МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2021 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

РОЗРОБКА 2D - ГРИ НА ПЛАТФОРМІ UNITY

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810325

Виконав студент 4-го курсу, групи 402
_____ І. І. Чернов
«___» _____ 2022 р.

Керівник: професор кафедри інтелектуальних інформаційних систем, д-р техн. наук
_____О. П. Гожий
«___» _____ 2022 р.

Миколаїв – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2021 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Чернову Іллі Ігоровичу.

1. Тема кваліфікаційної роботи «Розробка 2D – гри на платформі Unity».

Керівник роботи Гожий Олександр Петрович, д-р техн. наук, професор.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «07» грудня 2021 р. № 318

2. Строк представлення кваліфікаційної роботи студентом «27» червня 2022 р.

3. Вхідні (початкові) дані до роботи: тренди ігрової індустрії, системи та інструменти для створення ігрових застосунків.

Очікуваний результат: 2D – гра в жанрі платформеру із використанням алгоритмів пошуку шляху на базі ігрового рушія Unity.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз жанру платформеру та механік притаманних йому.

- аналіз вже створених проектів для кращого розуміння завдання.
- розробка основних механік застосунку.
- аналіз створення штучного інтелекту для відеоігор.
- розробка різних сценаріїв поведінки.
- дослідження створення візуалу для ігор.
- створення візуальної частини застосунку.
- аналіз графічного інтерфейсу в іграх.
- розробка графічного інтерфейсу для застосунку.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Охорона праці на робочих місцях у відділі розробки програмного забезпечення ЧНУ ім. Петра Могили»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Макарова О. В., старший викладач	

Керівник роботи __д-р техн. наук, проф. Гожий О. П._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Чернов І. І._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «__» _____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Розробка 2D – гри на платформі Unity

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	10.11.2021	15.11.2021	Виконано
2	Отримання завдання на виконання	21.11.2021	21.11.2021	Виконано
3	Складання календарного плану	05.12.2021	06.12.2021	Виконано
4	Робота над аналітичною частиною БКР	10.01.2022	11.03.2022	Виконано
5	Робота над практичною частиною БКР	15.02.2022	30.05.2022	Виконано
6	Попередній захист БКР	31.05.2022	31.05.2022	Виконано
7	Оформлення звіту з БКР	01.06.2022	10.06.2022	Виконано
8	Перевірка БКР рецензентом	16.06.2022	17.06.2022	Виконано
9	Захист БКР	27.06.2022	27.06.2022	Виконано

Розробив студент Чернов І. І. _____
(прізвище та ініціали) (*підпис*)
Керівник роботи професор кафедри інтелектуальних інформаційних систем, д-р техн. наук, Гожий О. П. _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 202__ р.

АНОТАЦІЯ

до бакалаврської кваліфікаційної роботи

Тема: «Розробка 2D – гри на платформі Unity»

Студент: Чернов Ілля Ігорович

Керівник: професор кафедри інтелектуальних інформаційних систем, д-р техн. наук Гожий Олександр Петрович

Кваліфікаційна робота присвячена розробці програмної реалзації 2D – гри в жанрі платформеру на базі ігрового рушія Unity.

Об’єкт дослідження - розробка ігрового застосунку у жанрі 2D платформеру із використанням алгоритмів пошуку шляху на базі ігрового рушія Unity.

Предмет дослідження - сучасні методи та технології розробки ігрових застосунків.

Метою дипломної роботи є аналіз та дослідження трендів ігрової індустрії, методів та систем розробки ігрових програмних застосунків з метою розробки 2D – гри.

Робота складається із фахового розділу та спеціальної частини з охорони праці. Пояснювальна записка дипломної роботи складається із вступу, 3 розділів, висновку, списку використаних джерел та додатків.

У першому розділі розкрита теоретична частина роботи. Проведений аналіз ігрової індустрії та її трендів. Проаналізовано особливості жанрів та їх відповідних механік. Досліджені методи та інструменти для створення ігрових застосунків. У другому розділі проведено моделювання та проектування досліджуваної системи, а саме ігрового застосунку. У третьому розділі описана розробка відеогри. У четвертому розділі розкрито питання спеціальної частини.

Кваліфікаційна робота містить 108 сторінок, 39 рисунків, 25 джерел, 2 додатки.

ABSTRACT

For bachelor`s work

Subject: "Development of 2D - games on the Unity platform"

Student: Chernov Illia Igorovich

Leader: Professor of the Department of Intelligent Information Systems, Dr. Tech. Sciences Gozhyz Olexandr Petrovich

Qualification work is devoted to the development of software implementation of 2D - a game in the genre of platformer based on the game engine Unity.

The object of research is the development of a game application in the genre of 2D platformer using path search algorithms based on the Unity game engine.

The subject of research - modern methods and technologies for developing game applications.

The purpose - is to analyze and study the trends of the gaming industry, methods and systems for developing game software applications to develop 2D - game.

The work consists of a professional section and a special section on labor protection. The explanatory note of the thesis consists of an introduction, 3 chapters, conclusion, list of sources and appendices.

The first section reveals the theoretical part of the work. An analysis of the gaming industry and its trends. Features of genres and their corresponding mechanics are analyzed. Researched methods and tools for creating game applications. In the second section, the modeling and design of the studied system, namely the game application. The third section describes the development of a video game.

Qualification work contains 108 pages, 39 figures, 25 sources, 2 appendices.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ШІ – штучний інтелект

NPC – non - player character

RPG – role playing game

URP – Universal Render Pipeline

Пояснювальна записка

до кваліфікаційної роботи

на тему:

«РОЗРОБКА 2D - ГРИ НА ПЛАТФОРМІ UNITY»

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810325

Виконав студент 4-го курсу, групи 402
_____*І.І. Чернов*
(підпис, ініціали та прізвище)
«___» _____ 2022 р.

Керівник: професор кафедри інтелектуальних інформаційних систем, д-р техн. наук
(наук. ступінь, вчене звання)
_____*О. П. Гожий*
(підпис, ініціали та прізвище)
«___» _____ 2022р.

Миколаїв – 2022

ВСТУП

Ігри завжди були частиною розвитку будь – якого організму. Так само, як і тварини люди також, в період юності, граючи зі своїми однолітками пізнають світ, вчатся мислити, або ж виконувати більш специфічні задачі, наприклад, рахувати, писати, і т. д.

Із розвитком людства та появою перших комп'ютерів та мов програмування люди навчились створювати перші комп'ютерні ігри. І якщо в ті часи це було лише забавою, із часом це все переросте в повноцінну ігрову індустрію.

Зараз, в наш час, коли ми маємо комп'ютери, здатні обраховувати інформацію в лічені мілісекунди ігрова індустрія процвітає. Деякі проекти навіть мають наскільки реалістичну графіку, що їх можна сплутати із фото із реального життя.

Раніше працювати в ігровій індустрії вважалось чимось неправильним. Оскільки люди не розуміли як можна заробляти гроші створюючи іграшки. Зараз все змінилось. Над розробкою ігор працюють великі компанії, бюджет даних проектів може сягати сотень міліонів доларів. Та фанатів у даної продукції дуже багато, особливо в наш час пандемії, коли люди були змушені сидіти вдома, кількість людей зацікавлених в відеоіграх різко зросла.

Створення відеогри нічим не відрізняється від створення інших програмних продуктів. У команді необхідні програмісти, менеджери, дизайнери, звукові дизайнери, дизайнери інтерфейсу і т. д. Тобто даний процес потребує людей із навичками з різних сфер. Економісти,, менеджери, художники(2D та 3D) різних направлень, наприклад, концепт художники, художники оточення, дизайнери персонажів; люди які розуміютсья на створенні музики, звуків, ефектів та багато інших.

Із розвитком ігрової індустрії з’являлись і нові жанри. Шутери, пригоди, головоломки, навчаючі, симулятори, платформери, RPG, стелс і т. д. Навчаючі ігри досить популярні серед дітей, оскільки в даних іграх можна навити дитину багато чого, наприклад, нову мову, рахувати, писати, нестандартно чи креативно мислити. Симулятори можуть використовуватись у навчальних закладах, наприклад, для ознайомлення людини із водінням автомобіля чи літака, і для цього не потрібно купувати дорогого обладнання, необхідно лиш придбати симулятор, та мати комп’ютер, на якому він запуститься, що буде значно дешевше.

Темою розроблюваного застосунку було обрано 2D екшн – платформер із використанням алгоритмів пошуку шляху. Для розробки був проведений аналіз ігрової індустрії та жанру платформеру в цілому, також способи використання різних обчислювальних алгоритмів в ігрових проектах.

Платформою для розробки проекту було обрано ігровий рушій Unity, інструментарій якого підходить для ігор різного масштабу. Тобто він використовується як інді – розробниками так і більшими студіями.

Характеристики розроблюваного проекту:

– Мова програмування – C#, мова яка використовується при роботі із рушієм.
– Ігровий рушій - Unity
– Враховані критерії відчуття від гри
– Різні сценарії поведінки у ворогів
– Призначена для одного гравця

Метою роботи є розвиток та покращення моторики, швидкості прийняття рішень за допомогою різних сценаріїв, які будуть представлені гравцеві.

Завдання, які необхідні вирішити для створення застосунку:

1. Аналіз жанру платформеру та механік притаманних йому.
2. Провести аналіз вже створених проектів для кращого розуміння завдання.
3. Розробити основні механіки застосунку.
4. Провести аналіз створення штучного інтелекту для відеоігор.
5. Розробити різні сценарії поведінки.
6. Провести дослідження створення візуалу для ігор.
7. Створити візуальну частину застосунку.
8. Аналіз графічного інтерфейсу в іграх.
9. Розробити графічний інтерфейс для застосунку.

Об’єктом є розробка ігрового застосунку у жанрі 2D платформеру із використанням алгоритмів пошуку шляху на базі ігрового рушія Unity.

Предметом є сучасні методи та технології розробки ігрових застосунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ

1.1 Жанр платформеру

Жанр платформеру один із самих давніх жанрів комп’ютерних відеоігор.

Super Mario, Contra, Prince of Persia, Castlevania, Megaman, Sonic the Hedgehog – це все назви ігор – платформерів, деякі назви яких знайомі кожному. Саме ці ігри стали основоположниками жанру, та прийнято наводити їх в приклад коли намагаєшся описати жанр платформеру.

Всі ці ігри мають між собою схожі геймплейні та геймдизайнерські елементи, такі як:

– вид збоку;

– архітектуру рівнів, яка змушує гравця стрибати по платформам;

– управління.

Даний жанр, із розвитком технологій, згодом еволюціонував, а саме перейшов із 2D простору до 3D. Наприклад, Spyro, Shrek, Crash Bandicoot. Раніше згадувані ігри також перейшли до 3D формату. Деякі більш успішно ніж інші. Наприклад перехід Prince of Persia до 3D був жахливим, гра була дуже незручною та закритикованою після виходу.

Згодом розробники із досвідом почали додавати до таких ігор нові механіки. Біг по стінам(вертикальний та горизонтальний), акробатичні елементи та багато іншого. Все це стало можливим із переходом до тривимірного простору.

Але навіть у наш час коли, як вже згадувалось раніше існують ігри із реалістичною графікою, 2D проекти досі створюються і деякі із них навіть можуть поконкурувати із великими 3D іграми. Наприклад, Hollow Knight, Ori and The Blind Forest, Dead Cells, Katana Zero, і т. д. Деякі із них навіть

публікувались та фінансувались студією Microsoft, всім відомим розробникам операційної системи Windows. Деякі мають ретро вид, тобто схожі на ігри із епохи 90-их, які були виконані у стилі піксель арт.

1.2 Ігрові механіки, притаманні жанру 2D платформеру

Стрибок – одна із найважливіших механік для платформеру оскільки даний жанр передбачає, що гравець буде стрибати дуже часто, та на рівнях будуть створені цілі платформинг секції, які необхідно подолати, щоб просунутись далі. Тому реалізації стрибку приділяють багато уваги при створенні таких проектів.

Стрибок має відчуватись, як у реальному житті, також управління має бути чуйним до вводу гравця. Задля цього звичайне відривання від землі модифікується деякими новими механіками. Наприклад, різна висота стрибка, буферизація стрибка, coyote time, стрибки по стінам.

1.2.1 Різна висота стрибка

Різна висота є частиною механіки стрибка. Реалізація даної ігрової механіки надасть змогу гравцеві більш точно керувати персонажем. Суть даної механіки досить проста, як і сказано в її назві забезпечити різну висоту. Реалізація полягає в тому, що доки гравець тримає клавішу, яка відповідає за стрибок персонаж буде підніматись вгору, як тільки гравець відпустить клавішу або досягне своєї максимальної висоти – персонаж впаде.

Механіка досить проста але дає змогу гравцеві краще відчувати персонажа та керувати ним.

В свою чергу дана механіка досить корисна і для дизайнерів рівнів оскільки враховуючи дану особливість керування можна придумати різні задачі та перепони для користувача.

