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 ГРИ "SZWARGOT" У ЖАНРІ ПЛАТФОРМЕР НА ДВИГУНІ UNITY»

Виконав: студентка 4 курсу, групи ПД–41

спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності/спеціалізації)

Леньо В. Я.

(прізвище та ініціали)

Керівник Дібрівний О.А.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного
забезпечення
_____Негоденко О.В.
"____" _____2022 року

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

ЛЕНЬО ВОЛОДИМИРУ ЯРОСЛАВОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема роботи: «розробка гри "Szwargot" у жанрі платформер на двигуні Unity»

Керівник роботи: Дібрівний Олесь Андрійович, доктор філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «16» лютого 2022 року №22.

2. Строк подання студентом роботи «1» червня 2022 року

3. Вхідні дані до роботи

3.1 Методи розробки відеоігрового програмного продукту;
3.2 Науково-технічна література з питань, пов'язаних з програмним забезпеченням щодо розробки відеоігор;
3.3 Офіційна документація Unity
3.4 Офіційна документація Microsoft Visual Studio
3.5 Офіційна документація мови програмування C#

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).
4.1Аналіз актуальності та проблематики розроблюваної гри
4.2 Аналіз та вибір інструментів для розробки даної гри
4.3 Опис проектування гри
4.4 Висновки
5. Перелік демонстраційного матеріалу (назва основних слайдів)
5.1 Титульний слайд
5.2 Індивідуальне завдання
5.3 Аналіз та загальна характеристика ігор-платформерів
5.4 Мета, об'єкт та предмет дослідження
5.5 Аналіз існуючих рішень
5.6 Технічні завдання
5.7 Висновки
5.8 Кінцевий слайд

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	09.04.2022	Виконано
2	Дослідження аналогів та актуальності додатку	15.04.2022	Виконано
3	Аналіз та вибір інструментів для розробки додатку	17.04.2022	Виконано
4	Проектування та реалізація	03.05.2022	Виконано
5	Вступ, висновки, реферат	16.05.2022	Виконано
6	Розробка обов‘язкових демонстраційних матеріалів	17.05.2022	Виконано
7	Попередній захист роботи		
8	Здача роботи	01.06.2022	

Студент _____ Леньо В. Я.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Дібрівний О.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи с., 4 рис., 42 джерела.

Ключові слова: відеогра, платформер, Unity, сюжет, механіка, ігровий двигун.

Об’єкт дослідження – гра жанру платформер.

Предмет дослідження – метод розробки комп'ютерної гри жанру платформер.

Мета роботи – розробка розважальної комп'ютерної гри жанру платформер з допомогою ігрового рушія Unity.

Наукова новизна – перетворення існуючих ігрових механік жанру платформер для створення цікавої та легкою в розумінні гри, у яку гравець буде хотіти витрачати час.

У дипломній роботі проаналізовано ринок комп’ютерних ігор жанру платформер, знайдено переваги та недоліки інструметів для розробки.

Комп’ютерна гра була створена на рушії Unity, код був написаний в інтегрованому середовищі розробки Visual Studio 2019 мовою програмування C#.

Ця гра може бути використана як спосіб ненадовго відволіктися та витратити час на нескладний процес.

ЗМІСТ

ВСТУП

В сучасному світі кожного дня людину оточує велика маса різноманітного програмного забезпечення, починаючи від серйозних технічних рішень і закінчуючи соціальними мережами і іграми.

Із підвищенням доступності різного роду девайсів, таких як смартфони, персональні комп’ютери, ігрові приставки, тощо, рівень зацікавленості населення в іграх росте шокуючими темпами.

Виходячи з такої великої актуальності теми ігор саме її було обрано як тему даної роботи, а саме розроблення інді-платформеру.

Для виконання поставленої задачі слід виконати наступні завдання:

- Провести огляд предметної області
    - Визначити, що таке розробка відеоігор
    - Визначити поняття платформера
    - Оглянути процес створення відеоігор
- Обрати засоби реалізації програмного забезпечення
    - Обрати рушій
    - Обрати мову програмування
    - Обрати середовище розробки
- Спроектувати внутрішню будову
- Розробити графічний інтерфейс
- Розробити основні механіки
- Провести тестування

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття відеогри

Відеогра або комп'ютерна гра – гра на електронному пристрої, головними елементами якої є керування користувацьким інтерфейсом або спеціальним пристроєм: джойстиком, контроллером, комп'ютерною мишею, клавіатурою, тощо – та візуальне відображення дій. Дії візуально відображаються потім на різних відео пристроях, наприклад: монітор, екран, дисплей, пристрій віртуальної реальності. Також часто існує аудіовідклик, котрий передається на динаміки або навушники, а також інші види відклику, як наприклад тактильний, наприклад вібрація. Проте комп'ютерні відеоігри складають лише частину усіх відеоігор, бо деякі не залежать від пристрою виведення відео.

Відеоігри виходять на платформи, котрі їх класифікують. Це може бути аркада, консолі (PlayStation, Xbox), персональні комп'ютери (ПК). Недавно на ринку з'явились мобільні пристрої, такі як смартфони та планшети, а також системи AR (доповненої) та VR (віртуальної) реальностей, ігри у хмарі. Відеоігри також діляться на жанри в залежності від типу взаємодій, що доступні гравцю.

Прабатьки ігор з'явилися у 1950-х та 1960-х роках та є простими налаштуваннями існуючих електронних пристроїв виводу зображення для великогорозмірних комп'ютерів. Першою грою для широкого споживання була аркада Computer Space, що вийшла у 1971 році. Через рік, у 1972, з'явилась одна з найпопулярніших ігор того часу Pong та перша користувацька консоль Odyssey від компанії Magnavox. Далі була золота доба відеоігор, що почалась на кінці 1970-х та протривала до початку 1980-х років, проте через неконтрольоване видавництво та перенасиченість однотипних ігор галузь зазнала краху. Потім ринок дозрів, на ньому з'явилися та почали домінувати

японські компанії Nintendo, Sega, Sony та інші. Також розробилися методи розробки та дистриб'юції відеоігор, котрі актуальні та активно використовуються і на сьогодення – це було заради запобігання схожого краху в майбутньому. Наразі процесс створення успішної відеоігри потребує вміння щоб розробити та вивести її на ринок, етапи включають у себе розробників, видавництва, магазини, комп'ютерних та консольних інженерів та професіоналів інших спеціальностей [13].

У першому десятилітті 21-го століття основний фокус індустрії виявився на AAA-проектах, тобто великих іграх від великих розробників та видавництв, тому експериментувати стало ризиковано та неприбутково. Проте з другого десятиліття та з розвитком інтернет-технологій незалежно розроблювати та видавати ігри стало набагато легче, почали з'являтися інді-студії та їх проекти, що і стало популярним напрямком індустрії відеоігор у 2010-х роках. З тих пір ігри стали доступніші, тому вага та вартість цієї індустрії почала зростати. На азіатському ринку великої популярності зазнають ігри на мобільних пристроях: смартфонах та планшетах, а також портативних консолях, наприклад, PlayStation Portable (PSP). Демографія також змінюється у сторону більшої інклюзивності, ігри стають цікаві все більш молодим поколінням. Також з'являються сервісні пропозиції ігор, так звані "Games as a Service" (GaaS): замість витрати єдиної суми для купівлі гри, тепер грати дозволяється після придбання помісячної, або іншої, підписки. Такий зріст дозволив світовій індустрії ігор мати дохід у 159 мільярдів доларів США станом на 2020 рік, що є втричі більше, ніж прибуток музичної індустрії роком раніше і в чотири рази більше, ніж прибуток індустрії кіно також роком раніше.

Перші відеоігри використовували різні формати виведення відео інформації. До найдавнішого прикладу можна віднести "пристрій для розваг із електронно-променевою трубкою" 25 січня 1947-го року: патент США 2455992 Томаса Т. Голдсмата-молодшого та Естла Рей Манна, котрий зрештою було видано 14 грудня 1948-го року. Натхнення для цього пристрою бралося з технології радіолокації, а тому він складається з аналового пристрою

для керування параболічною дугою точки на екрані, таким чином імітуючи постріл ракети по цілях, котрі у свою чергу є закріпленими на екрані паперовими малюнками. Іншими прикладами ранніх прототипів ігор можуть бути: "Чернетки" від Крістофера Стрейчі; комп'ютер Nimrod, представлений на Британському фестивалі у 1951 році; OXO; хрестики-нолики на комп'ютері EDSAC Олександра С. Дугласа 1952 року; інтерактивна для для двох Вільяма Хігінботама Tennis fro Two 1958 року; а також Spacewar!, від студентів Массачусетського технологічного інституту (MIT) Мартіна Греца, Стіва Рассела та Вейна Вітанена, котру часто називають творцем жанру платформерів. Кожна з перечислених ігор має власні засоби відображення: NIMROD використовує панель підсвічування для гри в Нім; OXO використовує графічний дисплей для виведення та гри у хрестики-нолики; Tennis for Two використовує осцилограф, що відображає тенісний корт, а Spacewar! – дисплей на основі векторів DEC PDP-1 для симуляції зіткнення космічних кораблів. [13]

Ці та інші прототипи вимостили доріжку для прабатьків сучасних відеоігор. Таким чином, у 1966 році у Sanders Associates, Ральф Х. Баєр за допомогою телевізійного екрану розробив систему для гри у настільний теніс. Потім Баєр запатентував цей винахід та створив "Brown Box", котрий винахід потім теж був запатентований та ліцензований компанії Magnavox для створення першої домашньої консолі Magnavox Odyssey, котра опинилася на ринку у 1972 році. Ця платформа дала можливість Нолану Бушнеллу та Теду Дабні, котрі надихнулись Spacewar!, у 1971 році створити подібну платформу, але меншу за розміром та що використовувала дешевший комп'ютер, котра мала у собі Computer Space – першу аркадну відеогру. Пізніше Бушнелл та Дабні створили компанію Atari Inc. і разом із Аланом Алкорном у 1972 випустили другу аркадну гру пінг-понг, котра брала своє натхнення від схожої гри у теніс на консолі Odyssey. Sanders та Magnavox потім подали до суду на Atari за порушення патентів Бера, проте справа була вирішена поза судом виплатою грошей за безстрокові права на патенти. Це дозволило Atari у 1975

році створити Pong – ще одну домашню консоль, котра теж виявилась успішною, як і Odyssey, що дало старт індустрії відеоігор, а Баєр та Бушнелл були названі "батьками відеоігор".

Термінологія

Термін "відеоігри" був створений для розрізнення цього типу розваг від тих, що використовували не відеодисплей для відображення візуальних елементів, а, наприклад, принтер або подібні інші пристрої. Це також відрізнило цей тип ігор від, наприклад, схожих до Merlin, котрі використовували лампи для індикації стану, проте не комбінували їх для створення зображення.

