросплатформених інструментів

Для полегшення завдання портування ігор використовують кросплатформені інструменти. Такі інструменти дозволяють розробникам створювати ігри, які можна легко переносити між різними платформами. Unity, Unreal Engine, та інші ігральні двигуни надають можливість створення ігор, які легко адаптуються під різні операційні системи та пристрої.

Перенесення на Unreal Engine. Він використовує C++ і підтримує розробку консольних платформ, ПК та мобільних ігор.

Перенесення на Unity, який використовує C# і підходить для розробки як 2D, так і 3D ігор на всіх існуючих платформах: веб, ПК, мобільні, консолі та AR/VR.

Портування на HTML5 та JavaScript (веб). Коли студії хочуть опублікувати веб-версію своєї гри, вони портують її - найчастіше з C++ або C# на JavaScript або HTML5.

Портування на UE та Unity зазвичай відбувається, коли гра була розроблена за допомогою мобільного SDK/інших ігрових рушіїв/для вебу дуже давно, або якщо SDK більше не підтримується, або якщо власник хоче отримати доступ до можливостей монетизації/мультиплатформенності, які надають ці рушії.

Якщо гра не була написана на мультиплатформенному рушію (наприклад: Unity 3D, Unreal Engine), то перенесення гри з ПК на консолі або навпаки дорівнює її переписуванню. Те саме з портуванням комп'ютерної гри на Android чи iOS.

Переписування можна спростити за допомогою транскрипторів - програм, які перетворюють код з однієї мови на іншу - але це все одно велика робота.

Якщо у вас стара гра або гра, яка була написана на спеціальному рушію з великою кількістю програмістської акробатики, часто неможливо розібратися у вихідному коді та інструментах, які використовувала попередня команда. У такому випадку жодне програмне забезпечення для портування гри не допоможе вам перенести її на іншу платформу. Проект перетвориться на просту розробку гри: повноцінну роботу за інструкціями.

У цю сферу входять і різні інструменти для арт-відділу (Blender, Houdini, Substance Painter), і фізичні підсистеми (PhysX, Havok, Box2D), і графічні технології (рейтрейсінг, постпроцесинг, тесселяція), і різноманітні бібліотеки для мережевих взаємодій, збирання аналітики, пристроїв управління, звукового супроводу, зберігання внутрішньоігрових даних, тощо.

І, скажемо чесно, ці інструменти покращують життя більше гравцям, ніж розробникам. Ці технології хоч і подають себе як "продукт, готовий до використання", на ділі це каша з великої кількості різних не дуже стабільних розробок, які намагаються однаково задовольнити потреби якомога більшої кількості клієнтів.

У цих технологіях часто достатньо багів, неактуальне API, а вимагають вони до себе максимально дбайливого ставлення, щоб ті не поламалися в критичний момент, якщо ви зберетеся зробити щось нестандартне. Під них доводиться спеціально підлаштовувати гру, а більшість можливостей із коробки ніколи не використовуються і залишаються мертвим вантажем на потужностях. Для їхнього ефективного використання потрібно підвищувати свою кваліфікацію. А ще інструменти можуть застаріти, через що ви більше не зможете здійснювати їхню підтримку, оскільки не ви - розробник технології.

Тому, звісно, чи є користь використання їх у своїй грі - завжди відкрите питання. Можливо, зусилля з підтримки переважують потенційну користь.

3. Оптимізація

Оптимізація відіграє важливу роль у багатьох сучасних іграх, і вона може суттєво відрізнятися на різних платформах. Вона є ключовим аспектом портування. Використання

різних рівнів деталізації, адаптація текстур і роздільної здатності дозволяють досягти оптимального співвідношення якості та продуктивності.

Ігри – це, зокрема, програми, які постійно проводять обчислення. Обчислення ці можуть бути різними: фізика, графіка, ШІ, звук, генерація локацій та інше.