1.2.2 Буферизація стрибка

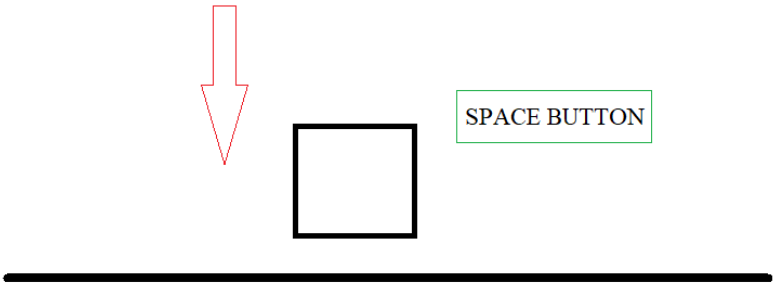

Рисунок 1.1 - Приклад ситуації для буферизації

В іграх даного жанру іноді виникає ситуація коли гравець може натиснути клавішу стрибка трішки раніше ніж персонаж торкнеться поверхні і якщо в грі немає відповідних механік то персонаж просто впаде і не стрибне, коли гравець не буде розуміти, що сталось бо буде вважати, що натиснув все вчасно.

Щоб запобігти цьому використовується буферизація.

Суть даної механіки полягає у додатковій перевірці коли саме гравець натиснув клавішу стрибку, і якщо виконуються специфічні умови то по приземленні персонаж автоматично стрибне вгору.

1.2.3 Coyote time

Coyote time – механіка, яку геймдизайнери використовують для покращення чуйності управління. Названа була в честь койота – персонаж із мультфільму “Road runner”. Там дуже часто зустрічались сцени із тим як він падав із гори, але не відразу, спочатку він декілька секунд стояв нібито на повітрі, а потім розуміючи, що має падати – падав.

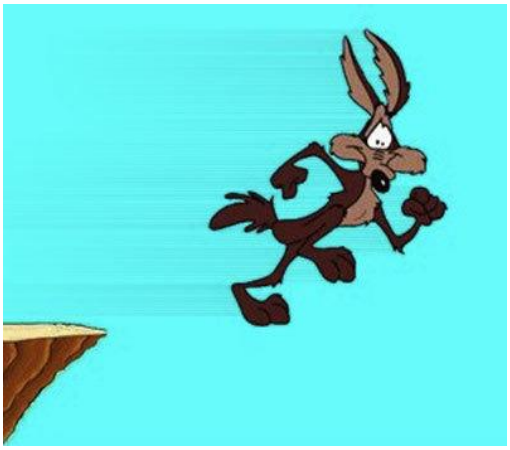

Рисунок 1.2 - Приклад coyote time із мультфільму

Гейм дизайнери створили цілу механіку завдяки цьому, яка покращує досвід користувача.

З тим як в мультфільмі койот падав із гори з деякою затримкою так само і в грі у гравця є декілька мілісекунд для того, щоб здійснити стрибок, хоча сам персонаж вже має впасти.

Було декілька причин по якій ця механіка була впроваджена. По – перше гравці майже завжди полюбляли стрибати із так званої safe zone на платформі, яка знаходиться перед її краєм.

Рисунок 1.3 - Безпечна зона

Або ж навпаки, сильно пізно, коли гравець вважає, що він достиг краю, коли насправді він вже пройшов його і починає падати.

Саме задля запобігання цим ситуаціям використовують Coyote time.

1.2.4 Стрибка по стінам

Стрибки по стінам дуже проста механіка, але додає вертикальності до ігрового процесу. Також це ще один із інструментів дизайнеру рівнів для створення додаткових викликів гравцеві.

Дана механіка існує вже дуже давно та має багато різних варіацій та способів використання. В деяких іграх вона використовується строго для переміщення в деяких як спосіб кинути виклик гравцю.

Реалізується досить просто, необхідно, щоб гравець, знаходячись на стіні, натиснув на клавішу стрибка та клавішу направлення одночасно. Є і інша реалізація, яка потребує простого натискання однієї клавіші стрибку, далі персонаж автоматично стрибне на сусідню стіну.

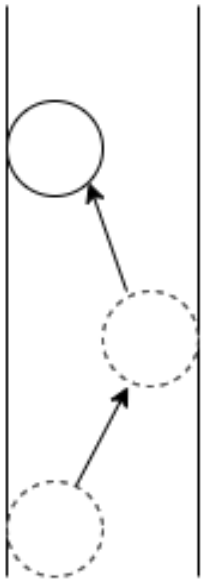

Рисунок 1.4 - Приклад стрибків по стінам

1.3 Проекти, які стали основоположниками жанру платформеру

1.3.1 Super Mario

Super Mario – гра, про яку всі знають. Була видана в 1985 році компанією Nintendo.

Ціллю гри є спасіння принцеси із рук злодія. На шляху у головного героя будуть траплятись різні вороги та виклики.

Бойова система побудована досить просто, щоб подолати ворога Маріо необхідно стрибнути йому на голову, за даними правилами необхідно подолати всі перешкоди. Але задля запобіганню однотипності стичок були додані різні варіанти ворогів. Наприклад черепаха – якій спочатку потрібно застрибнути на голову а потім стрибнути на її панцир, щоб запустити її вздовж платформи. Це можна використати як зброю, адже коли черепаха запущена, вона буде збивати всіх інших з платформ, але це стосується і головного героя. Крім звичайних викликів та стичок на рівні розкидані різні бонуси ти монетки, щоб надати гравцеві тимчасову перевагу і змінити темп гри та додати йому очків, щоб змагатись між собою.

Рисунок 1.5 - Гра Super Mario

1.3.2 Prince of Persia

Гра вийшла в 1989 році. Розроблена Джорданом Мехнером гра використовувала технологію ротоскопингу для анімацій руху головного героя. Завдяки цьому принц рухався досить реалістично. Даний проект є першим кінематографічним платформером та став натхненням для інших авторів у створенні власних проектів, наприклад, Another World.

Згодом даний проект переросте у цілу франшизу, створенням якої буде займатись компанія Ubisoft.

У даній грі гравцю давали більше варіантів управління персонажем. Коли у звичайному платформері гравець мав можливість лише бігати та стрибати у Prince of Persia до цих базових рухів було додано ще декілька. Принц міг здійснити один крок вперед, доволі дивна функція, але враховуючи як саме тут побудовані рівні це доволі часто використовувалось, а саме це допомагало знайти сховані пастки, та пройти підлогу із шипами. Також персонаж міг схопитись за уступ та підтягнутись, або ж навпаки, обережно підійти до краю, використовуючи крок, та повиснути на ньому, далі вже гравець вирішував підтягнутись йому назад чи стрибнути вниз.

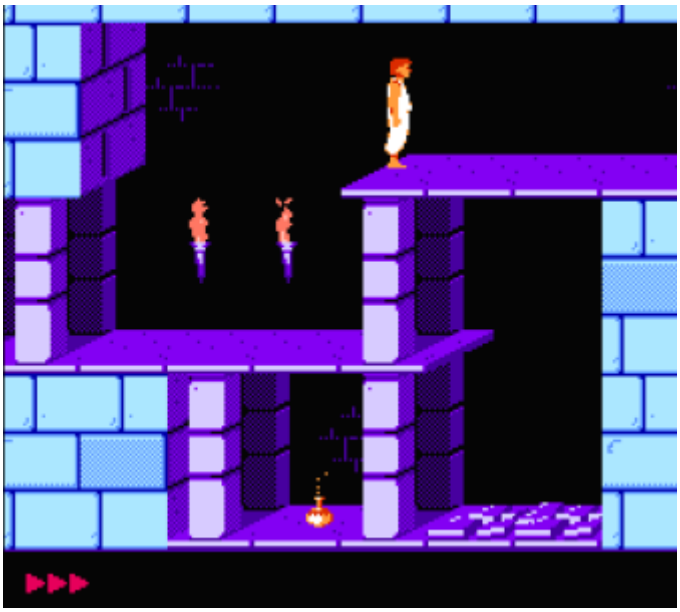

Рисунок 1.6 - Гра Prince of Persia

1.4 Візуальна частина

Візуальна частина також є важливим аспектом гри. Найперша причина цьому полягає в тому, що якщо гра виглядить гарно то на неї вже звернуть увагу, що збільшить шанси її придбання потенційним клієнтом.

Платформери можуть бути виконані в декількох стилях. Це може бути піксель – арт, звичайна 2D анімація, або ж завдяки розвитку технологій, це може бути 3D.

1.4.1 Піксель – арт

Піксель – арт – це зображення, яке створюється за допомогою редагування окремих пікселів.

Даний стиль зародився дуже давно, ще з появою перших відеоігор. Оскільки обчислювальні потужності тогочасних комп'ютерів були досить малі піксельна графіка була єдиним способом відображення об'єктів на екрані.

Та незважаючи на такі обмеження художники, які працювали над створенням спрайтів для тогочасних ігор навчились і за допомогою декількох пікселів створювати образи необхідні для розуміння.

Із розвитком технологій потужність комп'ютерів почала зростати, кількість допустимих пікселів збільшуватись, згодом перетворюючись в десятки тисяч доступних пікселів для редагування. Через це попит на традиційний піксель – арт почав спадати.

Але мода на речі циклічна тому інтерес до піксель – арту почав знову зростати. Так в 2010 році почали виходити ігри які були виконані у даному ретро стилі, що в свою чергу сподобалось людям які ще пам'ятали ту епоху, але не тільки серед них з'явились фанати пікселів, нове покоління теж вподобало даний стиль.

Рисунок 1.7 - Робота в піксель - арт стилі

Зараз цілком розповсюджене явище виходу ігор в піксель – арті. Особливо із різким заповненням ринку відеоігор інді проектами, більшість із яких обрала ретро стиль, через ностальгічний вид та простоту виконання.

Для створення зображень в даному стилі необхідно знати його прийоми, адже недостатньо просто зайди в будь – який графічний редактор, створити файл малого розміру та почати малювати як звичайне зображення. Оскільки створення малюнків в даному стилі досить відрізняється від звичайного тим, що в ньому від розташування одного пікселю може змінитись суть зображення. Наприклад, якщо створювати анімацію в піксель – арті то зміна положення одного пікселю вже досить замітна. Тому існують окремі прийоми, які необхідно знати для того, щоб створювати відповідні зображення у обраному стилі. Так, наприклад, до таких прийомів відносять дітеринг, створення градієнтів, кластерів і т. д.

Доволі часто піксель – арт обирають інді – розробники, оскільки його набагато простіше створювати ніж звичайне 2D. За бажанням та наявності додаткового часу сам розробник може створити графіку для своєї гри, не витрачаючи додаткових коштів. Якщо ж приймається рішення найняти когось для створення графіки, то це все одно виходить дешевше, оскільки працювати в даному стилі доволі просто і він пробачає більшість помилок, які неможливо було б ігнорувати в інших способах створення.

1.4.2 2D стиль

Схожий на піксель – арт але потребує додаткових знань та навичок, оскільки тут вже потрібно працювати із повноцінними зображеннями великих розмірів. Це означає, що для того, щоб створити гарну картинку або ж намалювати персонажа та створити для нього відповідні анімації необхідно мати досвід у малюванні, знати пропорції людського тіла, знати як саме створювати дизайн для персонажів та малювати фони. Також даний стиль не передбачає простої роботи з ним оскільки він не пробачає помилок.

Рисунок 1.8 - Гра Mark of The Ninja

Ігри із зображеннями великого розміру трапляються не так часто як піксель – арт. Оскільки їх дорожче та складніше створювати.

1.4.3 3D стиль

Першою тривимірною грою була Maze Wars створена в 1973 році.

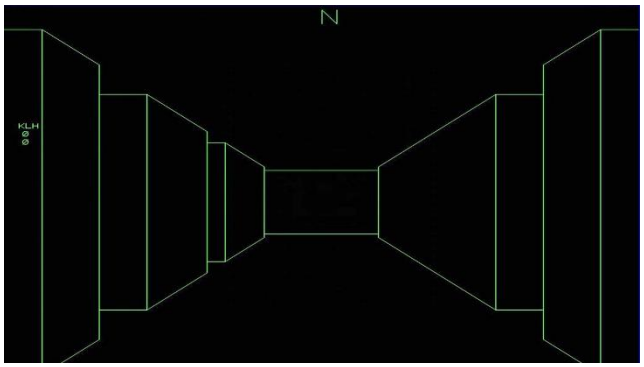

Рисунок 1.9 - Гра Maze runner

В той час тривимірні ігри не виглядали так як зараз. Об’єкти скоріше за все нагадували образи, це схоже на режим wireframe у деяких 3D редакторах.

Із розвитком комп'ютерів розробники отримали можливість використовувати повноцінні 3D моделі в своїх проектах та додати глибини до ігрового процесу. Але і ігри в основі яких лежало переміщення лише в двух площинах почали використовувати 3D.

Рисунок 1.10 - Гра Trine

Це дало змогу створювати реалістичне зображення, персонажів та анімації.

Для створення гри в 3D потрібно мати специфічний набір навичок. Треба знати деякі програми для створення 3D моделей, скульптингу, текстурингу, рігінгу, 3D анімації.

Через це розробка ігор в 3D дорога. Наприклад, створення моделі персонажу чи створіння для великого проекту може займати неменше неділі іноді більше, це не враховуючи, що після скульптингу йдуть інші кроки перед фінальним варіантом, який буде в грі.