Термін "комп'ютерна гра" можна також використовувати для опису відеоігор, адже всі відеоігри використовують комп'ютерний процесор, проте цей термін може бути також для більш специфічних ігор, котрі запускаються на персональних комп'ютерах, на відміну від тих, котрі працюють тільки на консольних системах. Інші терміни: "телевізійна гра" або "телегра" – також використовувалися у 1970-80-х роках для опису ігор, що запускалися на консолях, котрі підключалися до телевізорів. У Японії такі ігри відомі калькою з англійської "terebi geemu", адеж перші консолі, як Odyssey, та ігри до них імпортувалися з заходу, а тільки потім почали виготовлятися компаніями Toshiba, Sharp Corporation та іншими. Також для характеризування відеоігор можна використовувати термін "електронна гра", проте це також має на увазі ранні портативні електронні ігри, котрі розроблялись без відеовиходів. Термін "телевізійна гра" наразі також широко використовують.

«Відеогра» як термін з’явився десь у 1973 році. Стаття BusinessWeek від 10 листопада 1973 року була процитована Оксфордським словником англійської мови як перше використання цього терміну у друці, проте сам "батько відеоігор" Бушнелл вважав, що цей термін бере свій початок із журналу Computer Space 1971 року. Аналіз основних журналів Vending Times і Cashbox виявив, що цей термін з'явився у масовому вживанні, у тому числі виробниками аркадних ігор, у березні 1973 року. За аналізом відеоігрового

історика Кіта Сміта такій нагальній популярності термін вдячний теплим сприйняттям тими, хто задіяних у сфері відеоігор. За теорією, до цього причетний Ед Адлум, котрий до 1972 року керував відділом монетних розваг у Cashbox, а у 1975 році заснував журнал RePlay Magazine, котрий писав про галузь монетних розваг. Саме у 1982 році, у випуску цього журналу за вересень, Еду приписують перше вживання терміну "відеоігри": "Едді Адлум з Replay працював у "Cashbox", у той час "телевізійні" ігри тільки починали виходити. Тоді Бушнеллу, також його менеджеру з продаж Пет Карнс та ще кільком іншим розробникам тих ігор, Генрі Лейзеру та братам Мак'юен у тому числі, не сподобався термін "телевізійні ігри", тому вони запозичили назву з опису музичних автоматів з фільмами Billboard – "відеоігри". Ця назва була охоче сприйнята та стала популярною." Адлум пояснив, що до початку 1970-х років у розважальних центрах найчастіше були аркадні ігри без відео виходів: пінбол, електромеханічні ігри. З появою відеоігор виникла плутанина з приводу того, який термін потрібно уживати для точного опису нової продукції.

Платформа

Відеоігри запускаються на платформах. Платформа то певний набір електрокомпонентів або комп'ютерного обладнання та програмного забезпечення, написаного для нього. Термін система також є широко уживаним. Зазвичай ігри розроблюються для певної платформи, або для обмеженої їх кількості, а потім портуються, себто переписуються для того, аби працювати на інших. Проте іноді навмисне не пишуться версії для інших систем, а ексклюзивність використовується для переваги над конкурентами на ринку. Також процес портування може відбутися набагато пізніше випуску самої гри, коли перевага над конкурентами вже не потрібна, а продажі потрібно підвищити. Також для збільшення кількості продажей існує опція розробки ремастеру: коли більшість коду старої гри все ще використовується, проте змінюється візуальна та аудіоскладова, виправляються деякі баги, гра стає доступною для новіших платформ – або ж ремейку: написання повністю

нового коду, створення нової графіки та звукового оформлення за сюжетом старої гри.

Комп'ютерна гра

Більшість комп'ютерних ігор є саме іграми на комп'ютер, себто ті, що взаємодіють з комп'ютером (ПК), котрий підключений для пристрою виведення відео: монітор, телевізор. Персональні комп'ютери, проте, не є суто для ігор, а також мають велику варіативність у деталях, особливо порівнюючи з консолями, що може призводити до відмінностей, а також помилок між іграми, що на них запускаються. Окрім того, ПК це відкрита платформа, що дозволяє розробникам експерементувати над компонентами та різними їх комбінаціями, змінювати програмне забезпечення, використовувати більш дешеві його версії, емулювати ігри, модифікувати їх та інше. Окрім того вже існує можливість хмарного геймінгу, тобто коли користувач заходить на сервер, котрий проводить усі обчислення на своїй стороні, а через інтернет передає тільки картинку, проте це потребує хорошої якості з'єднання та великої швидкості, окрім того обов'язковим елементом такого досвіду є затримки.

Домашня консоль

Домашню консоль можна назвати більш стандартизованим варіантом ПК із зазвичай закритою до модифікацій платформою. Підключається до телевізора або монітора. Консолі розроблені спеціально для відеоігор, а тому розробникам набагато легче оптимізувати ігри під конкретний набір електронних та програмних компонентів, що дозволяє уніфікувати ігровий досвід та запобігти появі небажаних помилок. Зазвичай на консолях запускаються ігри, котрі розроблені саме під цю консоль, або під продукт від тієї самої компанії, але ніколи не не ті, що розроблені для пристроїв конкурентів. Саме це поляризує ринок консолей та обмежує поле для експериментів, яке існує у випадку з ПК. Найпопулярнішими консолями на даний момент є PlayStation та Xbox.

Портативна консоль

Портативна ігрова консоль це така компактна консоль, котра має контроллери, дисплей та аудіосистему вже вбудовані у своє тіло, а також батарею, та дозволяє грати у неї будь-де. Як правило портативні консолі менш потужні, ніж стаціонарні консолі або ПК. У 1990-х та 2000-х роках багато портативних консолей використовували картриджі, що дозволяло грати у більш ніж одну гру на цій платформі. Проте у 2010-х роках портативна консоль стала нерелевантною, адже з'явилися смартфони та ігри на них, і це виявилося більш комфортним варіантом для гравців.

Аркадна відеогра

Аркадна відеогра це така гра, у котру грають на максимально спеціалізованому типі пристрою, котрий, зазвичай, призначений для однієї цієї гри та розміщений у спеціальному просторі – ігровій шафі, котра має дисплей, динамік і контролери, а також часом інші потрібні для гри речі. Окрім того, часто такі ігри зібрані у спеціальних аркадних центрах і для того, аби їх запустити, використовуються монети. Зазвичай аркадні ігри мають яскраве оформлення, що приваблює погляд, та стосується теми гри. Тоді як більшість аркадних ігор використовують вертикальну шафу, деякі розміщуються у столі, коли екран розміщується у горизонтальній шафі, тобто столі, із прозорим верхом. У настільні ігри зазвичай грають сидячи. У десятих роках 21 століття аркадні ігри далеко не такі популярні, котрими вони було у 90-х та 2000-х роках, проте іноді в кінотеатрах та розважальних центрах можна знайти спеціально відведені під них зали.

Браузерна гра

Гра у браузері використовує переваги спільного середовища для гри, тобто спільного інтерпретатора коду, такого як Blink у Google Chrome, для створення уніфікованого ігрового досвіду. Такі ігри характеризуються за сайтами, на яких вони існують, наприклад, ігри від Miniclip, або ж за допомогою технологій їх розробки: Java, Flash.

Мобільна гра

Смартфони швидко стали популярною платформою для ігор, а мобільні ігри зайняли значний процент ринку, особливо після появи стандартизованих платформ iOS та Android. Ці ігри можуть отримувати переваги тих комплектуючих смартфонів, котрі інші платформи необов'язково мають, як наприклад акселерометр, геопозиція (GPS) або камера для підтримки технології доповненої реальності (AR).

Хмарні ігри

Хмарні ігри не потребують від гравця потужного комп'ютера, підійде будь-що: не потужний комп'ютер, консоль, ноутбук, мобільний телефон – будь-який пристрій, котрий має підключення до дисплею та вихід до інтернету, а також спеціальне програмне забезпечення постачальника сервісу хмарних ігор. Усі потрібні для гри обчислення відбуваються на сервері постачальника послуг, у той час як клієнт отримує тільки згенеровану картинку, що дуже комфортно для володарів не потужних пристроїв. Один із мінусів – затримка даних між введенням гравця та виведенням інформації на його екран, проте вона вирішується за допомогою методів прогнозування. Також потрібен швидкий інтернет та безперебійний доступ до мережі. Прикладами є: Xbox Cloud Gaming, Playstation Now, що використовують спеціальні блейд-сервери.

Віртуальна реальність

Ігри у віртуальній реальності (VR) – порівняльно новий вид ігор, котрі потребують від гравця використовувати спеціальний прилад – окуляри, що реагують на рухи голови гравця та дозволяють бінокулярно бачити 3D-об'єкти – та маніпулятори для максимального занурення в ігровий світ та прямої взаємодії з ним. Системи VR зазвичай потребують окремого блоку для обробки – ПК, консолі, тощо – що поєднується із пристроєм на голові. Наразі найпопулярнішими компаніями-виробниками пристроїв віртуальної реальності є Oculus (належить Meta, у минулому Facebook), Valve, Sony.

Емуляція

Емулятор – таке програмне або апаратне забезпечення, котре дозволяє симулювати середу, для запуску якої оптимізована гра. Зазвичай реалізується як віртуальна машина на сучасній системі, що імітує апаратне забезпечення оригіналу та дозволяє грати в старі ігри. Хоча в судовому праві США емулятори є законними, саме програмне забезпечення, котре буде уживатися на них, може підлягати під закон про авторське право. Є деяке емульоване програмне забезпечення від офіційних виробників, як наприклад Virtual Console або Switch online від Nintendo.

Зворотна сумісність

Зворотна сумісність це майже те саме, що емуляція, тому що в ігри зі старших платформ можна грати на новіших. Це досягається за допомогою вбудованого програмного забезпечення для підтримки ігор, розроблених для старших платформ. Наприклад, PlayStation 2 дозволяє запускати ігри для PlayStation просто зчитуючи дані з носія, таким же чином Nintendo Wii може запускати ігри для Nintendo GameCube.

Ігрові медіа

Перші відеоігри-аркади, домашні консолі та портативні ігри були спеціальними платформами з ігрою вбудованою в апаратні компоненти пристрою. Наразі у повальній більшості випадків зустрічаються ігри, котрі розповсюджуються на якихось медіа: CD-диски, DVD-диски, карти флеш-пам'яті, також через інтернет. Консолі розвили технології зчитування з диску, флеш-пам'яті, доступу до інтернету, стали більш уніфікованими, тепер дають змогу мати цілу колекцію ігор та грати у них на вибір навіть не вставляючи носій у консоль. На сьогоднішній день усе частіше фізичні носії служать для встановлення гри, тому що вони занадто повільні, аби використовувати їх для постійного зчитування інформації, тому гра записується на внутрішню пам'ять пристрою, а пізніше оновлюється з неї. Також розвинута практика купувати ігри без носіїв – тоді потрібен тільки доступ до інтернету.

Ігри розширюються за допомогою нового вмісту, пакетів, розширень – програмного забезпечення, котре розроблюються та йде окремо від ігри. Вони

пропонуються або безкоштовно, або можуть бути використані для монетизації гри після її виходу на ринок. Деякі ігри надають можливість створювати цифровий контент власноручи та ділитися ним із іншими гравцями. Інші ігри, зазвичай на ПК, можуть отримати додатковий функціонал за допомогою створених гравцями модифікацій або модифікацій, котрі змінюють або доповнюють гру, та часто є неофіційними і розробленими за допомогою реверс-інжинірингу гри, але інші компанії надають можливість офіційно модифікувати свої ігри.