За бажання, звісно, можна всю гру генерувати в реальному часі, але загалом що комплексніші задуми авторів, то більше доводиться витрачати на них процесорних потужностей, і не всім гравцям ці потужності доступні. У деяких (не у всіх) випадках можна зберегти обчислення заздалегідь (це називається "запекти" або "bake") і використовувати їх пізніше. Наприклад, запекти освітлення в текстури, зберегти результати зіткнень і пошкоджень в анімаційних кліпах, скласти моделі поведінки, зробити пререндер катсцен та інше. Запікання забезпечить максимальну якість за мінімальних вимог до потужностей.

Рис. 1. **Процес запікання тіней у текстуру, щоб не прораховувати їх у реальному часі**

Проблема в тому, що чим більше ресурсів ви збираєтеся зберігати підготовленими, то більше вам потрібно пам'яті і то більші вимоги до пропускної здатності пам'яті. Можливо, щось буде швидше обчислювати, ніж шукати для цього вільне місце в пам'яті та завантажувати. Та й сам процес запікання вимагає додаткових дій від розробника.

Як не дивно, якість коду не завжди корелює з його продуктивністю. Зрозуміло, чому: людина і комп'ютер несумісні, і те, що буде оптимальним для комп'ютера, не буде таким читабельним для людини.

Для початку, цікава кореляція: що швидші програми, написані мовою, то складніше нею користуватися. Python може вивчити будь-хто, C++ досі залишається однією з найскладніших для розуміння серед C-подібних мов, водночас за деякими тестами різниця в продуктивності між ними різниться у 80 разів на користь другої.

Досить універсальний код, який можна прочитати, теж пишеться не завжди з першого разу, а крім цього, у програмістів йде час на його підтримку, який вони могли б витратити на реалізацію фіч. Наприклад, дуже багато часу займає розгортання архітектури MVC, яка потім потенційно може принести користь (а може і не принести) за необхідності заміни тих чи інших компонентів програми.

В іграх використовуються різні види ассетів, як-от текстури, моделі, звуки, анімації, і що менші вони за об'ємом, то швидше комп'ютер їх опрацьовуватиме (насамперед тут мається на увазі швидкість завантаження ассетів у пам'ять і на ігровий рівень).

Схитрувати через програмний код тут особливо не вийде: контроль ресурсів повністю на стороні рушія і заліза. Тому крім як зменшувати обсяг ассетів і багаторазово їх перевикористовувати, виходу особливого і немає.

Створювати самі асети не дуже важко, особливо зараз: є дуже багато зручних інструментів створення асетів для будь-яких завдань, навіть безкоштовних. Однак ці інструменти дуже марнотратні. На прикладі 3D-моделей, легко зліпити модель за допомогою скульптора й автоматично згенерувати для неї текстурну UV-розгортку, але в ній будуть тисячі зайвих точок, що мало впливають на якість відображення, а відсоток простору, який використовується розгорткою, буде приблизно 40-60%, що потребуватиме використання текстур вищої роздільної здатності.

Процес оптимізації ассетів полягає в дослідженні ассетів на предмет неефективно займаного місця та усунення цих місць. Десь це усунення дублікатів за допомогою створення одного універсального асета або генератора, десь це стиснення або видалення того, що точно ніхто не побачить поблизу, десь це об'єднання різних сутностей в одну універсальну.