1.5 Анімація

Після візуальної частини, наступне на що зверне увагу гравець після запуску гри буде анімація. Оскільки ігри це імерсійний продукт важливим є те, як саме там виконані рухи на скільки вони відчуваються живими.

Наприклад, Marvel`s Spider – Man який являється ексклюзивом PlayStation 4 – гра яку вподобали не тільки за ігровий процес а і за анімації руху головного героя. Як при польоті відчувається, що ти керуєш справжньою людиною. Це звісно заслуга не тільки аніматорів, оскільки над створенням рухів працювали і каскадери в костюмах захоплення руху, використовувались спеціальні технології, inverse cinematics і т. д. Все це в сумі дало такий результат, що не відчувши повноційного ігрового процесу гравець вже хоче грати в даний продукт.

Як вже згадувалось раніше ігри можуть бути виконані в 2D або 3D стилі. Іноді вони поєднуються. Отже, анімація також може бути в 2D або 3D.

1.5.1 2D анімація

2D анімація знайома кожному, адже кожен із нас дивився мультфільми в дитинстві. Даний спосіб створення ілюзії руху повністю транслюється і на ігрову індустрію із усіма його правилами та особливостями. Єдине, що аніматорам тепер не треба малювати все в живу, а можна використовувати спеціальне програмне забезпечення.

Дана анімація створюється за допомогою програвання серії зображень з певним інтервалом, що в свою чергу створює ілюзію руху об’єкту.

Але даний спосіб створення анімації потребує деяких знань та навичок. По – перше аніматор повинен вміти малювати. Якщо це аніматор який працює із персонажами, то додатково він має знати анатомію та пропорції людського тіла або ж створіння для якого буде створюватись анімація. До цього потрібно знати перспективу, форму для того, щоб вміти “крутити об'єкти в просторі” та саме головне – принципи анімації, оскільки без них рухи будуть не реалістичними та схожі в деякій мірі на робота.

1.5.2 3D анімація

Створення анімації в тривимірному просторі дещо схоже на їх створення в двовимірному просторі але має свої відмінності. Перед тим як почати створювати 3D анімацію аніматор має отримати змодельований об’єкт, та, якщо це персонаж він має пройти стадію рігінгу. Рігінг це створення скелетної структури для моделі створіння, щоб потім, рухаючи їх, аніматор міг створювати анімаційні кліпи.

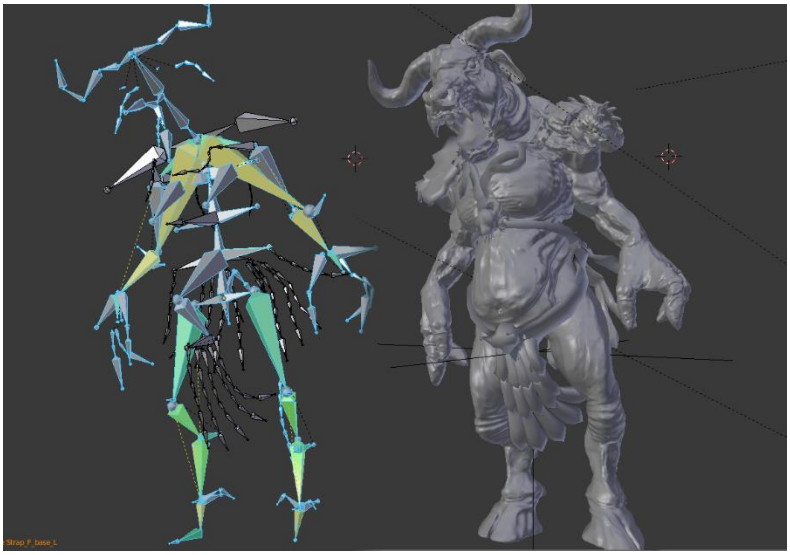

Рисунок 1.11 - Скелет 3D моделі

Завдяки даній технології можна зекономити час на створенні нових анімацій, а саме, якщо новий персонаж має схожі пропорції із тим для якого анімаційна кліпи вже були створені, то можна їх використовувати і на новому об’єкті.

Також із часом з’явились технології захвату рухів. Її можна використовувати як для тіла так і для обличчя. При даній технології, як і при класичній має бути готова модель із скелетом, потім необхідно знайти актора зі схожими пропорціями тіла як і в моделі або ж придумати, як наблизити його пропорції до необхідних. Наприклад, у фільмі “Повстання планети мавп” 2011 року використовувалась дана технологія. Мавп грали актори і для того, щоб пропорції тіла актора співпали із пропорціями моделі мавпи використовувались додаткові палиці.

Рисунок 1.12 - Вбрання для захоплення рухів

Інді – проект “Hellblade: Senua’s Sacrifice” також повністю використовував дану технологію. Це було необхідно не тільки для того, щоб анімації просто були гарними. В даному проекті розповідається чутлива історія героїні і для кращого споріднення із нею та розуміння її стану не тільки дивлячись на її обличчя, а і аналізуючи рухи героя було прийнято рішення використовувати технологію захвату рухів. Також в грі дуже багато сцен із крупним планом обличчя героїні, де дуже важливо передати точні емоції персонажу та зробити так, щоб вони відчувались реальними для гравця.

Ще однією перевагою даної технології є те, що можна створити певну систему, яка дозволить в реальному часу бачити змодельовану анімацію. Тобто на майданчику рухи будуть виконувати актори, а в камері буде видно персонажів, на яких ці рухи було спроектовано.

1.6 Штучний інтелект

Штучний інтелект(ШІ) – здатність певної системи самостійно працювати, обробляти та застосовувати отримані дані.

Ціль науковців, які працюють над створенням ШІ – отримати комп’ютер, який по ро розумовим здібностям буде походити на людину,

тобто мати здатність до самостійного навчання, використання отриманих знань, аналізу.

Вперше даний термін був вжитий Джоном Маккарті в 1956 році. Він створив перший ШІ для гри в шахи.

Але аналізуючи визначення ШІ, а саме здатність комп'ютера виконувати розумові завдання притаманні людині, можна зробити висновок, що досі повноцінного штучного інтелекту створено не було.

Дане поняття часто вживають говорячи про нейроні мережі. Нейронна мережа – набір програмного забезпечення для вирішення специфічних чітко поставлених цілей. Алгоритми, на яких працюють мережі, схожі на симуляцію роботи нейронів у людському мозку. Для того щоб мережа почала вірно працювати та видавати очікуваний результат, її, як і людину, спочатку необхідно навчити, а потім вона вже буде готовою до використання.

Використання нейромереж досить популярне в наш час, оскільки вони успішно справляються із рішенням поставлених ним практичних цілей.

Приклади галузей, в яких застосовують нейромережі:

- техніка
- медицина
- фізика
- бізнес
- геологія

Штучні нейронні мережі являють собою систему з масштабною кількістю простих паралельних процесів між якими налагоджена велика кількість зв'язків. Коли така система вперше була створена люди вважали,

що вони майже створили повноцінний штучний інтелект, але помилились. Оскільки, незважаючи на схожість архітектури мережі на людський мозок, вони були досить обмежені у своєму використанні, будучи здатні виконувати тільки специфічні задачі.

Перед початком роботи із нейронною мережею потрібно її навчити. Процес навчання відбувається на певній вибірці враховуючи відповідні парадигми та правила, які обираються залежно від поставленого завдання.

Процес навчання полягає у визначенні архітектури мережі та ітеративного налаштування вагових коефіцієнтів.

1.6.1 Парадигми навчання штучної нейронної мережі

До парадигм навчання нейронних мереж відносять:

- навчання із вчителем;
- навчання без вчителя;
- навчання з підкріпленням.

1.6.1.1 Навчання із вчителем

Навчання із вчителем – нейромережа має доступ до заздалегідь відомих правильних відповідей. Із кожною ітерацією навчання корегуються вагові коефіцієнти, щоб наблизити відповідь із наступних ітерацій до вірної.

На початку навчання за даною парадигмою вагові коефіцієнти визначаються випадково. Кількість ітерацій та змін коефіцієнтів визначається розробником. Коли мережа починає видавати результат наближений до бажаного навчання вважається завершеним.

Після успішного навчання нейронної мережі, тобто коли вона починає видавати правильний результат на навчаючій множині важливо перевірити її роботу на даних, які в навчанні не використовувались, тобто

на тестовій множині. Якщо при перевірці на нових даних результати роботи мережі незадовільні необхідно продовжити навчання.

1.6.1.2 Навчання без вчителя

Спосіб навчання мережі за яким немає необхідності використовувати зовнішні впливи для корегування ваг із внутрішнім контролем ефективності за допомогою пошуків тенденцій у вхідних сигналах.

Оскільки у мережі немає доступу до заздалегідь відомих вірних відповідей вона має мати інформацію, яка стосується її організації та набір правил.

Парадигма навчання без вчителя спрямована на знаходження зв’язків між окремими групами нейронів, які працюють разом. Якщо на вхід мережі потрапляє сигнал, який активує будь – який із нейронів у групі то активність усієї групи загалом починає зростати. Та навпаки, якщо активність нейрона починає спадати, то активність всієї групи починає зменшуватись.

1.6.1.3 Навчання з підкріпленням

Навчання з підкріпленням являється проміжним варіантом двох попередніх парадигм. Суть даного способу навчання полягяє у використанні іншого блоку, який називається “критикою”. За допомогою даного блоку відслідковується реакція сережовища на сигнали від вхідних даних та проводиться визначення евристичної похибки, яка використовується в процесі навчання.

Даний метод навчання використовується коли ведеться робота із великими системами, в яких точні методи стають нездійсненними.

1.6.2 Правила навчання штучної нейронної мережі

Існує 4 основних правил за якими проводиться навчання нейронної мережі:

– правило корекції за помилкою;

– правило навчання Больцмана;

– дельта правило;

– навчання методом змагання.

Правило корекції за помилкою - являє собою метод, за яким ваги зв’язків не змінюються поки реакція перцептрона являється вірною. Якщо з’являється неправильна реакція вага зв’язку змінюється на одиницю, а знак на протилежний.

Правило навчання Больцмана – правило, за яким вагові коефіцієнти будуть налаштовані так, що стани видимих нейронів задовольнятимуть бажаний розподіл ймовірності.

Дельта правило – правило, яке розвинулось із першого та другого правил Хебба. Полягає у навчанні перцептрону за принципом градієнтного спуску по поверхні помилки

Навчання методом змагання – правило, за яким вихідні нейрони змагаються між собою за активізацію. Даний спосіб навчання дає змогу об’єднати вхідні дані в кластери та модифікувати тільки ваги нейрона – переможця.

1.7 Штучний інтелект в іграх

Ігровий штучний інтелект – набір програмних методик, які використовуються розробниками відеоігор для створення ілюзії наявності інтелекту в неігрових персонажах.

Ігровий ШІ хоч і включає в себе методи притаманні традиційному штучному інтелекту, але не являє собою повноцінний ШІ в звичайному розумінні.

Оскільки розробка ШІ для гри тісно пов’язана із бюджетом проекту, системними вимогами та впливом на ігровий процес розробникам необхідно, балансуючи та враховуючи всі ці вимоги, створити просту

імітацію ШІ, яка буде невимогливою до ресурсів. Через це підходи до створення ігрового штучного інтелекту відрізняються, а саме, притаманні різні види спрощень, та хитростей, які в свою чергу дадуть бажаний результат – наявність ілюзії штучного інтелекту у неігрових персонажів. Наприклад, у іграх жанру шутеру боти мають інформацію про точне положення гравця на мапі, тому вони мають змогу слідкувати крізь стіни за ним, що в свою чергу приводить до миттєвої смерті гравця, оскільки в нього не було часу на реакцію. Щоб вирішити дану проблему розробники штучно знижують влучність ботів. Ще одним прикладом може бути гра Doom, теж в жанрі шутера. У даному проекті гравцеві дано більше можливостей для переміщення, а саме, окрім звичайних стрибків та бігу він ще має можливість бігу по стінам, ривкам, ковзанням. Тобто у даному проекті окрім горизонтального ігрового процесу присутній і вертикальний. Задля того, щоб гравець користувався перевагами вертикального геймплею розробники модифікують деякі параметри штучного інтелекту ботів, залежно від дій гравця. А саме, якщо гравець буде малоподвижним, то влучність ботів буде зростати, якщо ж він почне рухатись та збільшувати швидкість власного переміщення їхня влучність почне знижуватись.

Ігрових персонажів із штучним інтелектом поділяють на декілька категорій:

- неігрові персонажі;
- боти;
- моби.

Неігрові персонажі – категорія, до якої відносяться персонажі, дружні або нейтральні до гравця. Зазвичай це можуть бути торговці, квестники або напарники.

Боти – категорія, ворожо настроєних персонажів до гравця. По можливостям вони наближені до персонажу гравця. Зазвичай їх використовують у невеликих числах, або ж і зовсім у боях один на один, для ознайомлення користувача із новим типом ворога. Потім їх кількість може зрости.

Моби – категорія, ворожо настроєних персонажів із низьким рівнем ШІ. З'являються часто у великих кількостях.

Перші спроби створення ігрового ШІ були проведені у іграх 60 – х років. Але він був досить обмежений у своїх можливостях.