Пристрій введення

Відеоігри використовують декільки видів пристроїв для фіксації дій користувача та переведення їх у поведінку в ігровому світі. Найбільш поширеними є такі ігрові контролери, як геймпад, джойстик – це основні пристрої керування для консолей. Комп'ютери використовуються найчастіше з клавіатурою та мишкою, також мають можливість підключення вищевказаних контролерів. Елементи керування в найновіших контролерах включають у себе кнопки для визначення напрямку зору, тригери на боках, аналогові джойстики та перехрестя управління. Консолі зазвичай продаються із стандартними контролерами, котрі входять у комплект із самою консолею, у той час як інша периферія доступна для купівлі окремо від виробника консолі або від сторонніх виробників. Подібні засоби керування вбудовуються в портативні консолі та аркадні шафи. Новіші контролери мають у собі вбудовані додаткові технології, такі як дисплей, сенсорний екран, датчики виявлення руху, тощо, котрі надають більше можливостей для взаємодії із ігровим світом. Також для певних жанрів ігор існують спеціалізовані контролери, як наприклад руль та педалі, танцювальні майданчики, ігрові пістолети, тощо. Також цифрові камери та детектор руху можуть зчитувати рухи тіла гравця і переносити їх до ігрового світу, таким чином створюючи або доповнену, або віртуальну реальність, що сприяє кращому зануренню у гру.

Відображення та вихід

Як закладено у самому терміні, кожна відеогра призначена для виведення графіки на зовнішній дисплей, такі як, наприклад, телевізори на основі електронно-променевих трубок, новіші телевізори з рідкокристалічним дисплеєм (LCD), вбудовані екрани, проектори або комп'ютерні монітори зі схожими технологіями відображення та залежно від типу платформи, на якій грають. Такі функції, як глибина кольору, частота оновлення, частота кадрів, роздільна здатність екрану обмежуються як самою грою, так і платформою, на якій грають, куди до першої входять технології відображення, кількість операцій на секунду, кількість полігонів у графіці, а до останньої пристрій відображення та апаратне забезпечення. Відображення гри варіюється в залежності від обраного дисплею, технологій, на котрих він побудований, а також виду графіки: 2D або 3D – що використовує гра, та технологій, що задіяні у грі. [6] [5]

Графіку часто комплементує відповідний до програмного забезпечення звук, котрий подається на внутрішні динаміки платформи або на підключені зовнішні. Це найчастіше звукове оформленння дій користувача для отримання зворотного зв'язку, подій у ігровому світі, також фонова музика, або музика під якусь подію, тощо.

Також деякі платформи та їх контролери підтримують додаткові технології зворотного зв'язку з гравцем, котрі гра може використати. Це найчастіше тактильний відгук вбудованих у геймпади, коли контролер вібрує в руках гравця для імітації подій у грі, котрі торкаються гравця.

Класифікація.

Жанр

Відеоігри можна охарактеризувати розділивши на жанри, таким чином як й інші медіа. Однак на відміну від інших медіа, таких як фільми, музика та серіали, котрі використовують візуальні, сюжетні або музикальні засоби, відеоігри категоризуються за жанрами в залежності від їх взаємодії з ігровим процесом, оскільки це найголовніше, що гравець робить із ігрою. Сюжет не впливає на процес, адже платформер залишається платформером незалежно

від того, це фентезі чи фантастика. Єдиним виключенням можна назвати жанр ігор-жахів, котрі використовують прийому жахів, як надприродні явища, фантастику або психологічний тиск.

Назви жанрів зазвичай є одночасно містким описом ігрового процесу, наприклад екшн, платформер, шутер, рольова гра (RPG), тощо, хоча деякі жанри беруть свою назву від відомих та упливових творів, котрі визначили напрям, наприклад roguelike бере свою назву від гри Rogue, gta-клони від серії ігор Grand Theft Auto, Battle Royale ігри від фільму Battle Royale. Імена також можуть змінюватись з плином часу, наприклад, коли гравці, ЗМІ або розробники придумують нові терміни: таким чином шутери перше називались "Doom-like", або ж "клонами Doom" через їх схожість із цією впливовою грою, випущеною у 1993 році. Також жанри можна ієрархізувати, поділивши їх на жанри вищого рівня: "шутер", "екшн" – які мають широке значення та описують стиль гри загалом, а також кілька піджанрів для кожного, котрі описують конкретну реалізацію: шутер від першої особи, шутер від третьої особи. Також деякі ігри складно віднести до якогось одного жанру, тому є міжжанрові типи, як наприклад пригодницький екшн. [4]

Режим

Режим відеогри означає кількість гравців, котрі можуть використовувати гру одночасно. Відеоігри можна поділити на для одного гравця та для багатьох гравців. Також останню категорію можна розділити на підкатегорії: ті, у котрі можна грати на одному комп'ютері; ті, котрі потребують LAN-мережі та ті, котрі аспект багатьох гравців реалізують за допомогою мережі інтернет. Більшість відеоігор для кількох гравців містять у собі елемент змагання, також багато які з них командні або кооперативні, мають асиметричний ігровий процес. Онлайн-ігри використовують серверне апаратне забезпечення для організації ігрового процесу для декількох гравців одночасно, також ігри, котрі підтримують гру для сотень гравців одночасно класифікуються як MMO.

Нерозповсюдженими є ігри з умовною нульовою кількістю гравців, у яких гравець обмежено взаємодіє із самим ігровим світом. Зазвичай це симулятори, де можна виставити початковий стан симуляції та залишити процес на час, пасивно спостерігаючи за результатами обчислень. Так існує, наприклад, гра Життя, вигадана Джоном Конвеєм у 1970-у році.

Намір

Більшість відеогір мають розважальні цілі, також ще називаються "основні ігри". Існує також підмножина ігор, котрі розроблені для інших окрім розваг цілей. До таких ігор належать:

Казуальні ігри

До ігор, котрі створенні для щоденного ужитку, легко отримати доступ, вони мають простий для розуміння ігровий процес, простий набір правил, а також націлені на масовість, на відміну від хардкорних ігор, котрі випробовують уміння гравця. Часто такі ігри мають можливість відновлення ігрового процесу після виходу гравця зі гри, наприклад, під час перерви від роботи або очікування в черзі. Найчастіше веб-ігри та ігри для мобільних пристроїв відносяться до казуальних, адже не потребують інтенсивного мислення. Прикладами можуть бути ігри типу три-в-ряд, знаходження прихованих об'єктів, клікер, тобто де гравцю потрібно багато раз клікати по якимсь предметам, тощо. Казуальні ігри розповсюджені у соціальних мережах, часто використовують їх можливості, такі як мережа друзів, розповсюдження дописів, тощо, таким чином наділяючи гравця можливістю робити додаткові ходи, прискорювати події, отримувати внутрішньоігрові гроші, тощо. Популярними казуальними іграми є, наприклад, Tetris і Candy Crush Soda Saga.

Розвиваючі ігри

Також ігри можуть використовуватися в освітніх цілях для навчання дітей та студентів, адже вони інтерактивні та приносять розвагу. Розвиваючі відеоігри можна умовно поділити на дві групи. Перші зосереджуються на розвагах та зубрінні, але не вимагають критичного мислення. Інші мають на

меті змотивувати до вирішення проблеми та закріпити матеріал, нехтуючи розважальною цінністю відеогри. Прикладом може бути The Oregon Trail та серія Carmen Sandiego. Окрім того деякі елементи відеоігор роблять їх корисними для використання у процессі науки, хоча перше ці ігри створювали без думки про навчання. Прикладом може бути Minecraft через його відкритий світ, а також SpaceChem за її елементи головоломки.

Серйозні ігри

Серйозні ігри – такі, де елемент розваги не грає ролі, адже гра служить іншим цілям. Навчальні ігри є однією з форм серйозних ігор, проте саме їх поняття ширше та ця категорія може включати в себе, наприклад, фітнес-ігри, де гравцю потрібно виконувати значні фізичні вправи для підтримки форми, як у Wii Fit, симулятори польотів, як Microsoft Flight Simulator, рекламні ігри для надання популярності продукту, як Pepsiman, або новинні ігри, котрі пропагують певне послання, як NarcoGuerra.

Художня гра

Окрім того, що відеоігри самі по собі є творами мистецтва, вони можуть використовуватися як засіб донесення якоїсь історії або задуму. Ці ігри також можуть називатися артхаусними та створені для того, аби викликати у гравця емоційну реакцію. Також ці ігри можуть не мати звичних елементів відеоігрів інших жанрів, як наприклад кінечної цілі, а бути створеними суто для цілі дослідження гравцем ігрового світу та його сценарію. Зазвичай художні ігри це також інді-ігри, котрі беруть за основу персональний досвід та створені однією людиною або невеликою групою розробників. Прикладами можуть бути "That Dragon, Cancer", Flower, Passage.

1.2 Поняття платформеру

Гра-платформер, або рідко jump'n'run, – це відеоігровий жанр, що є піджанром екшн-ігор, у котрих основна метра це переміщення ігрового

персонажу, котрим керує гравець, по платформам, або іншим подібним об'єктам, у 2D або 3D просторі. Платформи – рельєф різної висоти, схили, підвісні блоки, тощо. Таке розташування об'єктів у світі примушує гравця використовувати стрибки, ривки та інші маневри, включаючи спеціальні здібності, для пересування світом та боротьби з ворогами, а також збирання спеціальних об'єктів, тощо. Такі ігри, де для стрибків не потрібно нічого робити, платформерами не рахуються, прикладом може бути 3D-гра The Legend of Zelda. [4] [7]

Хоча про платформери склався стереотип ніби то зазвичай консольні ігри, було створено багато відомих назв та навіть серій відеоігор для портативних консолей та ПК.

Коли був пік популярності ігор жанру платформер наприкінці 1980-х років та на початку 90-х, платформери складали від 25% до 30% усіх створюваних ігор для консолей, проте потім поступово були витіснені шутерами від першої особи. У 2006 році ігри цього жанру знизили свою долю ринку настільки, що становили усього 2%, у порівнянні з 15% наприкінці дев'яностих. Проте жанр до сих пір існує на ринку та розвивається, а кількість продажей рахується мільйонами. [13]

Поняття

Гра платформер вимагає від гравця точного контролю свого персонажу для маневрування між перешкодами: платформами, стінами, ігровими об'єктами, а також ворогами – для подолання дистанції та досягнення цілей. Ці ігри найчастіше мають вид збоку у двовимірному або тривимірному просторі, або ж у 3D, коли камера знаходиться позату персонажу або в ізометричному зображенні. Зазвичай ігровий процес платформерів динамічний і потребує від гравця хороший рефлексів та швидкої реакції, а також точності у володінні елементами керування.

Найрозповсюдженішими рухами у цьому жанрі є ходьба, біг, рух, стрибки, ривки, атака, лазіння, а також подвійні стрибки. Стрибки це найголовніший тип рухів у жанрі. Деякі ігри мають фіксовану параболу

польоту, деякі дозволяють гравцю маніпулювати траєкторією під час стрибку, таким чином або різко обриваючи його, або змінюючи напрям, або подовжуючи. Багато таких ігор мають перешкоди, котрі при контакті шкодять персонажу гравця та можуть привести до смерті та кінця гри. Це можуть бути шипи, ями, бездонні прірви, тощо. Також часто можна зустріти реалізацію усіляких бонусів, котрі гравець може використати для полегшення ігрового процесу. Це можуть бути зілля для відновлення здоров'я, різні покращення для швидкості, стрибучості, атаки, тощо, а також монети, котрі є частиною внутрішньоігрової економіки.