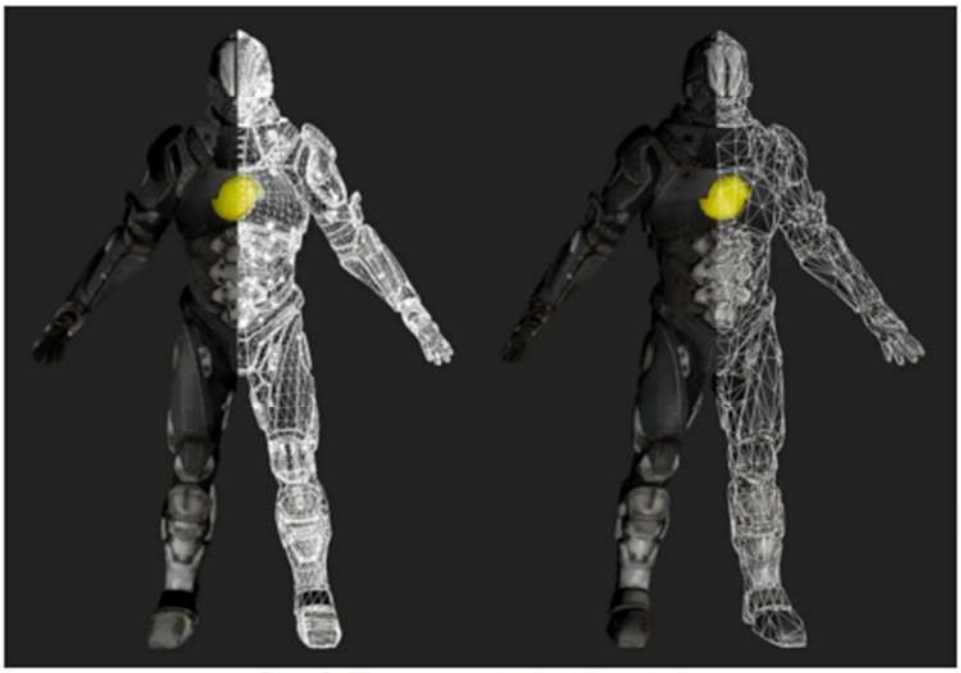


Рис. 2. **Процес оптимізації ассетів**

У прикладі (рис.2) кількість полігонів у моделі зменшилася в 15 разів, а візуально різниця майже не помітна.

Сучасні ігрові рушії дбайливо поділені на незалежні один від одного компоненти, як-от фізика, графіка, менеджмент об'єктів, відтворення звуку, допоміжні алгоритми для інтелектуальних агентів та інше. Код цих компонентів часто вже оптимізовані донезмоги і містять усередині себе величезну кількість налаштувань, що впливають на їхнє використання, зокрема на витрату потужностей. Робота ігрового програміста – правильно налаштувати ці компоненти.

Наприклад, якщо у нас тисячі фізичних об'єктів, можна поділити їх на групи (шари), об'єкти в яких будуть взаємодіяти тільки з сусідами по групі. Якщо у нас тисячі об'єктів з однаковими моделлю і матеріалами, можна увімкнути для них батчинг, щоб ті не перезавантажувалися в пам'яті під час кожного нового відтворення. Якщо у нас багато однотипних об'єктів, що перестворюються (наприклад, снаряди), можна їх усі створити раз заздалегідь і перевикористовувати, а не навантажувати систему постійними створенням і видаленням об'єктів у реальному часі.

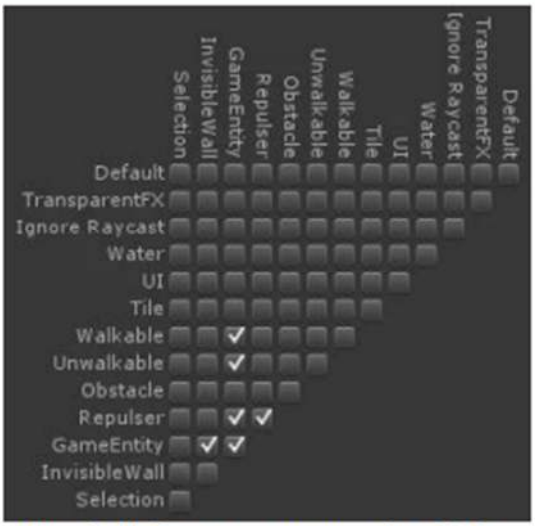


Рис. 3. **Таблиця компонентів в Unity**

Наприклад, в Unity є ось така табличка, що задає взаємодії між фізичними об'єктами з різних груп[1]. Інструменти для всього цього вже доступні, писати майже нічого не треба. У тих випадках, коли потрібно створювати нові ігрові підсистеми, вже вступає в справу повноцінний код – і ось тут варто бути обережнішим. Завдання програміста тут полягає в пошуку багаторазово повторюваних з одним результатом обчислень. Або результати цих обчислень кешуються, якщо вони корисні, або ці обчислення усуваються, якщо їхній результат марний.