В 1990 – х роках, з появою нових ігрових жанрів, розробники почали використовувати скінченні автомати. Але цього досі було недостатньо, це було видно по проблемам які мали деякі ігри із ШІ в той час. Наприклад, в деяких іграх був порушений пошук шляху, в інших, скінченні автомати для керування персонажами, що в свою чергу привело до невірного їх функціонування.

Згодом розробники почали збільшувати кількість методів, які вони використовували для написання ШІ.

GoldenEye 007 – гра розроблена в 1997 році. Була однією із перших, де боти реагували на дії гравця, могли виконувати перекати, та користуватись укриттями. Але недоліком ігрово ШІ в даному проекті було постійне знання ворогів про точне місцезнаходження гравця.

У Halo 2001 року розробникам вдалось покращити поведінку ботів. Тепер вони могли використовувати транспортні засоби і діяти в команді. Вони могли розпізнати деякі загрози, наприклад, гранати, які кинув в них гравець, або транспорт, маючи дану інформацію вони могли здійснювати певні переміщення щоб врятуватись.

Першим проектом із більш – менш продвинутим штучним інтелектом була гра F.E.A.R. 2005 року. Тут можливості ботів були збільшені задля забезпечення цікавого ігрово процесу. Вони могли використовувати укриття, гранати, та різні тактики ведення бою. Так наприклад, якщо гравець засиджувався в одному укритті надовго боти намагались вибити його звідти гранатами. До тактик, які вони використовували, можна віднести: штурм, відступ, виклик підкріплення, спроби оточення гравця. Задля забезпечення реалістичності поведінки ботів розробники дали можливість їм перекрикуватись один з одним під час бою із гравцем. В свою чергу це давало деяку перевагу користувачеві оскільки почувши дану інформацію він міг діяти наперед.

Left 4 Dead – онлайн кооперативний шутер розроблений в 2008 році компанією Valve. Окрім поведінкових сценаріїв які були розроблені для кожного виду боту в грі розробники створили інструмент процедурного наративу. The Director – система ігрового штучного інтелекту, яка, аналізуючи дії гравців, створювала для них відповідні сценарії. Наприклад, якщо гравці досить гарно грають та справляються із усіма цілями система ускладнить для них ігровий процес, додасть більше ботів, змінить їхній тип, буте створювати різні групи противників.

Фінальною метою розробки ігрового штучного інтелекту є досягнення рівня, при якому гравцеві буде складно відрізнити справжню людину від бота. Але в ігровому ШІ завжди будуть присутні штучні спрощення його поведінки задля забезпечення комфорту гравця.

1.8 Алгоритми пошуку шляху

Пошук щляху – задача, розв’язком якої є побудова найкоротшого шляху від токи А до точки Б за допомогою певної програми. Дана задача ґрунтується на алгоритмі Дейкстри.

До основних задач пошуку шляху відносять:

– пошук шляху між вузлами графа;

– знаходження оптимального найкоротшого шляху.

Існують різні алгоритми, які можна використовувати для рішення задачі пошуку шляху, але не всі вони оптимальні. Так наприклад пошук в ширину або глибину побудовані на принципі грубого перебору, досліджуючи кожний доступний шлях на графі. Вони підійдуть для специфічних задач, але для більш великих графів та для проблеми пошуку оптимального шляху вони не будуть кращим вибором.

1.8.1 Алгоритм Дейкстри

Даний алгоритм працює лише для графів із додатними вершинами. Робота даного алгоритму розпочинається з початкового вузла та спрямована на пошук вузлів – кандидатів. Кожна наступна ітерація – це пошук вузлів із найменшою відстанню до початкового. Вузли які біли позначені як закриті та близькі до них додаються до спеціального набору. Цикл повторюється до того моменту доки не буде знайдено найкоротшого шляху.

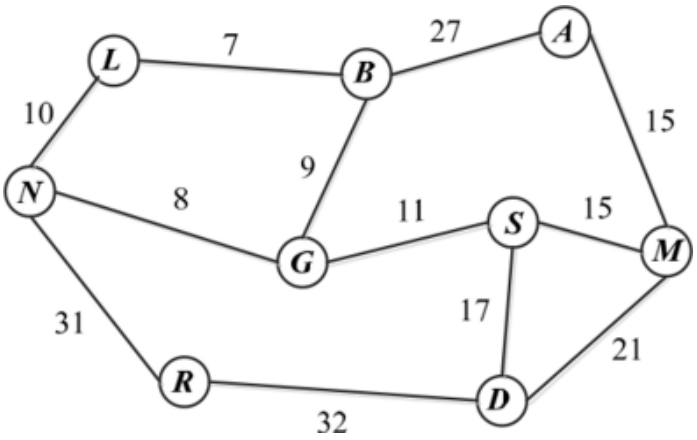

Рисунок 1.13 - Граф алгоритму Дейкстри

1.8.2 Алгоритм A*

Алгоритм A* - алгоритм пошуку шляху, який доволі часто використовується в іграх, є варіантом алгоритму Дейкстри. За даним

алгоритмом необхідно придати вагу відкритим вузлам та знайти приблизну суму ребра та фінішу, що буде їхньою відстанню. Дана відстань є евристичною і являє собою мінімальну можливу відстань між вузлом та фінішом. Це дає змогу алгоритму усунути всі довші шляхи, зменшуючи час роботи.

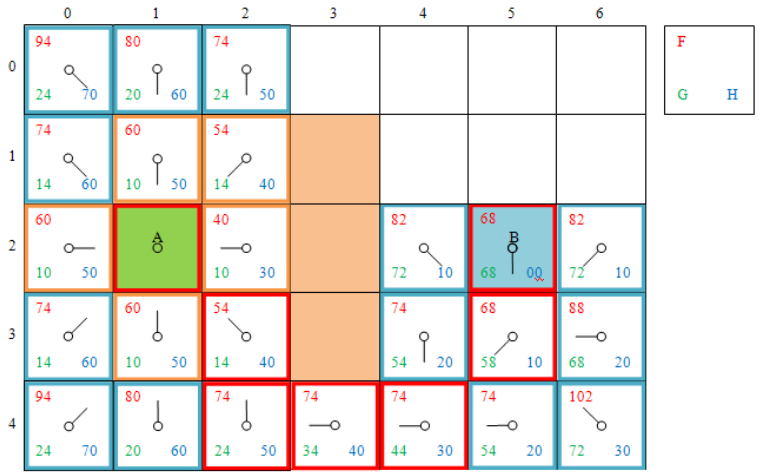

Рисунок 1.14 - Схема роботи алгоритму A*

1.9 Ігрові рушії

Ігровий рушій – центральна частина будь якої гри, яка відповідає за її технічну частину. Рушії дозволяють полегшити роботу над розробкою проекту за допомогою наявного графічного інтерфейсу та заздалегідь створених та вбудованих бібліотек. Однією із головних переваг рушія є можливість створювати мультиплатформені застосунки для таких платформ як ПК, Xbox та PS.

Ігровий рушій відповідає майже за всю основну функціональність застосунку, а саме:

– візуальна частина;

– анімації;

– звук;

- фізичні розрахунки;
- додавання скриптів;
- ігровий штучний інтелект;
- керування ресурсами.

В деяких випадках вже розроблений рушій можна використати для декількох різних проектів.

Сам термін ігрового рушія з’явився в 1990 – х роках. Причиною появи ново терміну стало зростання популярності 3D ігор в жанрі шутеру. Розробники почали економити ресурси під час розробки нових проектів, а саме, почали ліцензувати деякі частини ПЗ із вже створених проектів та використовувати їх в нових. Це дало змогу зменшити витрати на розробку та зекономити час.

Потім ці частини ліцензованого ПЗ почали удосконалювати та давати можливість користуватись іншим розробникам. Наприклад, Unity, Unreal Engine, Frostbite, CryEngine, Godot та інші. В свою чергу розробники рушіїв мали додатковий дохід від кожного проекту, який був розроблений за допомогою їхнього програмного забезпечення. Наприклад, компанія Epic Games, розробники Unreal Engine, отримують 30 відсотків від продаж проектів розроблених з використанням їх технологій.

Часто ігрові рушії мають компонентну архітектуру, тобто дають можливість змінити або доповнити вже існуючі системи новими, більш дорожчими, наприклад Havok, PhysX – для фізичних розрахунків, FMOD – для звуку та інші.

Але висока гнучкість ігрових рушіїв дозволяє використовувати її в інших направленнях окрім ігрової індустрії. Оскільки однією із особливостей ігрового ПЗ є створення графіки в реальному часі, то воно

досить часто використовується в розробці реклам, фільмів, в архітектурній візуалізації.

Ігрові рушії та системи рендерингу графіки побудовані на графічному API, наприклад, Direct3D, OpenGL. Також використовуються низькорівневі бібліотеки, завдяки чому забезпечується апаратно незалежний доступ до частин комп’ютера.

До найпопулярніших ігрових рушіїв належать:

- GameMaker: Studio;
- Unreal Engine;
- Unity.

1.9.1 GameMaker: Studio

GameMaker: Studio – ігровий рушій, розроблений Марком Овермарсом, є потужним інструментом для розробки ігрових проектів. Націлений на розробку двовимірних ігор. Зараз належить компанії YoYo Games. Дане ПЗ надає можливість кросплатформеної розробки, але лише в преміум версіях, безкоштовна дає змогу створити проект тільки для платформи Windows. Може працювати із різними бібліотеками. Присутні інструменти для роботи із скалярними та векторними величинами, математичними виразами. Вбудований фізичний движок Box2D дозволяє проводити маніпуляції із фізикою та фізичними об’єктами.

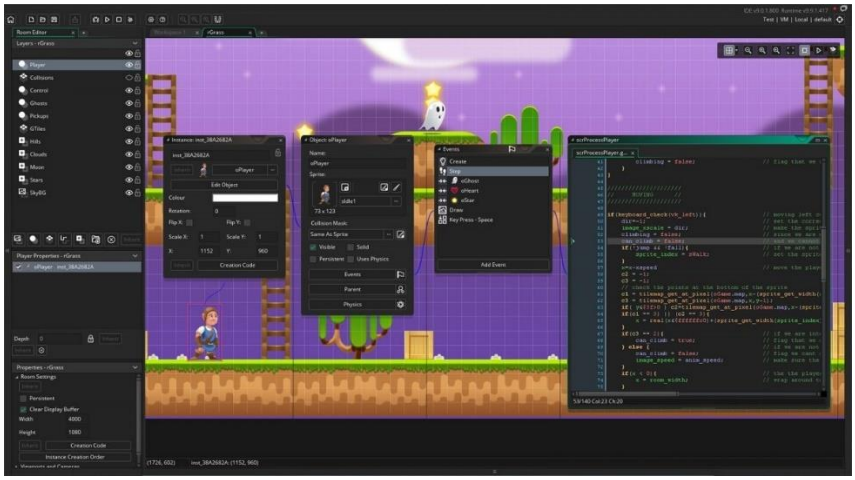

Рисунок 1.15 - Інтерфейс ігрового рушія GameMaker: Studio

Серед успішних ігрових проектів розроблених на базі GameMaker: Studio є:

– Hyper Light Drifter
– Katana Zero
– Undertale
– Серія ігор Hotline Miami

1.9.2 Unreal Engine

Ігровий рушій Unreal Engine один із самих давніх ігрових ПЗ. На ньому розроблялись ще такі проекти як Unreal 1998 року, в честь якого і був названий рушій. Після цього дане ігрове ПЗ використовувалось в безлічі інших проектах. Хоча він був розроблений для створення шутерів, розробники знайшли способи його використання і для інших жанрів. Так в наш час розроблялись RPG, MMORPG, файтинги та інші.

Мовою програмування для Unreal Engine є C++.

Щодо архітектури рушія, то всі елементи представлені у вигляді об’єктів, у яких присутні набори характеристик та клас, за яким визначаються доступні функції.

Рисунок 1.16 - Інтерфейс ігрового рушія Unreal Engine

До основних класів відносяться:

– Actor;

– Pawn;

– World.

Actor – базовий клас, у якому знаходяться усі об’єкти по відношенню до ігрового процесу або в них наявні просторові координати.

Pawn – може бути як моделлю гравця так і об’єкту, що керується ігровим ШІ.

World – об’єкт, який характеризує властивості рівня, фізичну взаємодію об’єктів, тертя, силу тяжіння та інші.

1.10 Ігровий рушій Unity

Unity – ігровий рушій розрахований на багатоплатформену розробку розроблений Unity Technologies. Застосунки, створювані на базі даного ігрового програмного забезпечення, можуть бути виконані в двовимірному або тривимірному просторі. Також є можливість працювати із пристроями віртуальної реальності.

Мовою програмування було обрано C#, раніше була можливість працювати із Boo та JavaScript, але розробники вирішили відмовитись від їх підтримки.

Інтерфейс рушія складається із різних вікон. За замовчуванням на головному виді програми є вікна оглядача ресурсів, інспектор, в якому можна проводити певні маніпуляції із об’єктами, вікно сцени із різними режимами, та оглядач ієрархії проекту.