Більша частина ігр жанру платформер мають зростаючу складність, котра змінюється з кожним рівнем, таким чином утримуючи гравця зацікавленим та випробовуючи його уміння, одночасно з тим постійно вводяться нові ігрові механіки та відбувається процес навчання, що утримує увагу людини. Зазвичай порядок рівнів визначений, проте деякі ігри цього жанру можуть мати сюжетні гілки, коли гравець може вирішити яким саме шляхом потрібно зараз піти далі. Також у іграх-платформерах можна зустріти головоломки – ще один поширений елемент жанру. [11]

Сучасніший варіант ігор-платформерів називають бігунами. Це такий підтип жанру, коли персонаж гравця постійно рухається уперед, не зупиняючись, а людина може контролювати коли стрибати, йти наліво або направо маршруту, а також збирати поліпшення та монети на шляху. Зазвичай гра закінчується як тільки гравець б'ється о перешкоду. Найпопулярніший цей жанр на мобільних пристроях. [14]

Іменування

Різні назви жанру почали з'являтися після випуску у світ гри Donkey Kong у 1981 році. Сігеру Міямото ще до випуску називав цю гру "jump'n'run", а пізніше той же автор уже використовував термін "спортивні ігри".

Donkey Kong створив собою напрям ігор з бігом, стрибками та вертикального переміщення, котрого раніше не було, тому журналісти почали вигадувати та пропонувати власні назви для нового жанру. Серед варіантів

були "ігри типу Donkey Kong" від Computer and Video Games журналу, або "Kong-style". "Ігри про лазіння", котрими так їх назвав Стів Блум у своїй книзі "Video Invaders", випущеній у 1982 році, та в деяких інших журналах. Блум описав такі ігри, де гравець повинен підніматися знизу догори, а також уникати, або руйнувати перешкоди та ворогів на шляху. Таким терміном він назвав Space Panic та Frogger, а також Donkey Kong.

У 1982 році рецензент у Creative Computing назвав гру Beer Run жанру драбин в одному з оглядів.

Інший термін, котрий використовували для опису подібних ігор – ігри про персонажів, а більш широко таким терміном називали ігри про Маріо, Соніка, тощо, тобто де головним героєм був оригінальний персонаж, котрий виділявся.

Платформер став стандартним терміном наприкінці 80-х років минулого століття, а саме через популярність у пресі Великобританії. Британські журнали використовували термін "платформер" для опису ігор, котрі схожі на Mario. [13]

1.3 Розробка відеоігор

Розоробка відеоігор – процес створення відеогри від моменту ідеї до моменту створення працюючої копії та виправлення багів. Цей процес може виконуватись як одним розробником, так і командою людей, навіть із різних країн, далеких одна від одної. Традиційно розробку ігор для ПК та консолей зазвичай фінансує видавець, а строки завершення можуть вимірюватися парою років. Інді-ігри потребують менше часу та грошей на реалізацю і можуть бути повністю створеними однією людиною, або невеличкою інді-студією. Таке явище як інді-ігри виявилося на підйомі своєї популярності завдяки зростанню рівня технологій та програмного забезпечення створення ігор як рушії Unity та Unreal, окрім того поява нових середовищ розповсюдження ігор, котрі мали

менші вимоги до пропрацьованості, як наприклад популяризація мобільних пристроїв Android та iOS, а також ріст популярності платформ розповсюдження ігор як Origin, Steam, тощо. [1]

Перші ігри та їх прототипи почали з'являтися у 60-х роках зовсім не для комерціалізації, а для забезпечення роботи великих комп'ютерів – мейнфреймів – і були доступні тільки персоналу, котрий працював із ними, а не широкому загалу. Випускатися на ринок для купівлі відеоігри почали декадою пізніше в 70-х роках, коли почали з'являтися перші ігрові консолі та ранні ПК домашнього користування. На той час такі системи не володіли великими потужностями, а тому поодинокі ентузіасти самотужки могли розробити повноцінну гру для пристрою. Про у 80-90-х роках потужності компутацій та рівень технологій зростали, гравці ставали більш вибагливими у своїх очікуваннях, а тому створити гру для ПК або консолі самотужки перестало бути реальністю. Вартість розробки triple-A проекту, або ж проекту великого розміру, виросла з одного мільйона доларів у середньому у 2000 до 5 мільйонів доларів США у 2006, а потім до 20 мільйонів доларів станом на перший рік нової декади.

Процес розробки комп'ютерних та консольних ігор зазвичай має декілька етапів: на першому етапі, котрий називається попереднім виробництвом, відбувається опис ідеї, створення прототимів, документу дизайну гри, презентація; якщо абстракція схвалена і є фінансування, починається другим етап, тобто власне сама розробка. Зазвичай розробка нової великої гри типу AAA це спільні зусилля десятків людей, котрі відповідають за різні аспекти процесу, як наприклад дизайн рівнів, графіка, код, організація процесу, тощо. [15]

Огляд

Ігри це також процес розробки програмного продукту. Вони створюються з творчою основою у думці та з прибутком на меті. Створення ігор це і мистецтво, і наука. Розробку зазвичай фінансує видавець, адже цікава та видовищна гра приносить прибутки, однак також важливо оцінювати куди

витратити фінанси та людський час, аби достатньо проробити усі потрібні аспекти продукту. Тому більшість ігр, що продаються, насправді не приносять ніяких прибутків.

Індустрія відеоігор постійно потребує іноваційних рішень та свіжих ідей, адже неможливо отримувати прибуток від виробництва сіквелів та імітацій. І краще великих студій із цим справляються маленькі студії, котрих багато створюється щороку та яким вдається розробляти успішні ігри. Але таким же чином багато студій закриваються, адже не можуть знайти фінансування, а вже розроблені ігри прибутків не приносять. Також процес монетизації ігр останнім часом спростився за рахунок популярності мобільних ігрових платформ. Тоді вже в залежності від популярності розроблених продуктів та прибутків команди можуть розширятися та створювати більш зрілі та більші за розміром ігри. [15]

Бюджет, котрий витрачається на розробку ігор для багатьох платформ, зазвичай складає біля 25 мільйонів доларів США в середньому, а ААА-проекти оцінюються в більш ніж 40 мільйонів доларів США.

На початку появи та популярності консолей та ПК у 80-х роках одному розробнику було під силу провести гру крізь усі стадії розробки від ідеї до дистриб'юції, а також займатись усіма аспектами самої гри, як дизайн, код, графіка, звук, тощо. Розробити гру можна було усього за 6 тижнів у середньому. Наразі високі очікування гравців потребують відповідних по рівню продуктів на ринку. Це значить, що ігри можуть розроблюватися декілька років сотнею розробників, котрі працюють повний робочий день.

Розробка відеогри – процес, що чинається з ідеї, або ж конепту гри. Дуже часто це просто модифікація вже існуючої концепції. Ідею можна охарактеризувати одним або декількома жанрами одночасно, окрім того ігрові дизайнер часто експериментують з мультижанровістю. Зазвичай дизайнер створює документ початкових нарисів гри, де описується ідея, ігровий процес, список функціональних вимог, а також такі вимоги як цільова аудиторія,

вимоги, графік, оцінку персоналу та потрібних фінансів. У різних компаній процедури та філосовії дизайну різні, але існують спільні риси.

Розробником гри може бути як одна людина, якщо це інді-проєкт, так і ціла велика студія, якщо розробляється ААА-гра. Існують як незалежні студії, так і видавничі. Незалежні студії отримують фінансування від видавця, а тому етап розробки гри від ідеї до першої версії не фінансується. Тоді вноситься пропозиція видавцям і ставляться строки фінансування розробки. Видавець за собою зберігає права на видавництво гри, а також права інтелектуальної власності на продовження серії. Також видавничі компанії можуть або володіти окремою компанією розробки, або мати внутрішню команду для цих потреб.

Усі компанії-розробники, за винятком маленьких, у котрих немає багато людських ресурсів, зазвичай працюють над декількома проектами відеоігор одночасно. Це необхідно для максималізації прибутків, адже іноді час між віправленням гри та отримуванням прибутків може сягати півтори роки.

Для випуску гри на консоль вона повинна відповідати технічним вимогам, котрі встановлюють виробники цієї консолі, аби бути схваленою, наприклад Microsoft для XBox або Sony для PlayStation. Окрім того концепцію потрібно обговорювати з виробником консолей, котрий може внести свої корективи.

Розробка більшості відеоігр для ПК та консолей займає від трьох до п'яти років. На тривалість розробки впливає цілий ряд фактів, як жанр, розмір гри, список технологій для використання, кількість грошей, розмір команди, тощо.

Розробка деяких ігор, тим не менш, займає довше часу, ніж потрібно. Так, наприклад, Duke Nukem Forever розроблювалась 14 років, з 97-го року аж до 2011, Spore – дев'ять, Prey – вісім, Team Fortress 2 – дев'ять. За словами головного розробника Гейба Ньюела, насправді за цей час було випущено три чи чотири різні гри.

Дохід від відеогри розповсюджується нерівномірно між усіма ланками ланцюга розробки: студією або командою розробників, видавцем, магазинами, роялті виробникам консолі, тощо. Багато студій розробки не отримують належної кількості грошей та банкрутують. Інша опція це пошук альтернативних шляхів монетизації проекту, як наприклад через вбудовані покупки, рекламу, інтернет-можливості, підписку, тощо.

Процес розробки

Розробка відеоігор – процес створення відеогри, а також процес творчості, адже гра сама по собі це і мистецька складова, і технологічна. Ігри з погано організованим процесом розробки перевищать потрібні витрати передбачені бюджетом, займуть більше часу, ніж потрібно, а також будуть складатися з багатьох багів. Тому планування важливе, особливо для великих проектів.

Процес розробки відеоігор не підходить під загальну модель життєвого циклу розробки та існування програмного забезпечення, як модель водоспаду, наприклад.

Дуже популярним методом є agile development, котрий має у основі ітерації та прототипи на кожній із них. Agile полягає у зворотному зв’язку та уточненнях наприкінці кожної ітерації, поступово збільшуючи функціонал. І цей метод є ефективним тому, що з початку розробки відомо не всі вимоги, а також вони можуть змінюватись під час цього процесу. Найпопулярнішим agile-методом є Scrum.

Іншим успішним методом є процес персонального програмного забезпечення (PSP), який вимагає додаткового навчання персоналу для підвищення обізнаності щодо планування проекту. Цей метод є більш дорогим і вимагає прихильності членів команди. PSP можна поширити на Team Software Process, де вся команда сама керує. Цей процес має на меті тренування персоналу для передбачення роботи коду та спостереження за його реальним шляхом відтворення. Такий процес показує та навчає розробників як керувати їх власними продуктами, як створювати правильний план розробки

та як відслідковувати передбачену та наявну поведінку коду в проекті, а також пропонує дані для підкріплення планування. Розробники потім можуть оцінити свою працю та побачити шляхи покращення через аналіз часу розробки, дефектів та інформації про розмір виконаної роботи.