4. Обробка різниці у введенні та контролі

Різниця у системах введення та контролі може бути проблемою при портуванні ігор. Важливо адаптувати способи управління грою до характеристик цільової платформи[2].

Також слід враховувати відмінності в елементах керування на різних платформах. Це питання є найбільш важливим при портуванні на ПК та мобільні пристрої або з них. Комп'ютерні ігри управляються за допомогою миші та клавіатури, які мають вищий рівень точності, ніж геймпади. Водночас, елементи керування на сенсорних екранах повинні бути розроблені таким чином, щоб не заважати ігровому процесу.

Зіставлення клавіш клавіатури з входами контролера не завжди є вдалою ідеєю. Тому на пошук хорошого рішення йде більше часу.

Консольні SDK мають багато зручних функцій, які може бути важко перенести на ПК. Наприклад, вони можуть надавати доступ до апаратних таймерів або хорошого аудіо API. Ці функції зазвичай недоступні на ПК. Тому вам доведеться використовувати інші методи, щоб досягти цих цілей.

5. Адаптація елементів гри

Припустимо, ваша гра була розроблена для ПК. Деякі елементи її геймплею обов'язково покажуть це. Наприклад, вони можуть вимагати використання миші та точного керування нею. Ці елементи повинні бути адаптовані для покращення користувацького досвіду на різних платформах.

6. Створення відповідного інтерфейсу та схеми управління

На цьому етапі процесу портування гри важливо переконатися, що всі гравці отримують однаково плавний і приємний досвід.

Цей крок є особливо складним, коли мова йде про портування гри на мобільну платформу або з неї, оскільки управління на сенсорному екрані є дуже специфічним. Воно повинно надавати всі необхідні функції, мати зручний користувацький інтерфейс і не заважати ігровому процесу.

7. Запуск численних тестів

Після завершення портування важливо провести ретельне тестування на різних пристроях. Це дозволяє виявити та усунути можливі проблеми, пов'язані з апаратними особливостями та конфігураціями пристроїв.

Не намагайтеся скоротити час реалізації проекту, відмовляючись від забезпечення якості. Тестування – один з найважливіших етапів процесу перенесення гри; перенесена гра може відрізнятися від оригіналу, що може призвести до несподіваних взаємодій, помилок і лагів[3]. Обов'язково проведіть якомога більше тестів, щоб переконатися, що у вашій грі немає багів, і ви зможете випустити якісний продукт, який не розчарує користувачів.

Висновки

Портування ігор — це складний та відповідальний процес, який вимагає глибокого розуміння різних платформ та ефективних стратегій адаптації. З використанням вищезазначених методів і алгоритмів розробники можуть створити ігри, які успішно функціонують на різних пристроях, незалежно від їх технічних характеристик.

Список використаних джерел:

1. Unity Documentation: [Інтернет-портал]. URL: https://docs.unity3d.com/Manual/index.html (дата звернення: 23.10.2023).
2. Комп'ютерна гра: [Інтернет-портал]. URL: https://en.wikipedia.org/wiki/PC_game (дата звернення: 23.10.2023).
3. Hocking J. Unity in Action. Multiplatform game development in C# with Unity 5. Shelter Island, NY: Manning Publications Co, 2015. 352 c.
4. Ferrone H. Learning C# by Developing Games with Unity 2020. Shelter Island, NY: Packt, 2020. 366 c.