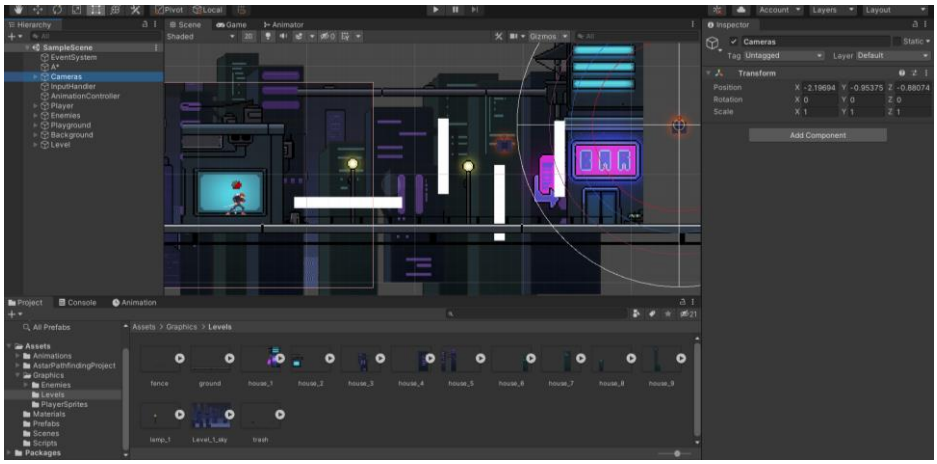

Рисунок 1.17 - Інтерфейс ігрового рушія Unity

Проект поділений на сцени, окремі файли, що мають всю інформацію про певний рівень, який створюється вручну розробником. Файл сцени може містити, об’єкти(3D моделі чи 2D спрайти), об’єкти можуть бути пустими, але містити у своїх властивостях скрипти, тобто виконувати роль контролерів, тригерів, точки збереження чи переходу між сценами.

Кожен об’єкт можна переміщувати, змінювати його масштаб та крутити за допомогою відповідних інструментів або за допомогою спеціальних компонент у вікну інспектору об’єкта. Всім об’єктам на сцені можна присвоїти спеціальний тег або додати до спеціального шару з метою їх групування. Теги та шари створюються користувачем. Об’єкти підтримують систему успадкування, тоді дочірні об’єкти будуть повністю повторювати рух батьківського.

Рушій підтримує фізику твердих тіл та тканин. Можна вручну, в налаштуваннях проекту змінити сценарії взаємодії тіл або силу гравітації.

Рендеринг об’єктів відбувається за допомогою віртуальної камери. На сцені можуть бути присутніми декілька камер, рух яких задається користувачем. Режим роботи камери може бути перспективним або ортографічним, тобто тривимірним або двовимірним. Але перехід камери в ортографічний режим не означає, що розробник втрачає можливість працювати із глибиною. Всі об’єкти як і раніше можна переміщувати вглиб сцени.

Графічний рушій працює із використанням DirectX тільки на Windows платформах; OpenGL – Windows, Mac, Linux; OpenGL ES – мобільні платформи. Для роботи із шейдерами використовується ShaderLab, який підтримує шейдерні програми написані на GLSL, Cg.

Рушій підтримує створення користувацьких скриптів. Після їх створення можна додати їх до об’єктів у вигляді спеціальних компонентів.

До переваг даного рушія можна віднести наявність візуального середовища розробки та мультиплатформена підтримка. До того ж мультиплатформеність заключається не тільки в тому, що проекти створені за допомогою даного ігрового ПЗ можуть запускатись на різних платформах, а і в наявності необхідних інструментів для розробки під них.

До недоліків можна віднести ускладнення роботи в складних сценах через обмеження візуального редактору, та деякі проблеми при роботі із зовнішніми бібліотеками.

Даний ігровий рушій досить популярний серед інді – розробників та розробників мобільних ігрових додатків. Велика кількість навчаючих відео та досить гарно зібрана документація по функціональності рушія спрощує роботу та навчання в даному ПЗ.

До найпопулярніших проектів створених на базі даного ігрового рушія можна віднести:

- Серію ігор Ori;
- Rust;
- Valheim;
- Enter the Gungeon;
- Superhot;
- Cuphead;
- Genshin Impact;
- Inside;
- Subnautica.

Висновки до першого розділу

У даному розділі було проаналізовано жанр платформеру та ігрові механіки притаманні йому. Досліджено створення анімацій та візуальної частини для відеоігор, використання штучного інтелекту в ігрових застосунках. Проведений аналіз існуючих ігрових рушіїв, їх переваги та недоліки.

2 МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ІГРОВОГО ЗАСТОСУНКУ

Перед початком роботи необхідно було провести аналіз створюваної системи та створити приблизну її модель.

Для проектування потрібно було обдумати стани в яких може знаходитись гравець, систему класів та їх зв’язки, анімації та як вони будуть працювати із технічною частиною програми відповідні контролери.

Скінченний автомат – абстрактна машина, яка може знаходитись тільки в одному стані в певний момент часу. Зміна станів може відбуватись за виконання деяких умов, наприклад, вводу користувача. Кожному із станів притаманні відповідні специфічні набори функцій.

Для роботи скінченного автомату необхідно визначити який із станів буде початковим та відповідні умови переходу між ними.

В розробці ігрових застосунків скінченний автомат може бути використано для різних цілей. Наприклад, для роботи із анімацією, із станами в яких може знаходитись головний герой. Оскільки, в великих проектах можливості головного персонажу досить великі, скінченний автомат спрощує роботу з ними та імплементацію нових за необхідністю.

В Unity скінченні автомати використовуються для роботи із анімаційними кліпами будь – якого об’єкту для якого вони були створені.

Дані маніпуляції із станами проводять за допомогою вбудованого компоненту Animator (рис. 2.1).

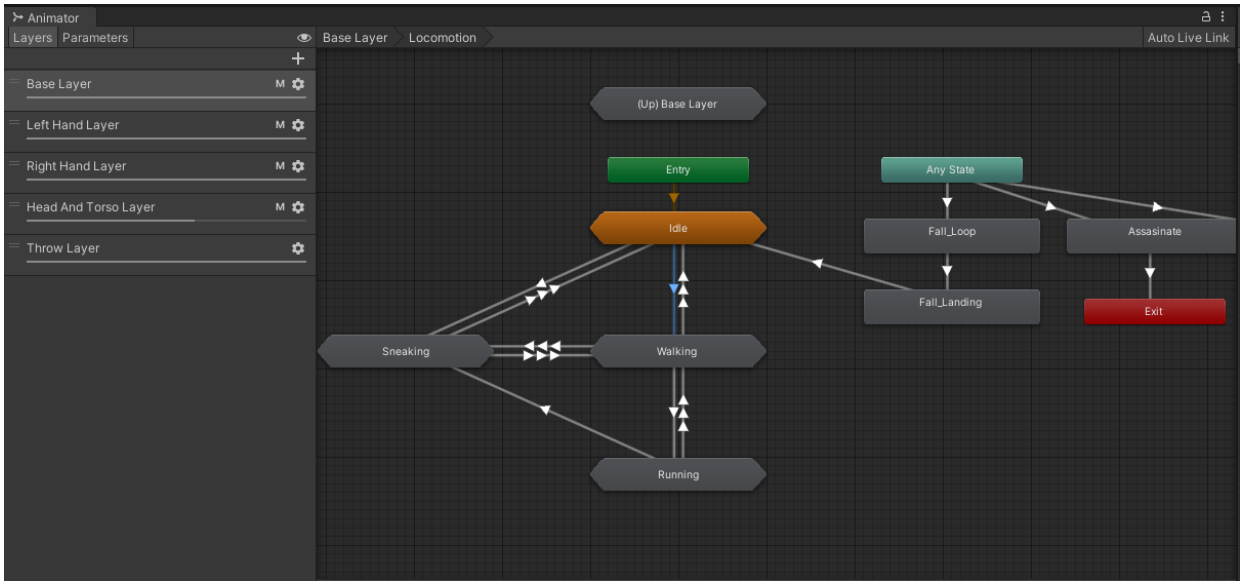

Рисунок 2.1 - Компонент Animator

Тут можна задати початковий стан, переходи між ними, створити відповідні змінні для переходів, відредагувати поведінку анімації, а саме, швидкість переходу від однієї анімації до іншої, плавність та інші. Можна створити групи станів для полегшення роботи.

2.1 Системи гравця

До систем притаманні об’єкту гравця відносяться стани та класи притаманні лише цьому об’єкту.

Рисунок 2.2 - Схема зв'язків між класами

Діаграма станів гравця (рис. 2.3) демонструє можливості та обмеження, які були запроваджені. Далі вона допоможе створити дерево станів в ігровому рушії для анімацій.

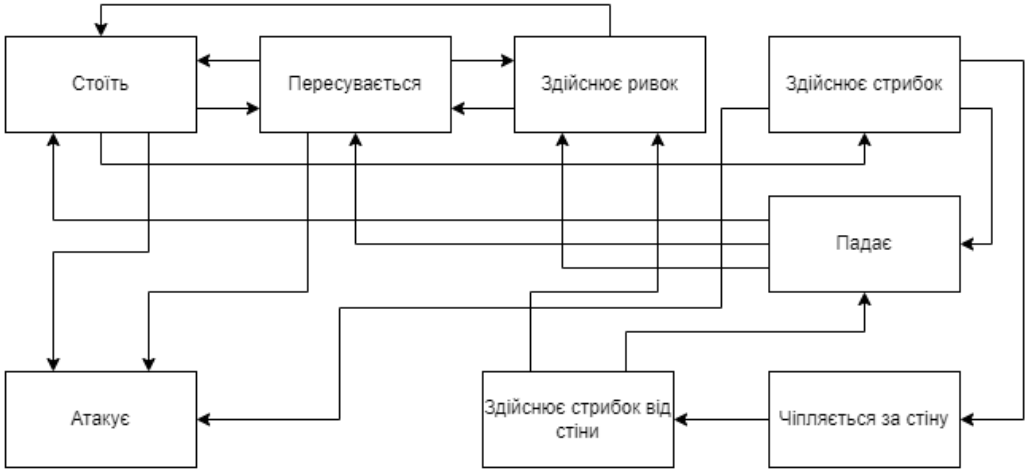

Рисунок 2.3 -Діаграма станів гравця

2.2 Системи ботів

Для ботів система станів (рис. 2.4) стала дещо простішою від систем гравця. Їм притаманні декілька станів:

- Сон
- Активність
- Стоїть
- Рухається
- Атакує

Сон – стан, в якому бот не буде здійснювати жодних дій окрім перевірки потрапляння гравця в зону його дії.

Активність – стан, який активує поведінку бота, а саме він починає шукати гравця та буде переходити у відповідні стани по ситуації, наприклад, стан рухатись чи атакувати.

Стоїть – підстан, в якому бот не буде здійснювати переміщень. Його подальші дії будуть визначені станами сну чи активності.

Рухається – один із підстанів активного, в якому бот буде намагатись триматись гравця на певній відстані. Відстань залежить від типу бота.

Атакує – один із підстанів активного, в якому бот буде проводити спроби атакувати гравця.

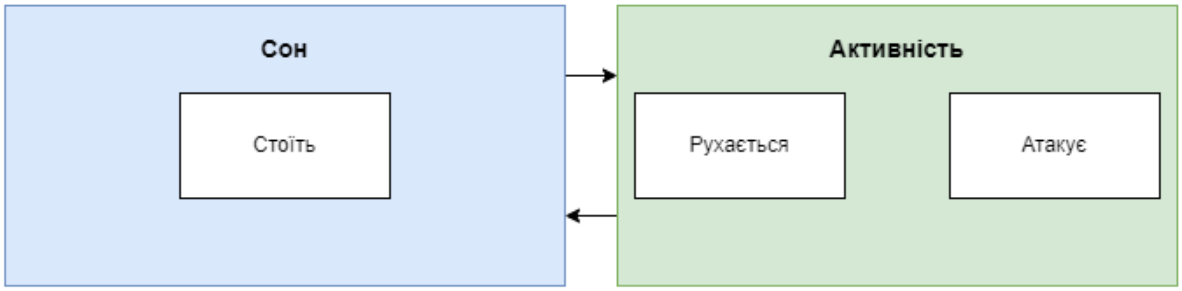

Рисунок 2.4 - Діаграма станів бота

До класів ботів відносяться:

– Seeker;
– AIPath;
– AI Destination Setter;
– Drone Shoot;
– Awake Sleep.

Класи *Seeker*, *AIPath* та *AI Destination Setter* відповідають за пошук шляху до цілі, тобто гравця. За допомогою їх поле гри розбивається на сітку, в якій є інформація про положення головного героя та перешкоди.

Drone Shoot – клас який відповідає за стиль бою бота, в даному випадку це постріл. Атаки можуть бути як в ближньому так і на дальньому боях.

Awake Sleep – клас, який забезпечує активність або сон бота, задля того, щоб він не починав рухатись до моменту поки його не знайде гравець.

В стані сну бот не буде виконувати жодних дій окрім потрапляння гравця в його зону активності. Потім бот перейде в активний стан та почне атакувати користувача.