Ця методологія передбачає високу якість програмного продукту, а якість вимірюється кількістю дефектів. З точки зору PSP, якісний процес розробки програмного забезпечення це такий, котрий дає мінімум дефектів, при цьому відповідаючи вимогам користувача.

Розробка ігор зазвичай передбачає накладання цих методів. Наприклад, створення активів може здійснюватися за допомогою моделі водоспаду, оскільки вимоги та специфікації ясні, але дизайн ігрового процесу може бути виконаний за допомогою ітераційного прототипування.

PSP та Agile схожі за деякими аспектами, а саме: визначення цілей та стандартів, оцінка витрат часу на задачі та створення графіку праці, визначення реалістичного та комфортного графіку, створення покращень у планах та процесах.

Також PSP та Agile мають спільну ідею, що кожен член команди бере відповідальність за свою частину роботи, а також працює з іншими разом для узгодження реалістичного плану, створення атмосфери довіри та відповідальності.

2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Огляд ігрового рушія

Unity – мультиплатформний ігровий рушій, тебто той, що працює та створює ігри для багатьох платформ, котрий розробила компанія Unity Technologies. Уперше його показали у червні 2005 року на AWDC, котру проводила компанія Apple, та він задумувався як рушій ексклюзивно для платформи Mac OS X. Проте із плином часу його функціонал став доступним на 25 платформах, включаючи найпопулярніші, такі як Windows та Linux, а також стало можливим створювати ігри ще на iOS та Android, а також для веб-середовища. Рушій можна використовувати для створення як 3D, так і 2D ігор, ігор VR та AR, а окрім того для створення 3D-роликів, моделювання 3D-об'єктів для машинобудування, будівництва будівель, проектування архітектури, тощо, а також для симуляції фізики. [1] [12]

Ціллю створення цього рушія було дати доступний інструмент для створення відеоігор, котрий міг би зрозуміти та використовувати будь-який розробник-початківець. Наступного року рушію було призначено винагороду "Найкраще використання графіки Mac OS X" на церемоції нагородження від Apple.

Наступна версія двигуна 2.0 була випущена 2007-го року та могла виконувати біля 50 різних функцій. У тій версії оптимізували 3D обробник, що дозволило створювати більш деталізовані 3D-середовища, оптимізовано завантажувати спрямоване світло та тіні у реальному часі. Також ця версія містила нову організацію асинхронного коду, тобто такого, котрий виконується одночасно, що дозволило підвищити кількість операцій на секунду, котрі виконуються одночасно.

Unity 3.0 був наступним великим оновленням, котре знову торкалось обробки 3D-простору. Тут було покращення відображення карт тіней,

відкладеного завантаження, з'явилась можливість легше створювати та редагувати дерева, а також покращена робота з UV mapping, тобто організації 3D-простору на 2D-зображенні.

Станом на 2012 рік понад мільйона розробників уживали Unity щоденно. Також опитування, проведене у тому же році журналом Game Developer, визначило Unity як найпопулярніший ігровий рушій для створення ігор на мобільні платформи. Восени того же року було випущено четверту версію двигуна, у котрій з'явилася підтримка DirectX 11 та Flash, а також передрелізна для Linux.

У 2013 році компанія Facebook убудувала у свій сайт підтримку цього рушія. Це дозволяло слідкувати за рекламними кампаніями, аналізувати певні взаємодії гравців з іграми та діями в них. Також у 2016 році Facebook випустив нову платформу для створення ігор на ПК, рушійною силою котрої є Unity. Це спростило та пришвидшило процес публікації ігор на Facebook.

У 2015 році The Verge назвав Unity "кроком до майбутнього" за його мету зробити розробку ігор доступним процесом. У тому же році була випущена п'ята версія двигуна, котра дозволила публікувати ігри для Web, адже з'явилась підтримка WebGL, що дозволило створювати ігри без потреби попередньо встановлювати доповнення для браузеру. Також були покращення в освітленні та обробці аудіо, наприклад: глобальне освітлення в реальному часі, попередній перегляд результату освітлення, Unity Cloud для оптимізації постачання проектів на хмарне сховище, а також перехід на новий рушій фізики PhysX версії 3.3 від Nvidia. Також підверсія 5.6 додала підтримку платформи Switch, підтримку API Vulkan, програвання 4K відео та створення 360 огляду для пристроїв VR. Також Unity зіштовхнувся із критикою, адже завдяки ньому з'являється багато низької якості ігор. Проте генеральний директор зявив, що це є тільки побічним ефектом популярності рушія.

З 2016 року версії почали писатись інакше, тому наступною версією Unity вже була Unity 2017. У цій версії було додано опрацювання графіки в реальному часі, а також інструменти звіту про ефективність. Також у цій версії

Unity Technologies вказали на ще один вектор розвитку рушія, а саме кінематографічний. Тоді було додано таймлайн, що спрощувало створення анімацій, а також вбудовано розширення Cinemachine для простішої роботи з камерами. [2]

У 2018 році Unity представили налаштування відображення графіки в залежності від платформи, а також інструменти машинного навчання.

У 2019 році рушій навчився використовувати мову Wolfram, що дозволило обчислювати складну математику.

У 2020 році було представлено MARS: набір для створення ігор у просторі доповненої реальності.

Наразі стабільною версією є 2021.3 LTS, випущена 19 травня 2022 року. LTS – це абревіатура, що означає "Long Time Support", тобто з довгим періодом підтримки у 2 роки.

Unity дозволяє розробникам створювати як 2D, так і 3D ігри та анімацію, а також використовувати C# для написання коду для створення логіки. Наразі C# є єдиною мовою програмування, що підтримується Unity. До того була ще Boo, підтримку котрої припинили у версії Unity 5, та UnityScript, котру перестали підтримувати у Unity 2017.1, повністю перейшовши на C#. А також існують плагіни для створення логіки без написання коду, а за допомогою візуального скриптингу. [3]

2D можливості Unity дозволяють використовувати спрайти, для 3D існують технології зменшення розміру текстур, карти текстур, налаштування роздільної здатності в залежності від платформи, зображення вибухів, відбиття від поверхонь, ефекту паралаксу, SSAO, динамічних тіней, постпроцесинг – також цей перечень доступний для 2D ігор.

За статистикою 2018-го року на Unity створено приблизно половина ігр для мобільних пристроїв та більше половини ігор VR та AR, на деяких платформах сягаючи 90%. Також Unity підтримує машинне навчання та бібліотеку Google TensorFlow. Також програмне забезпечення Unity використовується для побудови автопілотів.

До застарілих та відкинутих технологій Unity можна віднести розширення до веб-браузера для запуску Web-ігор – його витіснив WebGL завдяки коду, котрий компілюється у JavaScript у дві стадії: з C# на C++, а потім уже на JavaSript.

Також Unity та Nintendo мають об'єднання та постачають набір для розробки програмного забезпечення (SDK) для розробки відеоігор на Wii U у комплекті із кожною ліцензією розробника для Wii U.

2.2 Огляд мови програмування

C# (вимовляється сі-шарп) – мова програмування загального ужитку, котра підтримує багато парадігм, використовує строгу типізацію, тобто коли змінна не може змінювати свій тип під час виконання програми, область видимості, а також такі парадигми програмування, як функціональна, імперативна та об'єктно-орієнтована. Ця мова програмування була створена у 2000 році Андерсом Хейлсбергом в рамках проекту .NET на Windows. Mono – це безкоштовний проект, що має відкритий код та котрий орієнтувався на перенесення проекту .NET на інші платформи за допомогою підтримки CLI. C# є однією з мов програмування, створених з підтримкою CLI.

Остання версія мови програмування C# на даний момент – C# 10, котра була випущена для .NET 6.0.

Спочатку бібліотеки для .NET фреймворку були написані на системі компіляторів для керованого коду SMC (Simple Managed C). Історія C# розпочинається з 1999 року, коли Андерс зібрав команду аби розробити COOL "C-like Object Oriented Language", котрий був перейменований у C# на конференції Professional Developer Conference у 2000 році, також тоді цією мовою переписали бібліотеки ASP.NET. Хейлсберг – найголовніший архітектор у Microsoft, до розробки C# займався TurboPascal, Visual J++. Під час інтерв'ю з ним та у своїх технічних паперах він указав на проблеми, котрі

існують у наявних тоді мовах програмування: C++, Java, Smalltalk – та котрі призвели до створення CLR та C#.

Джеймс Гослінг, творець Java, нарекли C# "імітацією Java", проте потім додали, що "це Java, де немає надійності, продуктивності та безпеки". Також Клаус Крефт та Анжеліка сказали, що Java та C# схожі до ідентичності мови програмування та що ті ніяким чином не змінили те, як усі пишуть код, а також що C# позичають у один одного особливості. Проте думка Хейлсберга така, що C# більш схожий до C++ за дизайном.

Проте після випуску C# 2.0 у листопаді 2005 року шляхи C# та Java розійшлися у двох зовсім різних напрямах, таким чином ці мови програмування стали зовсім різними. Найбільшу різницю можна виділити у використанні дженеріків, або узагальненого програмування, адже дві мови мають сильно реалізації, що сильно відрізняються. C# використовує реіфікацію, або ж використовує дженерік клас як будь-який інший клас, а генерація коду відбувається під час завантаження цього класу до процесу. Також C# розширився за допомогою особливостей функціональної парадигми, таким чином створюючи LINQ, а також додаючи лямбда-вирази, або ж анонімні функції – себто котрі не мають назви, проте мають параметри та вихідний тип; методи-розширення, себто методи, котрі додаються до об'єкту певного типу після того, як його скомпільовано; а також анонімні типи, тобто такі об'єкти, для котрих не потрібно попередньо створювати визначення класу. Ці особливості дозволяють використовувати такі техніки парадигми функціонального програмування, як замикання, котре дозволяє виконувати підпрограму під час виконання основного коду. LINQ же у свою чергу зменшує кількість повторення розповсюдженого коду, як наприклад логічних операцій, або вираховування середнього, максимального, або мінімального значень з масиву, тощо.

Дизайн C# був створений таким чином, аби повністю відображати CLI, котрий лежить у його основі. Майже усі його внутрішні типи відповідають тим типам, котрі існують в CLI. Проте опис мови не має чіткого обмеження щодо

коду, знегерованого компілятором, а значить дозволяє, теоретично, генерувати машинний код з коду C#, як це є у Fortran або C++.

C# підтримує оголошення строго типізованих змінних за допомогою ключового слова var, а також масивів за допомогою ключового слова new, дозволяючи компілятору самому визначити тип змінних.

C# містить підтримку строгого булевого типу даних, таким чином можна використовувати тільки вирази, котрі реалізують оператор true. Порівнюючи із C++, де булевий тип може вільно конвертуватися між типом цілих чисел, C# не дозволяє випадків, де число може значити true або false, таким чином схиляючи програмістів до використання суто булевого типу, таким чином запобігаючи розповсюджених помилок типу if(name1=name2), де має бути if(name1==name2).

Також C# дає більшу безпеку для конвертації типів, аніж C++. Єдиним неявним перетворенням, тобто котре робиться без написання додаткового коду з боку розробника, котре дозволяється є безпечне перетворення, наприклад конвертація чисел з int до long, адже long містить більший діапазон чисел. Це правило діє при компіляції, у випадку компіляції коли код виконується (JIT) та під час виконання. Також не відбувається перетворень між булевими та числовими типами даних, а також будь-яка небезпечна взаємодія з кодом повинна бути явно визначена, наприклад, за допомогою safe і unsafe блоків.

C# також підтримує явне визначення коваріантних та контраваріантних перетворень типів та їх підтипів у дженеріках, на відміну від C++, де це просто семантичний нюанс типів, що повертають віртуальні методи.

У C# немає такого поняття, як глобальні змінні або глобальні функції, тому що усі методи та члени типів повинні бути вказані всередині класів. Статичні члени публічних класів можуть давати схожий функціонал, при потребі.

Локальні змінні не можуть перекривати змінні блоку вище, як це у C й C++.

C# підтримує строгі вказівники на функції за допомогою ключового слова delegate. Це дає можливість реалізовувати відносини типу publisher-subscriber, коли виклик методу може викликати інші методі з інших класів. Також ключове слово event реалізую шаблон спостерігача, або підписки на подію та виклик методів, коли ця подія (член, визначений event) відбувається.

У C# присутній прибиральник сміття – підпрограмма, котра запускається одночасно із основною програмою, та слідкує за тим, аби пам'ять звільнювалась від об'єктів, котрі в подальшому не будуть використовуватись. Тому пам'ять не може бути звільнена розробником мануально, таким чином прибираючи цю відповідальність зі списку обов'язків розробника.

У C# заборонене множинне наслідування, тобто коли клас має більше одного базового класу, проте дозволяється реалізація класом багатьох інтерфейсів. Це спеціальне рішення з точки зору дизайну аби уникнути ускладнень та спростити вимоги до архітектури CLI. На відміну від C++, де воно присутнє. Окрім того, при реалізації класом двох інтерфейсів, котрі мають методи з однаковою сигнатурою, дається можливість реалізовувати їх одночасно відповідно до інтерфейсу та викликати за потреби той чи інший, або ж створити одну реалізацію та викликати саме її за допомогою будь-якого інтерфейсу.

Також C# підтримує перевантаження операторів, котре відсутнє в Java.

Також тому що LINQ можна використовувати через .NET Framework існує можливість отримати будь який об'єкт колекції, дані XML, набір даних ADO.NET, дані з SQL бази даних через Entity Framework, тощо. Використання LINQ дозволяє інтегрованому середовищу розробки (IDE) розуміти код та давати підказки, окрім того проводити фільтрування даних і маніпулювати ними. [17] [20]

C# була обрана основною мовою розробки даного проекту з огляду на усе вищеописане, а саме — на функціонал, який підтримує дана мова, операційні системи, які підтримуються та простота вивчення.

Загалом — функціонал C# найкраще підходить під дану розробку, за рахунок простої реалізації основних принципів ООП та підтримки Unity.

2.3 Огляд середовища розробки

Microsoft Visual Studio — це IDE від Microsoft, що використовується для створення програм для різних платформ, а також веб-сайтів, сервісів та додатків для веб та мобільних додатків. Visual Studio дозволяє розроблювати ПЗ для Windows, використовуючи такі технології Microsoft, як Windows API, WinForms, WPF, Windows Store, Microsoft Silverlight, тощо. Він може генерувати користувацький код та компілювати його. Visual Studio містить редактор коду, який підтримує розумне сприйняття коду IDE і комфортне його перероблювання – рефакторинг. Налагоджувати можна як сам код, так і скомпільовану його версію. Окрім того існує вбудований аналізатор коду, конструктор графічного інтерфейсу, конструктор веб-інтерфейсу, конструктор класів і конструктор схем для баз даних. У ньому використовуються плагіни, котрі можуть додавати різний функціонал для спрощення або покращення існуючих функцій, як наприклад систем керування версіяими як Git, тощо, додавати нові інструменти, як наприклад підтримка інших мов за потребою, візуальні дизайнери або інші інструменти для різних галузей, а також для розробки ПЗ для постачання (DevOps). IDE розуміє 36 мов програмування і дозволяє редактору коду та інструментам відладки підтримувати будь-яку мову програмування за присутності спеціальних сервісів для мови. Visual Studio Community доступна безкоштовно кожному.

Девіз спільноти Visual Studio – «Безкоштовна повнофункціональна IDE для студентів, відкритих кодів та окремих розробників». Наразі підтримується Visual Studio версії 2022. Початково Visual Studio не має жодного функціоналу пов'язаного з розробкою, але усе підключається за допомогою VSPackage

файлів та через програму-інсталятор. Сама IDE як платформа дозволяє відбуватися комуніції між різними сервісами: SVsSolution для роботи з рішеннями та проектами, SVsUIShell для функцій вікон та інтерфейсу користувача та SVsShell для реєстрації пакетів функціоналу VSPackage. Усі редактори, дизайнери, та інші інструменти постачаються як окремі пакети доповнень VSPackage. Visual Studio уживає Component Object Model для отримання доступу до VSPackage. SDK для Visual Studio також має MPF, яка являє собою набір керованих обгорток навколо COM-інтерфейсів, які дозволяють писати пакунки будь-якою мовою, сумісною з CLI. Однак MPF не надає всіх функцій COM-інтерфейсу Visual Studio. Ці служби потім допомагають та дають освнову для створення додаткових пакетів розширень, які додають новий функціонал для Visual Studio IDE. Підтримка додаткових мов програмування реалізується за допомогою функціональності VSPackage, котра називається Language Service. Цей сервіс визначає різні інтерфейси для реалізації VSPackage розширень для додання різного нового функціоналу. Цей функціонал, наприклад, може змінювати кольори відображення підсвічування синтаксису, завершення написання коду за розробником, перевірка дужок на присутність пари, підказки з інформацією про параметри та методи, списки членів класу, а також відображення помилок на етапі написання коду. Якщо такий інтерфейс реалізується, то функціонал доступний для обраної мови, а послуги надаються в залежності від цієї мови та реалізуються або на рідному коді, або на керованом, окрім того можуть використовувати код із аналізатора синтаксису або компілятора обраної мови. Для нативного коду підійде також COM інтерфейс або фреймворк Babel, котрі є частиною SDK Visual Studio. Для керованого коду також існують обгортки MPF для створення нових модулей сервісу мов. Visual Studio базово не має у собі системи керування кодом, але вона визначає дві альтернативи для інтеграції системи керування кодом із IDE. Управління вихідним кодом VSPackage може надати власний налаштований інтерфейс користувача.

Плагін MSSCCI дає функціонал для реалізації різних аспектів систем контролю версій коду та керування ними через вбудований користувацький інтерфейс IDE Visual Stuido. Спочатку цей плагін був уживаний для інтегрування Visual SourceSafe до VS 6.0, потре потім віз став доступний у SDK. VS 2002 уживає цей плагін версії 1.1, а VS .NET 2003 – 1.2, VS 2005, 2008 і 2010 же – 1.3, додаючи підтримку зміни назви та видалення дистрибуивів та асинхронного відкриття. Visual Studio підтримує декілька варіантів середовища виконання (кожен із власним набором пакетів типу VSPackage). Екземпляри використовують різні записи реєстру для збереження стану конфігурації та розрізнення їх по AppId (Ідентифікатор програми). Примірник запускається спеціальним для AppId.exe, який вибирає ідентифікатор додатку, встановлює кореневу папку і запускає IDE. Пакети VSP, зареєстровані для одного AppId, інтегруються з іншими пакетами VSP для цього AppId. Створюйте різні версії продуктів Visual Studio з різними програмами. Продукти Visual Studio Express інсталюються з власним AppId, але Standard, Professional і Team Suite мають той самий AppId. Таким чином, Express Edition можна встановити разом з іншими випусками, на відміну від оновлення інших випусків тієї ж інсталяції. Професійне видання має VSPackages стандартної версії, а набір команд включає додатковий набір VSPackages у двох інших виданнях. Система AppId використовується оболонкою Visual Studio у Visual Studio 2008. [18] [19]

Це середовище розробки було вибрано на основі кількох критеріїв:

- Вбудована підтримка C#
- Простий та зрозумілий інтерфейсТак як ця IDE підходить по всім критеріям, середовищем розробки було обрано саме її.

3. ПРОЕКТУВАННЯ І РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Розробка діаграми класів продукту

В інженерії програмного забезпечення діаграма UML класів – статична діаграма, що відображає та описує структуру системи, її класи, їх властивості, методи і реляції між різними об'єктами системи.

Діаграми класів це основні будівельні блоки об'єктно-орієнтованого моделювання, котрі використовуються для створення концепції структури проекту та перетворення моделей у програмний код. Діаграми класів Також можуть бути використані для моделювання даних. Класи на такій діаграмі це основні елементи системи, а також взаємодії між ними.

На діаграмі класи представляються вікнами, котрі у свою чергу містять три відділення:

- Верхній відділ з назвою класу. Друкується жирним шрифтом та ставиться по центру, перша літера кожного слова велика.
- Середній відділ з атрибутами класу. Вони вирівнюються за лівим краєм, букви з маленької
- Нижній відділ з методами класу. З лівого краю, букви маленькі.

Під час проектування системи багато класів збираються у схему класів для визначення реляцій між ними. Також класи часто поділяють на підкласи.

Залежність це реляція між залежними елементами схеми та незалежними. Залежність можна утворити між двома елементами, якщо при зміні першого елементу виникають зміни у другому. Залежності схематично показуються пунктиром з незаповненою стрілкою лінією, котра проводиться від замовника до постачальника.

Далі ці діаграми класів можуть бути доповнені діаграмами станів UML.

Асоціація це серія посилань. Бінарні асоціації, тобто між двома об'єктами, часто показуються у вигляді лінії. Асоціація може пов'язувати необмежену кількість класів. Асоціація трьох класів називається потрійною. Асоціаціям можна дати ім'я, а кожен кінець назвати якоюсь роллю, власністю, іншими атрибутами.

Існує 4 типи асоціацій: двонаправлені, з одним напрямком, зворотні та агрегатні. Найпоширенішими є перші два типи.

Агрегація це один із підвидів відношення "має/has". Вони конкретніші за асоціації і представляють собою частину мети або відносини. Агрегатні реляції також можуть мати такі ж елементи, що і асоціації, однак вони обмежуються тільки двома об'єктами одночасно. Окрім того, іноді різницею між агрегацією та асоціацією на схемі нехтують.