2.3 Цикл ігрового процесу

Цикл ігрового процесу або пейсинг важлива частина геймдизайну за допомогою якої можна повністю заволодіти увагою гравця та зосередити його на ігровому процесі. Це здійснюється за рахунок вірно розставлених активностей гри, що в свою чергу буде змінювати геймплей для того щоб користувач не займався однією справою протягом всієї гри. Наприклад, якщо в грі потрібно буде просто бігати та стріляти і вона буде досить великою то скоріш за все гравцеві просто набридне виконувати однотипні дії на великому проміжку часу.

Ігровий дизайн – одна із основних частин розробки відеогри, суть якої полягає у проектуванні, створенні та контролю за ігровим процесом.

На початку розвитку ігрової індустрії даними питаннями займались самі програмісти, тобто тоді дизайнерів ігрового процесу як окремої професії не було. З часом із розвитком технологій, збільшенням масштабу ігор та часу який був необхідний на їх подолання стало актуальним питання правильного ігрового дизайну.

Кількість ігрових дизайнерів у компанії може різнитись, їх може бути десяток або ж один, залежить від розміру компанії.

До обов’язків ігрового дизайнеру можна віднести:

- Дизайн світу
- Дизайн ігрових механік
- Дизайн та завдань
- Дизайн рівнів

Дизайн світу – створення світу, опис його історії, подій притаманних йому.

Дизайн ігрових механік – створення та впровадження систем, які мають прямий вплив на ігровий процес, це можуть бути як окремі здібності головного героя, на яких може бути побудова основа бойової системи та спосіб переміщення по світу, так і системи очок, торгівлі, спілкування із неігровими персонажами.

Дизайн завдань – створення сюжетів в яких буде приймати участь гравець, прийняття рішень стосовно, локацій в які він потрапить, персонажів або ворогів, яких він зустрінить.

Дизайн рівнів – планування локацій на яких відбувається ігровий процес, розташування ботів на них, схованих точок інтересів, та пазлів.

Дані обов’язки можуть бути положені на одного ігрового дизайнера або розділені між декількома, які будуть направлятись головним дизайнером.

Пейсинг – важлива частина будь – якої гри, та досить гарний інструмент в руках гарного ігрового дизайнера. Пейсинг – це набір методик спрямованих на контроль уваги користувача підтримуючи відповідний темп оповідання.

В іграх доволі часто зустрічається чергування активностей, якими займається гравець протягом гри. Так, наприклад, після активної фази такої як сутичка з ботами втеча від когось, майже завжди на гравця чекає фаза спокою та навпаки. Оскільки ігри являються інтерактивним жанром на відміну від кіно, ігрові дизайнери не можуть тримати гравця в постійному напруженні, як це роблять в деяких картинах, бо у фільмі ми тільки спостерігаємо за головними героями, а в іграх змушені не тільки

спостерігати та співчувати їм а й керувати, реагувати на різні збуджувачі, що у великих кількостях може просто втомити користувача.

Стимуляція гравця поділяється на високорівневу та низькорівневу.

Високорівнева стимуляція – це спроби вплинути на емоціональну складову користувача, тобто змусити його відчути напруження, страх, сміх ті будь – які інші емоції необхідні для отримання.

Низькорівнева стимуляція – стосується механік гри, дизайну рівнів, перешкод на них, розташування ботів та інше. Якщо вірно користуватись низькорівневою стимуляцією то можна зекономити час на використання високрівневої. Так, наприклад в іграх від компанії FromSoftware розробники майже повністю зосередились на нисзькорівневій стимуляції гравця. Там немає чіткої історії, користувач не знає хто він, та до чого приведуть його дії. Завдяки правильному дизайну рівнів гравцю для проходження гри і не потрібно знати сюжету, він зрозуміє куди треба йти. Але попри сильному зосередженню на низькорівневій розробники не забули і про високорівневу стимуляцію. Так вони змушують гравця відчути страх перед новим босом через його дизайн та дизайн місця в якому він знаходиться. Для допитливих розробники створили історію для кожного предмету в грі. Тобто, для того, щоб повністю зрозуміти історію, користувачеві необхідно читати опис предметів та складати історію ігрового світу наче пазл.

Інші ігри пропонують гравцеві обрати свій темп самотужки. Тобто, на рівні розміщуються певні перешкоди та способи їх рішення. Єдиного вірного способу не існує, тільки користувач обирає, що для нього буде вірним. Наприклад, серія ігор Dishonored.

При проходженні першого рівня, на його початку розробники демонструють, що є декілька способів проходження (рис. 2.5).

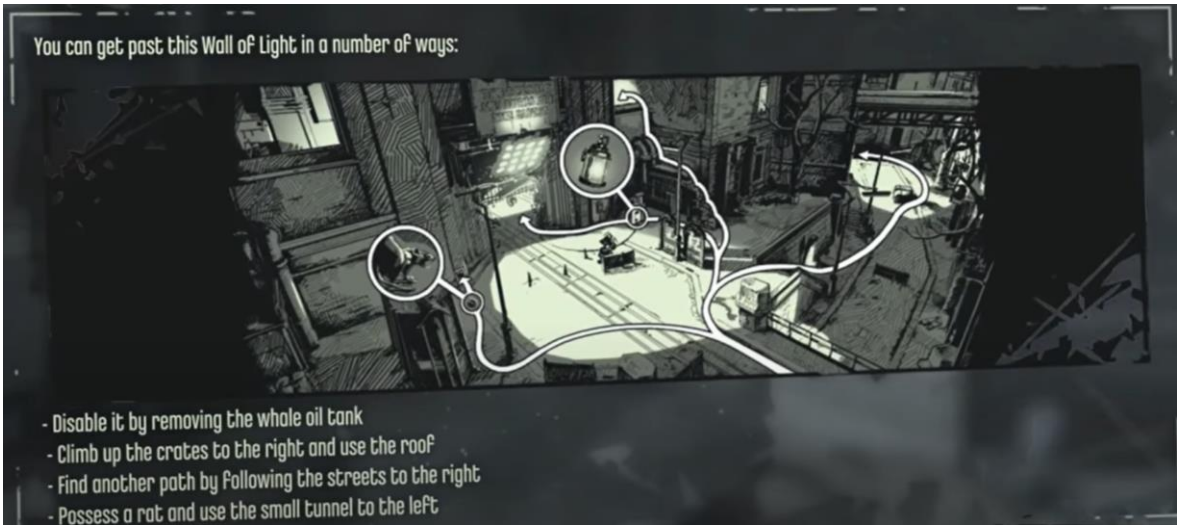

Рисунок 2.5 – Способи проходження рівню гри Dishonored

При роботі з проектами у відкритому світі розробники зійшлись на думці, що гравця, якщо він вирішить відійти від основних завдань та просто дослідити світ, потрібно чимось зацікавити, відовлікти. Через це в іграх такого типу кожні 40 – 80 секунд(залежить від проекту) на гравця будуть чекати нові події.

2.4 Планування першого рівню

Працюючи із жанром платформеру є декілька способів зберегти темп гри. Першим буде активна фаза із боями або втечею. Другим - фаза спокою із простим платформингом чи частинами із розповіддю історії. Третім – платформинг секціями, які будуть кидати виклик гравцеві.

Оскільки це перший рівень та перше знайомство гравця із грою – це найважливіша частина проекту оскільки він має зацікавити його. Тому було вирішено зберігати більш – менш активний темп за допомогою подій із простими боями та простими платформинг секціями. Повноцінні виклики кидати на гравця ще доволі рано, оскільки тут він тільки вчиться управлінню та механікам.

Висновки до другого розділу

У другому розділі було проведено опис систем притаманні гравцеві та ботам. Було описано цикл ігрового процесу та досліджено важливість ігрового дизайну при створенні ігрових застосунків. Здійснено планування першого рівню.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА ДОКУМЕНТАЦІЇ

3.1 Розробка механік гравця

Розробка ігрового застосунку розпочалась із розробки механік гравця. Потрібно було забезпечити наступну функціональність:

– звичайне переміщення(біг);

– стрибки із описаними вище особливостями(різна висота, буферизація, і. д.);

– можливість схопитись за стіну;

– можливість відстрибнути від стіни;

– ривок;

– атака.

Для більш зручної роботи було вирішено розбити функціонал гравця на три скрипта.

1) Player.

2) InputHandler.

3) AnimationController.

Player – містить в собі всі опис всіх методів, необхідних для переміщення гравця та його анімації.

InputHandler – зчитує ввід від користувача та викликає відповідний метод із срипта Player.

AnimationController – зчитуючи інформацію із InputHandler відтворює відповідну анімацію.

3.1.1 Впровадження переміщення

Було прийняте рішення розробити фізично достовірну систему переміщення гравця. Ігровий рушій Unity дозволяє виконати дану задачу у

декілька шляхів, а саме, за допомогою компонента Transform, або RigidBody. Перший забезпечує переміщення не враховуючи закони фізики, другий – враховуючи. Тож для роботи із фізичними функціями які є вже вбудованими необхідно було додати компонент Rigidbody 2D до створеного об’єкту Player. Також необхідно було додати компонент Capsule Collider 2D для того щоб об’єкт отримав фізичну форму певного геометричного тіла, у нашому випадку – це капсула. Маючи ці два компоненти можна розпочати створювати скрипти, які будуть відповідати за фунціонал гравця. В середовищі Unity вони також представляються у вигляді компонентів, але створені вони вже користувачем.

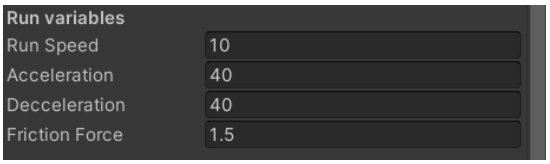

Рисунок 3.1 - Властивості компонента Player

За основні фізичні властивості бігу відповідають 4 змінні – Run Speed, Acceleration, Decceleration, Friction Force.

Run Speed – відповідає за швидкість руху.

Acceleration та *Decceleration* – відповідають за швидкість набору максимальної швидкості та швидкість зупинки.

Friction Force – імітує силу тертя.

Завдяки цьому було створено правдиву систему переміщення гравця.

3.1.2 Впровадження стрибків

Стрибок – найважливіший функціонал для платформеру, тож необхідно було створити його в найкращому виді.

Спочатку було написано звичайний функціонал стрибку за допомогою вбудованої функції RayCast – функція яка випускає вектор певної довжини та повертає true у випадку, якщо вектор у щось потрапив,

або false, якщо ні. Дана функція приймає декілька параметрів – origin, direction, distance, layerMask.

Origin – початкова точка вектору.

Direction – напрямок у якому буде направлено вектор.

Distance – довжина вектору.

LayerMask – з якими шарами буде стикатись вектор. В Unity кожен об’єкт можна додати до певного шару. До восьми завчасно створених рушієм або до власно створених. Потім їх можна використовувати в різних цілях. Так, наприклад, як для методів рейкастингу або для пошуку об’єктів на сцені в реальному часі за допомогою коду.

За допомогою рейкастингу вирішалась проста проблема, яка виникає при розробці стрибків, а саме нескінченних можливостях натискати на клавішу, яка відповідає за стрибок. Тобто тепер для стрибку гравцеві не просто необхідно натиснути на клавішу стрибку, а й бути на землі під час цього.

Разом із вирішенням однієї проблеми з’являється інша – завдяки доданим новим умовам стрибка користувачеві необхідно бути на землі. Але як показує досвід ігрової індустрії то в певних ситуаціях лише даний функціонал стрибка може призвести до того, що користувач буде вважати управління незручним. Наприклад, якщо гравець потрапляє на рівень де необхідно здійснити багато стрибків, і деякі мають бути виконаними відразу по приземленню, тоді виникає ситуація коли гравець може натиснути на клавішу занадто рано, або ж навпаки – занадто пізно.

Для цього додають ще одну перевірку до стрибків – *буферизацію*.

Завдяки цьому гравець отримує можливість натиснути клавішу стрибку трішки раніше ніж він торкнеться землі, що в свою чергу додає зручності управлінню.

Для покращення зручності управління також був доданий функціонал Coyote Time для декількох додаткових секунд, якщо користувач почав падати з платформи.

Буферизація та Coyote Time було реалізовано за допомогою звичайних таймерів.

Останнім додатковим функціоналом до стрибків була різна висота.

За допомогою нової змінної Jump Cut Multiplier висота стрибка в певний момент часу, коли гравець відпускав клавішу стрибка, починала зменшуватись.

3.1.3 Впровадження механік стрибків по стінам

Стрибки по стінам – механіка, яка додає варіативності до ігрового процесу. Для впровадження даного функціоналу було створено декілька нових змінних.

Wall Check – точка, додана до об’єкту Player, яка є початком вектору для методу RayCast. За допомогою даного методу йде перевірка наявності стіни перед гравцем.

Is Wall – змінна типу LayerMask, яка використовується для пошуку стіни, а саме вона перевіряє чи являється об’єкт перед гравцем стіною.

Wall Hop Force – сила з якою гравець впаде зі стіни у разі, якщо користувач не буде нічого далі робити.

Wall Jump Force – сила з якою гравець відстрибне від стіни, у разі натискання відповідної клавіші.

Wall Hang Time – кількість секунд які гравець буде знаходитись на стіні.