В UML асоціації представляються за допомогою ліній, що з'єднують пов'язані класи між собою. Кожна лінія може мати додаткові позначення. [16]

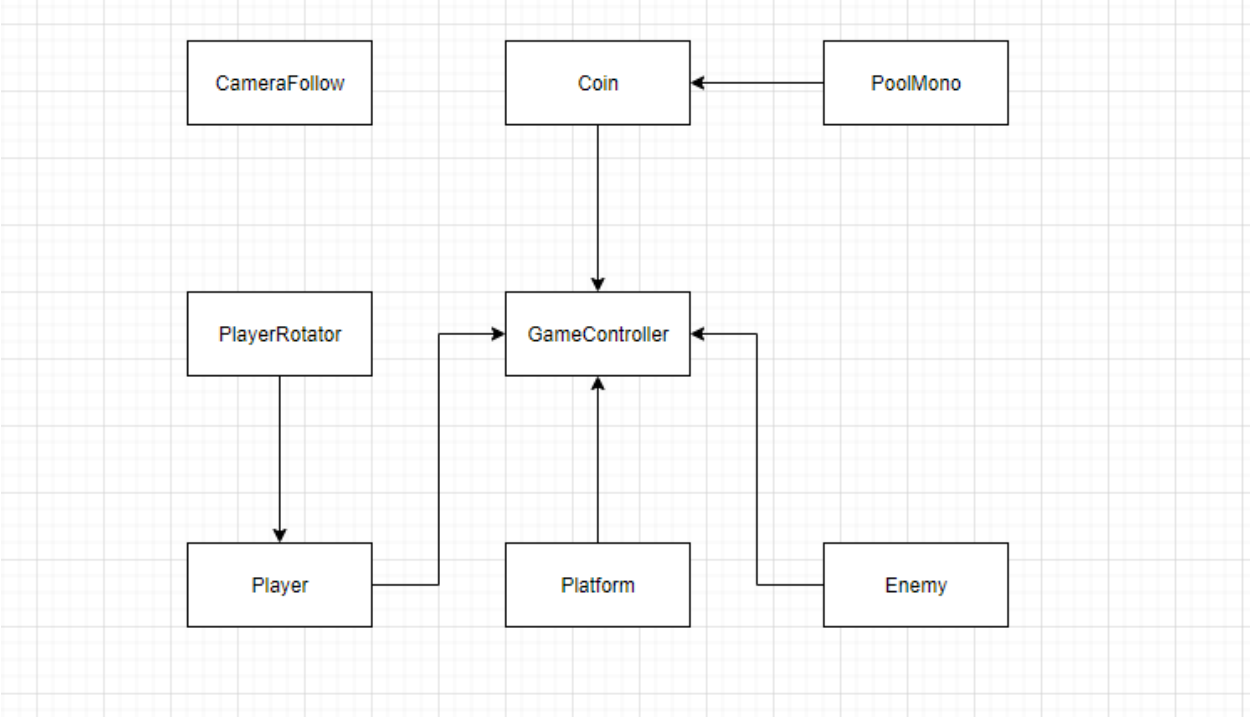

Рисунок 3.1 — Діаграма класів системи

3.2 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;
```

```
    }
}
```

3.3 Розробка графічного інтерфейсу.

Гра починається з меню, в якому вказано найкращий рахунок та показано, якими кнопками виконується управління (рис. 3.2).

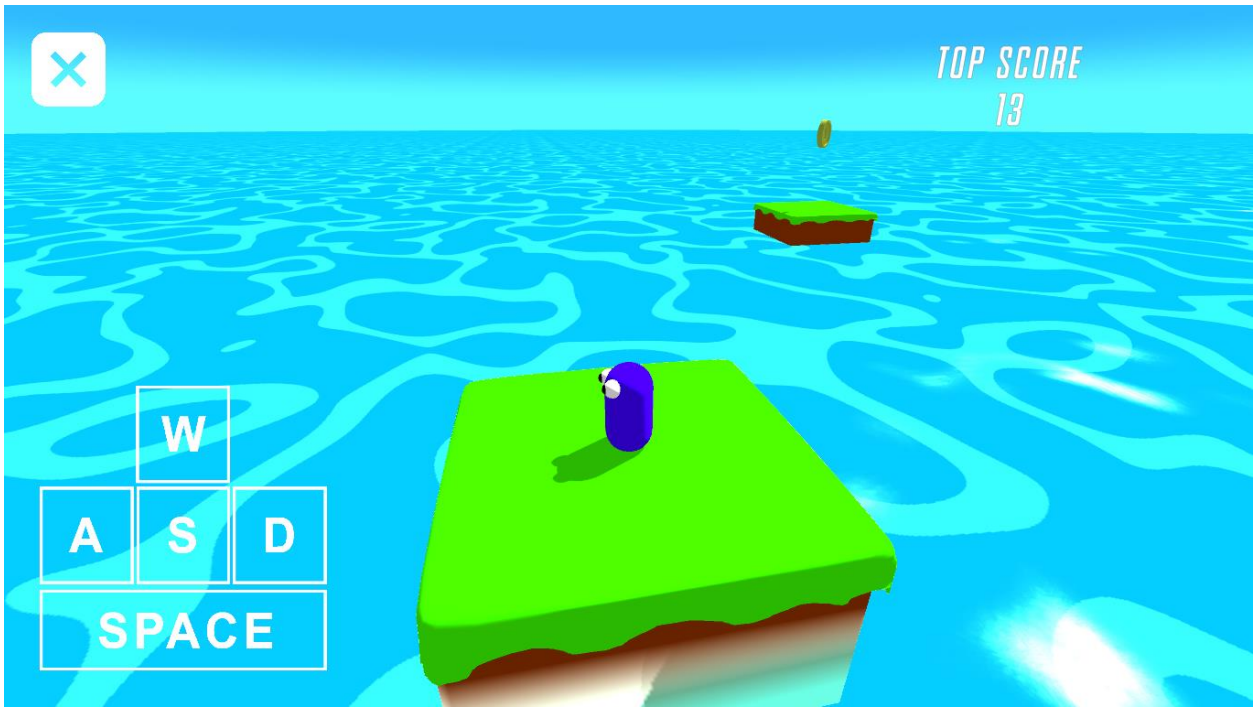

Рисунок 3.2 — Меню

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 3.3.

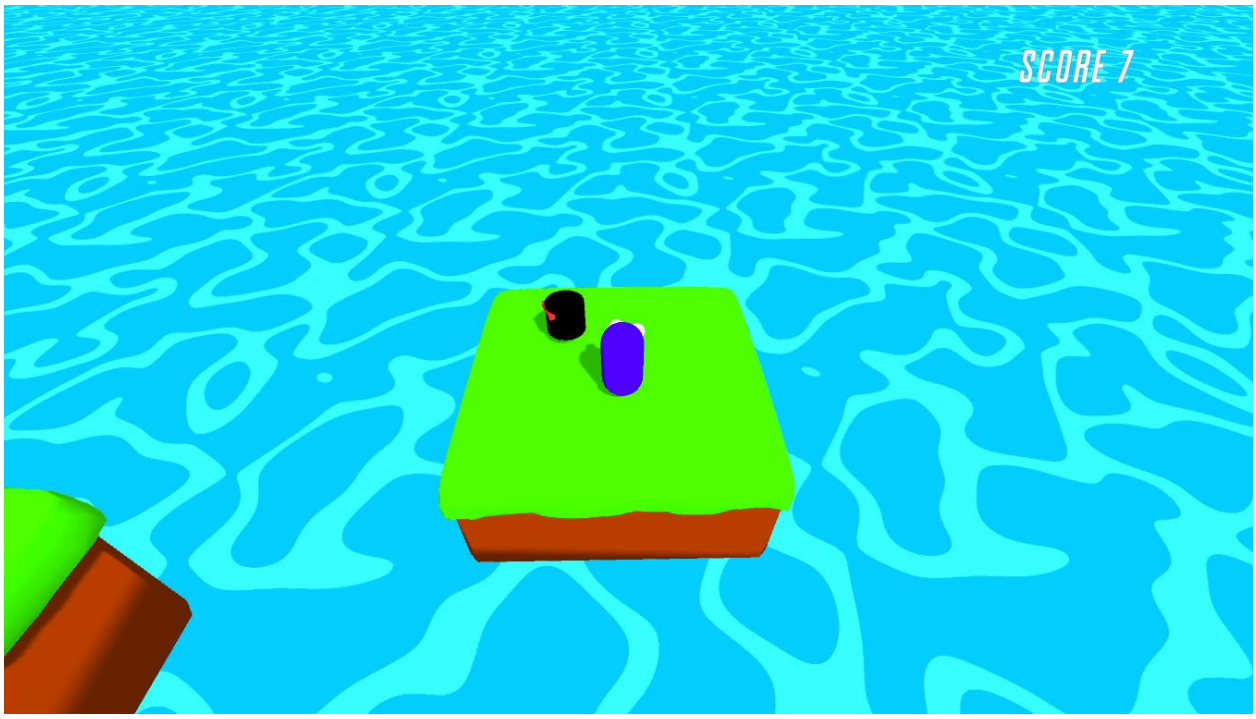

Рисунок 3.3 — Ігровий процес

При програші користувачеві показується екран програшу (рис. 3.4.)

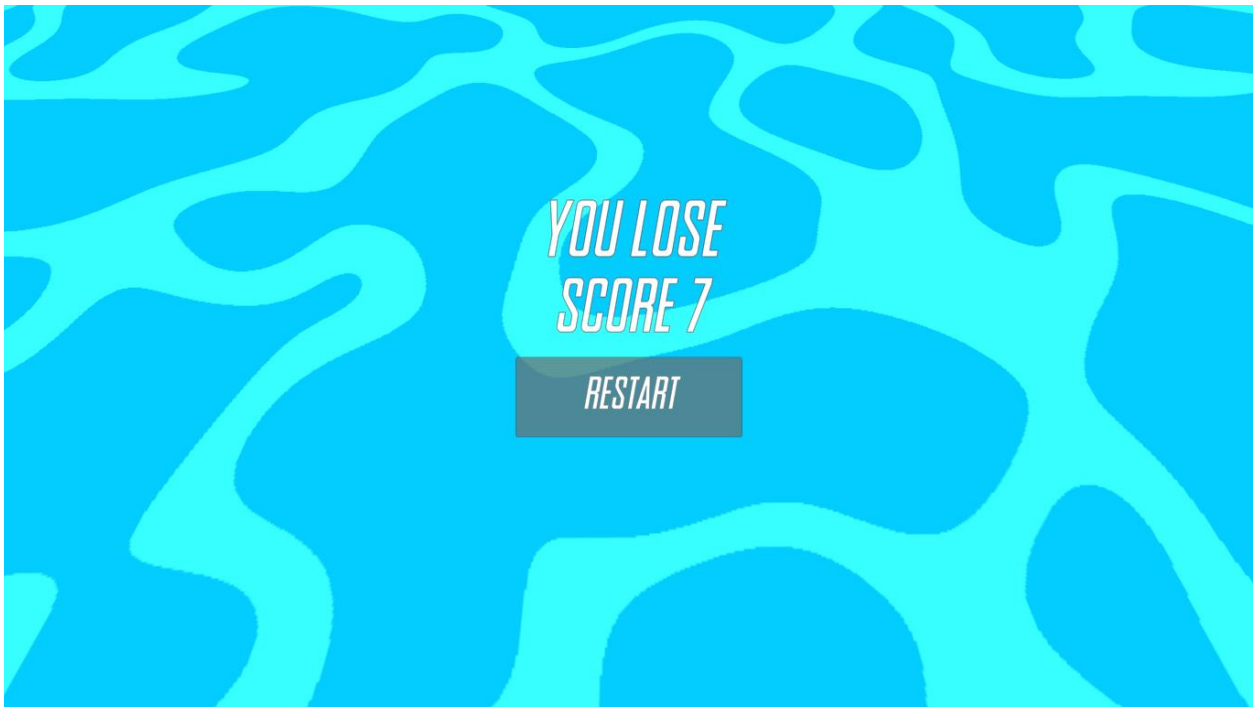

Рисунок 3.4 — Екран програшу

3.4 Тестування

Тестування ПЗ – це акт перевірки артефактів і поведінки програмного забезпечення, що тестується, шляхом перевірки та верифікації. Тестування ПЗ окрім того дає змогу створити об’єктивне, незалежне уявлення про програмне забезпечення для аналізу зі сторони бізнесу ризиків впровадження програмного забезпечення. Методи тестування включають, але не обов’язково обмежуються:

- аналіз вимог до продукту щодо повноти та правильності в різних контекстах, таких як галузева перспектива, бізнес-перспектива, доцільність та життєздатність впровадження, зручність використання, продуктивність, безпека, міркування інфраструктури тощо.
- перегляд архітектури продукту та загального дизайну продукту
- робота з розробниками продуктів над удосконаленням техніки кодування, шаблонів проектування, тестів, які можна написати як частину коду на основі різних методів, таких як граничні умови тощо.
- виконання програми для перевірки поведінки
- перевірка інфраструктури розгортання та пов’язаних скриптів та автоматизації
- брати участь у виробничій діяльності, використовуючи методи моніторингу та спостереження

Тестування ПЗ показує користувачам або спонсорам об’єктивну оцінку якості ПЗ та ризики його збою.

Несправності та збої

Баги у програмному забезпеченні виникають через наступний процес: Програміст робить створює код, що створює збій або неочікувану поведінку у коді, що виконується. Якщо ця помилка існує, у іноді система дає неправильні результати, що призводить до збою.

Не всі баги обов'язково створюють збої. Наприклад, помилки в мертвому коді ніколи не призведуть до збоїв. Несправність, яка не виявила збоїв, може призвести до збою при зміні середовища. Прикладами може бути інше програмне забезпечення, котре виконується на новому комп'ютері, зміна вихідних даних або взаємодію з іншим програмним забезпеченням. Одна несправність може призвести до широкого спектру симптомів відмови.

Не всі помилки програмного забезпечення викликані помилками кодування. Часто причиною дорогих дефектів у ПЗ є недостатньо описані або недостатньо зрозумілі вимоги, тобто нерозпізнані вимоги, які призводять до помилок, пропущених розробником програми. Прогалини вимог часто можуть бути нефункціональними вимогами, такими як тестованість, масштабованість, ремонтопридатність, продуктивність та інші вимоги. безпеки.

Статичне, динамічне та пасивне тестування

У тестуванні ПЗ є багато підходів. Огляди, списки to-do (покрокові) або перевірки – це статичне тестування, або мануальне, у той час як написання коду для тестування коду із визначеними випадками – динамічне, або автоматичне.

Мануальне тестування часто є неявним, як-от коректура, а також коли інструменти програмування/текстові редактори вихідний код на помилки до компіляції, а компілятори перевіряють синтаксис і взаємодію код у статичному розумінні. Автоматичне тестування відбувається під час запуску самої програми. Код для автоматичного тестування може бути написаним до того, як програма буде на 100% завершена, щоб перевірити окремі логічні частини коду та застосувати до окремих функцій або класів. Типовими методами для них є або використаня підмін із потрібною логікою, або виконання з середовища налагодження.

Мануальне тестування виконується людиною та перевіряє код поверхнево, а автоматичне виконується написаним кодом та перевіряє код ізсередини.

Пасивне тестування має на увазі перевірку поведінки системи без взаємодії з кодом або самою програмою. У випадку пасивного тестування тестувальники не дають ніяких тестових даних, а читають журнали системних записів та трасування. Вони шукають моделі та специфічну поведінку, щоб приймати якісь рішення. Це пов’язано з перевіркою часу виконання в автономному режимі та аналізом журналу.

Дослідницький підхід

Дослідницьке тестування — це підхід до тестування ПЗ, коли одночасно досліджується продукт, проектується тест та одразу виконується. Джем Канер, який ужив цей термін уперше у 1984 році і визначив цей тип тестування як «стиль тестування ПЗ, котрий підкреслює дає свободу та створює відповідальність для окремого тестувальника за постійну оптимізацію якості своєї роботи, розглядаючи тест- пов’язане навчання, дизайн тесту, виконання тесту та інтерпретація результатів тесту як взаємодопоміжні дії, які виконуються паралельно протягом усього проекту».

«Коробковий» підхід

Методи тестування ПЗ зазвичай поділяються на тестування білого та чорного ящиків. Ці два підходи описують точку зору, яку дотримується тестувальник під час розробки тестових випадків. До методології тестування ПЗ також можна застосувати гібридний підхід, або тестування сірого ящика. Оскільки концепція тестування сірого ящика, яка розробляє тести на основі конкретних елементів дизайну, стає все більш популярною, це «довільне розходження» між тестуванням чорного та білого ящиків дещо зникло.

ВИСНОВКИ

В ході виконання роботи було виконані наступні завдання:

- Провести огляд предметної області
- Визначити, що таке розробка відеоігор
- Визначити поняття платформера
- Оглянути процес створення відеоігор
- Обрати рушій
- Обрати мову програмування
- Обрати середовище розробки
- Спроектувати внутрішню будову
- Розробити графічний інтерфейс
- Розробити основні механіки
- Провести тестування

Завдяки чіткому виконанню поставлених на початку роботи завдань в результаті створено повноцінний відеогру в жанрі платформер.

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

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

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт рушія Unity [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.unity3d.com
2. Офіційний сторінка розширення для Unity Cinemachine [Електронний ресурс]:[Веб–сайт] – електронні дані. – Режим доступу: https://unity.com/unity/features/editor/art-and-design/cinemachine
3. Офіційний сторінка розширення для Unity Bolt [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://assetstore.unity.com/packages/tools/visual-scripting/bolt-163802
4. Стаття про ігри в жанрі платформер в енциклопедії Britannica [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.britannica.com/topic/electronic-platform-game
5. Стаття про ігри в жанрі платформер Wikipedia [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://en.wikipedia.org/wiki/Platform_game
6. Стаття про українські ігри-платформери [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://leogaming.net/ua/site/news/maloizvestnye-kompyuternye-igry-ot-ukrainskih-razrabotchikov-1-nachalo
7. Типи платформ у іграх-платформерах [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.idtech.com/blog/10-types-of-platforms-in-platform-video-games
8. Домашня сторінка рушія Unity [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://unity.com/
9. Домашня сторінка рушія CryEngine [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.cryengine.com/
10. Домашня сторінка рушія Unreal Engine [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.unrealengine.com/

11. Леньо В. Я. Платформери як явище і їх сутність. *Сучасні інфокомунікаційні технології:* матеріали наук.-тех. конф., м. Київ, травень 2022 р. / Державний університет телекомунікацій, кафедра інженерії програмного забезпечення. Київ, 2022
12. Леньо В. Я. Засоби для створення 2D-платформерів. *Сучасні інфокомунікаційні технології:* матеріали наук.-тех. конф., м. Київ, травень 2022 р. / Державний університет телекомунікацій, кафедра інженерії програмного забезпечення. Київ, 2022
13. Kent S. The ultimate history of video games. / Kent S. - New York : Crown, 2001. - 624 p.
14. Bycer J. Game Design Deep Dive: Platformers. / Bycer J. - Florida : CRC Press, 2019. - 152 p.
15. Schell J. The Art of Game Design / Schell J. - Florida : CRC Press, 2008. - 520 p.
16. Fowler M. UML Distilled / Fowler M. - Boston : Addison-Wesley Professional, 2003. - 208 p.
17. Price M. J. C# 10 and .NET 6 / Price M. J. - Birmingham : Packt Publishing, 2021 - 824 p.
18. Visual Studio Documentation [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/visualstudio
19. Mehta, Vijay. "Extending Visual Studio 2005". CodeGuru. Archived from the original on March 17, 2010. Retrieved January 1, 2008.
20. C# Language Documentation [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/

ДОДАТОК А

ПРЕЗЕНТАЦІЯ ДО ЗВІТУ

Презентація до звіту

Леньо В. Я.

Державний Університет Телекомунікацій

група: ПД-41

1

Індивідуальне завдання

- Огляд та аналіз літературних та інтернет джерел, існуючих рішень ігор в жанрі платформер
- Визначити недоліки існуючих рішень
- Оглянути інструменти, котрі можуть бути використані при розробці ПО для кваліфікаційної роботи бакалавра
- Визначити об’єкт, предмет, мету та постановку завдань кваліфікаційної роботи бакалавра

2

Аналіз та загальна характеристика ігор-платформерів

Характерними ознаками ігор можна назвати:

- Взаємодія з органами чуттів людини
- Взаємодія гравця з елементами ігрового світу

Ігри поділяються на жанри в залежності від типу взаємодій користувача, але не в залежності від візуальної складової або наративної.

Платформер – жанр ігор, що характеризується пересуванням персонажу по різної висоти платформам. Зародився в Японії у 80-х роках 19 століття.

Рисами жанру є:

- Керування персонажем за допомогою пересування, стрибків, ривків, атак. Стрибки це один з найголовніших рухів.
- Динаміка того, що відбувається на екрані.

Мета, об’єкт та предмет дослідження

Останнім часом ігри в жанрі платформер переживають занепад та не користуються попитом. Потрібно створювати легкі, швидкі ігри такого жанру, котрі користувалися би попитом в Україні, як цільовій країні. Тому розроблений продукт є актуальним.

- **Мета роботи:** розробка додатку в жанрі платформер
- **Об’єкт дослідження:** процес розробки комп’ютерної гри в жанрі платформер зі збереженням основних рис жанру
- **Предмет дослідження:** комп’ютерна гра-платформер

Розглянуті ігри	Donkey Kong	Limbo	Braid	Celeste
Платформи	Arcade	Xbox 360, PlayStation 3, Windows, OS X, Linux, Xbox One, PlayStation 4, PlayStation Vita, Nintendo Switch, iOS, Android	Xbox 360, Microsoft Windows, Mac OS X, Linux, PlayStation 3	Linux, macOS, Microsoft Windows, Nintendo Switch, PlayStation 4, Xbox One, Stadia
Складність сюжету	Відсутній	Спрощений	Складний	Складний
Доступність	Від $7.99	Від $9.99	Від $14.99	Від $19.99
Зручність інтерфейсу	-	+	+	+
Візуальна складова	-	+	+	+
Локалізація на українську мову	-	-	-	-
Інші особливості	Засновник жанру	Специфічна стилістика	Унікальна механіка	Унікальне графічне оформлення

Технічні завдання

- Просте та захоплююче керування
- Простий сюжет
- Гарне візуальне оформлення
- Вороги, система здоров’я гравця та ворогів
- Можливість атакувати
- Поліпшення у вигляді сильнішого удари, ліків, броні, золота

Висновки

1. Проаналізовано процес розробки комп’ютерних ігор-платформерів
2. Розглянуто та деконструйовано існуючі ігри даного жанру
3. Визначено об’єкт, предмет та мету бакалаврської роботи
4. Визначені головні функції додатку бакалаврської роботи та практичні шляхи до реалізації

Дякую за увагу !