Рисунок 3.2 - Властивості компонента Player

3.2 Розробка графіки для гри

Графіка розроблялась паралельно розробці механік гри. Для даного проекту було розроблено:

- дизайн головного героя;
- дизайн ботів;
- анімації головного героя;
- анімації ботів;
- графіка для меню(задній фон, кнопки та їх анімації, лого);
- задній фон гри;
- шейдер туману;
- система частинок дощу.

Вся графічна частина розроблялась в декількох програмних забезпеченнях. Дизайн та анімації в Aseprite, фон в Adobe Photoshop.

3.2.1 Розробка графіки персонажів

Робота над графікою розпочалась із розробки дизайну персонажів.

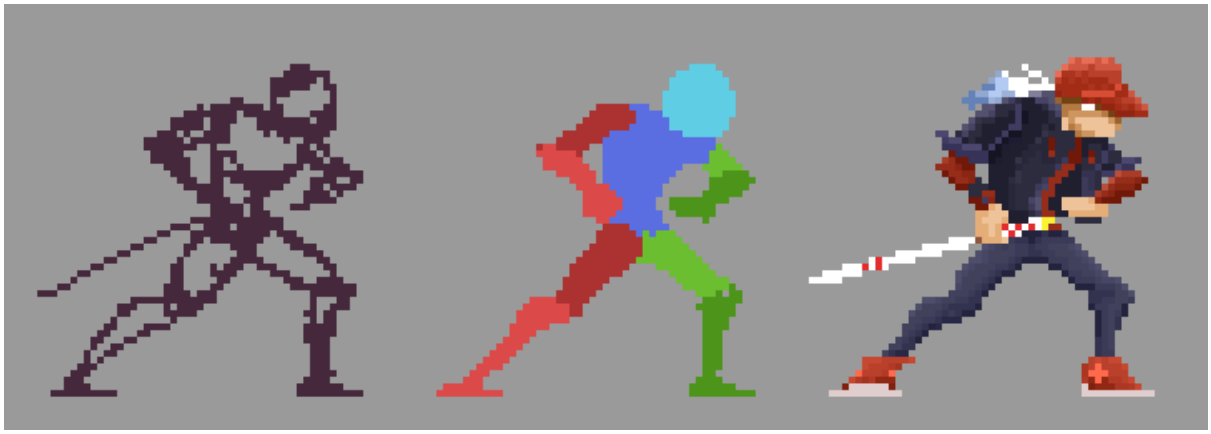

Рисунок 3.3 - Процес розробки дизайну головного персонажу

Розробка дизайну проводилась в три шляхи. Створення базової форми, створення макету та фінальний дизайн.

Макет буде використовуватись в подальшій розробці анімацій, оскільки в ньому тіло героя поділено на частини.

В дизайні головного героя присутні три основні кольори темно – синій, червоний, та білий. Ці кольори дозволяють вирізнити героя від фону.

Після створення дизайну проводилась робота над створенням анімацій. Анімація проводилась в програмі Aseprite.

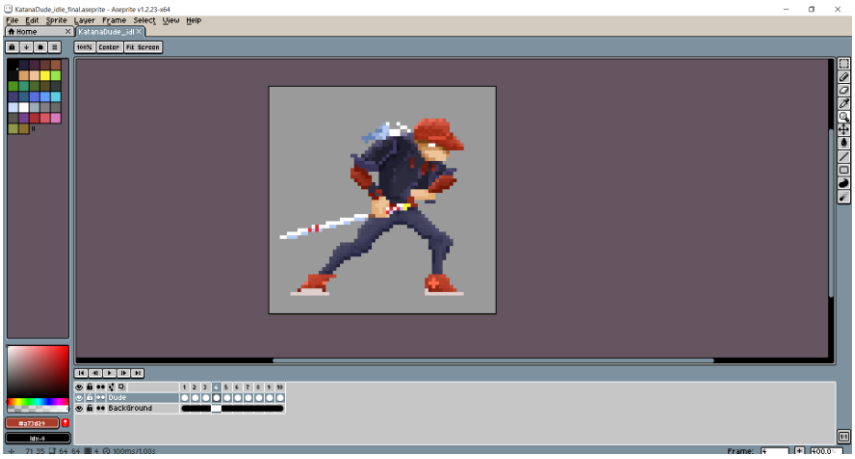

Рисунок 3.4 - Вікно програми Aseprite

Дане програмне забезпечення дозволяє створювати покадрову анімація та потім експортувати як спрайтовий лист, який буде далі використовуватись в ігровому рішії.

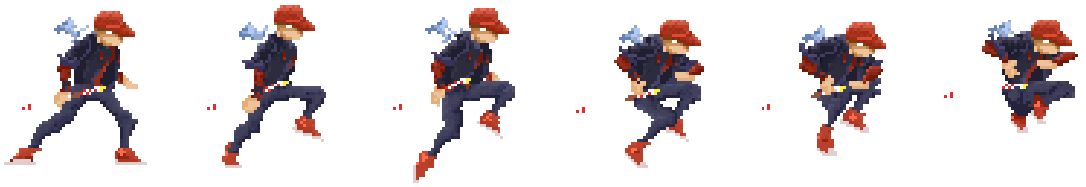

Рисунок 3.5 - Спрайтовий лист анімації стрибка

Було створено анімації:

– простою(idle);

– бігу;

– стрибку;

– знаходження на стіні;

– ривку;

– атаки;

– інші – частинок приземлення, стискання героя при приземленні.

Після створення спрайт листів із готових анімацій вони були імпортовані до проекту із відповідним налаштуванням.

Оскільки спрайт лист – це суцільне зображення його необхідно було нарізати на відповідні кадри, щоб анімація працювала вірно. До цього необхідно було вказати, що даний спрайт лист містить в собі декілька зображень, вказати відповідну фільтрацію та максимальний розмір файлу для того щоб при використанні не з’явилось неочікуваних артефактів.

За схожою схемою було створено та анімовано інших персонажів, а саме – повітряного дрону, та наземного роботу.

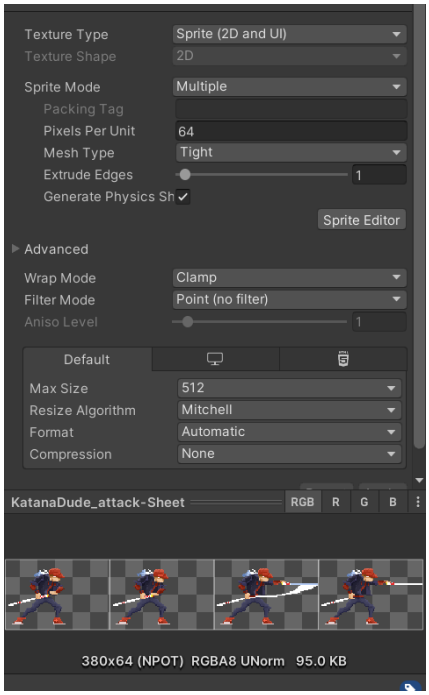

Рисунок 3.6 - Налаштування спрайтового листа

3.2.2 Розробка графіки фону та головного меню

Графіка фону проводилась у програмі Photoshop.

Спочатку було створено базовий рівень деталізації (рис. 3.7) потім проводилась робота над додаванням додаткових деталей та кольору до роботи.

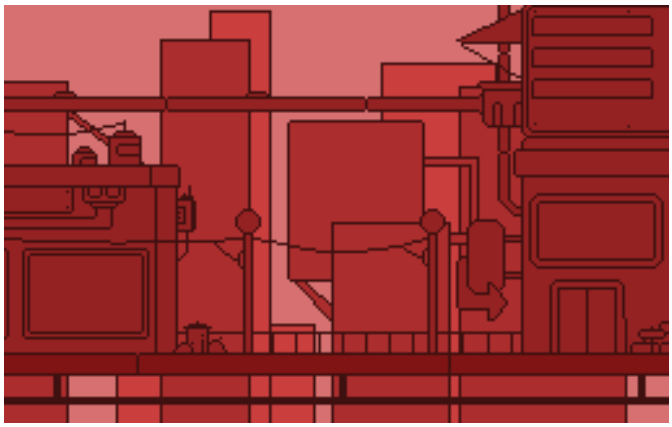

Рисунок 3.7 - Базовий рівень деталізації першого рівня

Коли фон було завершено він був також імпортований до проекту.

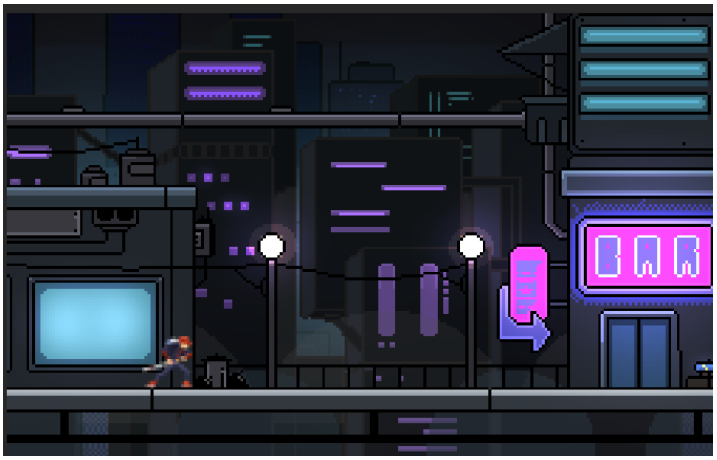

Рисунок 3.8 - Фінальний вигляд першого рівня

Після розробки фону для першого рівня проводилась робота над графікою для головного меню.

Для розробки використовувались прийоми малювання такі як перспектива, контрастність форм та кольору, знання форм об’єктів.

Рисунок 3.9 - Дизайн заднього фону для головного меню

Після головного меню було розроблено дизайн лого для гри у ретро стилі.

3.3 Розробка штучного інтелекту для ботів

3.3.1 ШІ для ботів – дронів

Типів повітряних ботів два, але відрізняються вони лише типом атаки. Так перший тип веде ближній бій, а другий – дальній.

Для створення повітряного бота потрібно було створити скрипти які відповідають, за пошук гравця на сцені, прокладання шляху до нього та здійснення відповідного переміщення. Для того щоб дрон знаходився на тому місці, де він має бути було додано ще один компонент який відповідає за зону активації та деактивації боту.

Пошук шляху здійснювався за допомогою алгоритму A*. Unity вже має вбудований пакет із даним алгоритмом.

Встановивши пакет A* з’являється можливість розбити сцену на сітку.

Для цього потрібно створити новий об’єкт та додати до нього компонент A*. Далі створити нову сітку (рис. 3.10), налаштувати її розмір та додати перешкоди, якщо вони є.

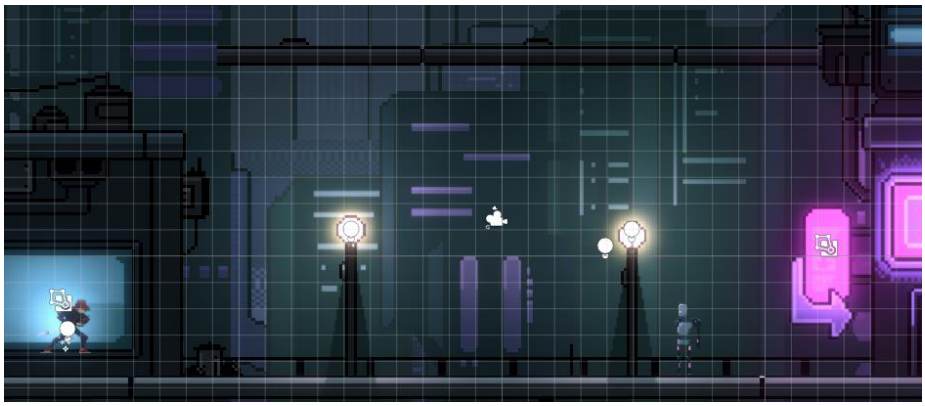

Рисунок 3.10 - Сітка A*

Бот буде здійснювати переміщення по даній сітці, тут також можна вказати перешкоди, від місцезнаходження яких алгоритм буде формувати відповідний шлях до цілі.

Для того, щоб дрон почав користуватись сіткою створеною за допомогою компонента A* потрібно до нього додати декілька нових – це Seeker, AIPath, AI Destination Setter.

Головним компонентом тут є AIPath. В ньому присутні налаштування зони пошуку нової точки, уповільнення та відстані, на якій має бути об’єкт до цілі, щоб це вважалось за пройдений шлях.

Також тут є можливість задати фізичне тіло для об’єкту та швидкість із якою він буде рухатись.

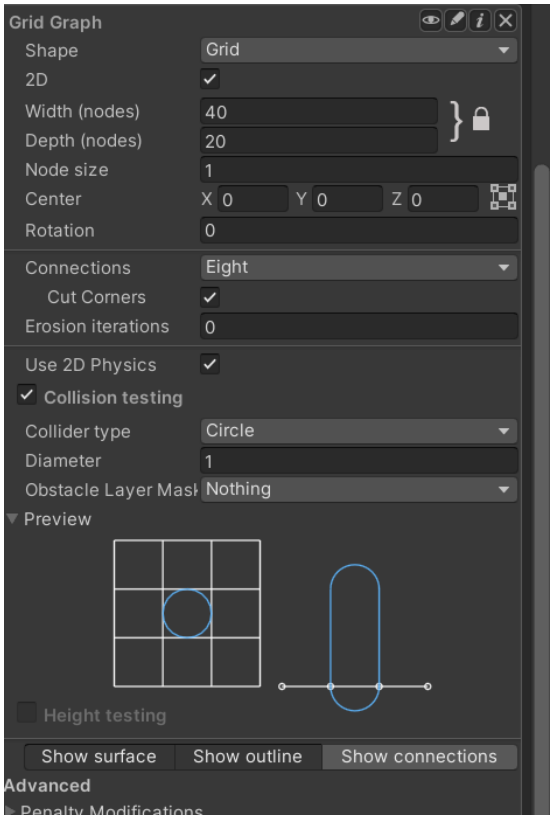

Рисунок 3.11 - Налаштування компоненту A*

Для того, щоб бот не тільки слідував за гравцем, а і мав можливість атакувати його було створено новий скрипт DroneShoot.

```
private void CheckIfCanShoot()
{
    shootingCounter -= Time.deltaTime;

    if (shootingCounter <= 0f)
    {
        Shoot();
        shootingCounter = shootingSpeed;
    }
}


private void Shoot()
{
    GameObject bullet = Instantiate(projectile, shootingPoint.position, Quaternion.identity);

    Vector2 destination = (target.position - transform.position).normalized;
    bullet.GetComponent<Rigidbody2D>().velocity = destination * bulletSpeed;

    Destroy(bullet, 2);

    Debug.Log("Shooting!");
}
```

Рисунок 3.12 - Лістинг коду DroneShoot

Тут є головними дві функції: CheckIfCanShoot та Shoot.

Перша відповідає за перевірку можливості здійснення пострілу та контролює проміжок із яким проводяться постріли. Окрім цього тут також можна задати швидкість польоту пулі, ціль в яку має буди здійснений постріл та точку із якої його здійснити. Для точки здійснення пострілу до головного об’єкту RangedDrone був доданий дочірній об’єкт ShootingPoint. Рушій дає змогу додавати дочірні об’єкти до вже створених. Дочірні об’єкти повторюють переміщення та кручення батьківського.

У функції Shoot йде створення нового об’єкту на сцені, який є пулею. Потім визначається напрям типу Vector2, у якому має здійснюватись постріл. Оскільки вектор може бути різної довжини оскілки гравець та дрон знаходять у постійному русі необхідно було забезпечити сталу довжину вектора. Це здійснюється за допомогою функції normalized. Після вичислення направлення йде звернення до компоненту пулі RigidBody та задається певна швидкість, із якою вона буде рухатись. Якщо пуля потрапить у гравця вона знищеться, якщо ні – знищення відбудеться через дві секунди.

Для того щоб бот не слідував за гравцем через весь рівень, а залишався в певній його області був доданий ще один скрипт Awake Sleep.

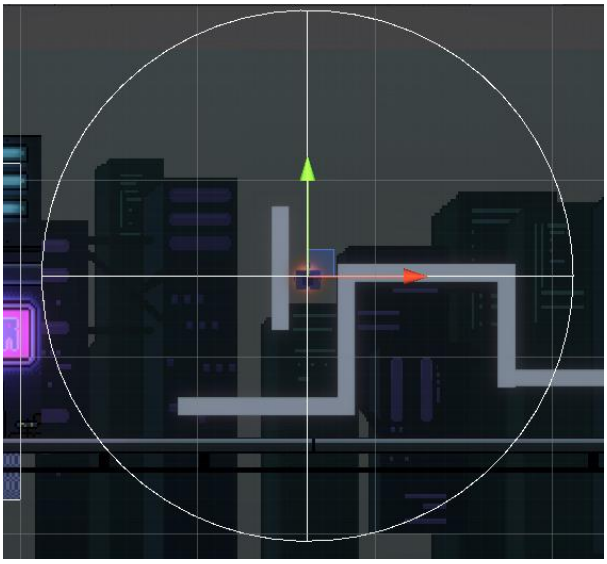

Рисунок 3.13 - Зона активації дрону

Він відповідає за утримання бота в певній зоні. Коли гравець потрапляє в зону активації бота вмикаються скрипти відповідні за пошук шляху та бій, коли виходить – вимикаються і дрон знаходиться в пасивному стані, скануючи зону на наявність в ній гравця.

За схожим алгоритмом був створений другий тип – дрон ближнього бою.

3.3.2 ШІ для наземних ботів

Наземні боти створювались без використання алгоритмів пошуку шляху. Їхній сценарій поведінки полягав у патрулюванні певної області та, у разі потрапляння у цю область, почати атакувати гравця. Якщо ж гравець виходив із цієї зони – продовжити патрулювання.

Для цього скрипти для наземного бота були поділені на відповідні стани.

1) Patrol State.
2) Combat State.

3) Robot State Controller.

Patrol State – стан, у якому знаходиться бот у разі патрулювання. Даний скрипт запам’ятовує початкову точку у якій бот заходився на старті сцени. Дана точка буде вважатись правою границею його зони патрулювання. Вказавши відстань патрулювання, йде розрахунок лівої границі – так отримується область роботи бота. Також, окрім відстані, можна задати швидкість переміщення.

Combat State – стан, у якому буде знаходитись бот у разі потрапляння гравця в його зону. Даний компонент відповідає за пошук відстані від гравця до бота, пошук кута між ними для того, щоб бот мав можливість постійно слідкувати за гравцем. Наприклад, якщо гравець активує бота та змінить свої місце положення робот має знайти його та повернутись у відповідному напрямку для здійснення пострілу.

Robot State Controller – компонент – контроллер станів бота. В ньому міститься інформація про зону активації, доступні стани та умови їх переходів.

Здійснення пострілу проводиться в два етапи – перший перевірка наявності грацвя на лінії вогню, другий – здійснення пострілу. Для цього було створено декілька методів у компоненті Combat State. CheckHit, який відповідає за перевірку потрапляння гравця у зону ураження. Перевірка здійснюється за допомогою рейкастингу. Оскільки тип стрільби робота оснований не на проджектайлах а на промені. Далі за допомогою вбудованого компонента Animator до анімації пострілу була додана подія(Animation Event). Анімаційні події дають змогу викликати функцію об’єкта, до якого відноситься анімація. Так можна обрати будь який кадр анімації та додати виклик певної функції. У нашому випадку це виклик функції пострілу із скрипта Combat State. Це спрощує розробку деякої

логіки проекту, оскільки хникає необхідність створювати таймери або ж додаткові умови для спрацювання деяких методів.

3.4 Створення головного меню

Для створення головного меню необхідно було виконати декілька кроків. Створити кінематографічну камеру, створити кнопки та їх анімації та лого гри.

Для створення кінематографічної камери було створено новий скрипт та додано до об’єкту, який відповідає за її поведінку. Даний компонент дає змогу виставити верхню та нижню границю для переміщення камери. Маючи границі камера почне рухатись у довільному напрямку імітуючи легке трясіння нею.

Далі, використовуючи Photoshop, був створений приблизний вид головного меню. Після цього дане зображення було нарізане на окремі частини та імпортовано до проекту.

В ігровому рушію провелось налаштування імпортованих зображень та вони були додані до сцени, як дочірні об’єкти до Canvas. Даний компонент дозволяє більш точно налаштувати розмір зображення. А саме його масштаб при різних розширеннях екрану. Наприклад, якщо дана гра запуститься на смартфоні або ПК то малюнок залишиться в тому масштабі в якому він був.

Після розташування об’єктів як це було задумано було створено новий префаб. А саме префаб кнопки. Оскільки вона створювалась не за допомогою стандартних інструментів Unity, а була створена повністю самостійно простіше було створити єдиний екземпляр для подальшої роботи з ним.

Далі була проведена робота з анімаціями кнопок. Були створені анімаційні кліпи та дерево анімацій для наступних станів кнопки:

– кнопка є обраною;

– на кнопку натиснули;

– кнопка не активна.

Рисунок 3.14 - Дерево анімацій

Також для більшої зручності та зрозумілості головного меню було знайдено та додано відповідні звуки перемикання між кнопками та натискання на них.

По задумці на фоні в головному меню має йти дощ та бути присутнім туман. Тому тут проводилась додаткова робота із системою частинок та шейдерами.

Шейдер – підпрограма, в якій описані додаткові фізичні властивості об’єкту. Наприклад, як сильно він розсіює світло, якого типу в нього поверхня та інше.

Для створення ефекту дощу була використана система частинок Unity. Провівши певні маніпуляції в компоненті Particle System було отримано потрібний ефект дощу.

Ефект туману був створений за допомогою шейдеру. Шейдери являються частиною пакета Universal Render Pipeline, тож для їхньої коректної роботи він був встановлений. Даний пакет також дає змогу працювати із постпроцесингом для 2D проектів. Наприклад, додати джерело світлу, яке б впливало на додані спрайти, тобто підсвічувало їх та змушувало відкинути тінь.

Після встановлення нового пакету з’явилась можливість створювати шейдери, завдяки чому був створений шейдер туману.Тут присутні налаштування швидкості туману, густоти, кольору, рівню пікселизації, оскільки проект створювався в піксель – арт стилі.

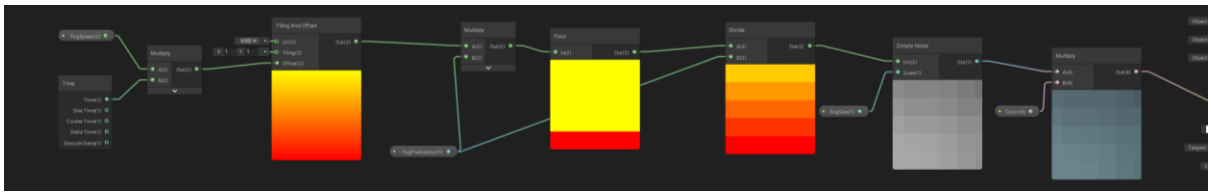

Рисунок 3.15 - Структура шейдеру туману

Далі було створено відповідний матеріал, йому було присвоєно даний шейдер. Після цих кроків можна було додавати його на сцену.

Після виконання всіх кроків був отриманий фінальний вигляд головного меню.

Рисунок 3.16 - Фінальний вигляд головного меню

3.5 Робота з постпроцесингом

Додавання пакету URP до проекту дало можливість працювати із постпроцесингом для 2D проектів.

Постпроцесинг – сукупність усіх додаткових ефектів, які працюють на удосконалення уже завчасно створеного зображення. Це можуть бути додаткові тіні, джерела світла та різноманітні ефекти пов’язані з ним.

Для роботи із постпроцесингом необхідно створити новий об’єкт Global Volume. У ньому буде присутній компонент Volume, який відповідає за ефекти на сцені.

Провівши роботу із даним компонентом було додано ефекти Bloom та Vignete. Bloom відповідає за розсіювання світла, яке відбивається від об’єктів. Vignete – для створення відчуття нічної атмосфери шляхом затінення відповідних частин сцени.

Дані налаштування поширюються на сцену меню та сцену гри.

Для того щоб об’єкти відкидували тінь при спрямуванні світла на них до даних об’єктів необхідно було додати новий компонент Shadow Caster 2D. Даний компонент забезпечує створення тіні відповідної форми. Так до об’єкту Player був доданий даний компонент та створена відповідна форма.

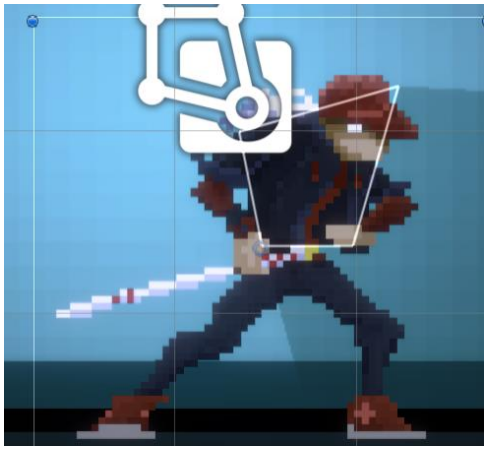

Рисунок 3.17 - Форма компонента Shadow Caster 2D

Висновки до третього розділу

У даному розділі було описано процес розробки ігрового застосунку. Був описаний процес розробки механік для гравця, графічної частини та

анімацій, розробки штучного інтелекту для різного типу ботів, створення головного меню.

Спеціальний розділ

ОХОРОНА ПРАЦІ

до кваліфікаційної роботи

на тему:

«Розробка 2D – гри на платформі Unity»

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810325

Виконав студент 4-го курсу, групи 402
_____*І.І. Чернов*
(підпис, ініціали та прізвище)
«__» _____ 2022 р.

Консультант *старший викладач*
(наук. ступінь, вчене звання)
_____*О.В. Макарова*
(підпис, ініціали та прізвище)
«__» _____ 2022 р.

Миколаїв – 2022

4 ОЦІНКА УМОВ ПРАЦІ В КОМП'ЮТЕРНІЙ АУДИТОРІЇ ЧНУ ІМ. ПЕТРА МОГИЛИ

У даному розділі роботи розглянуто питання охорони праці у комп'ютерній аудиторії ЧНУ імені Петра Могили, в якій виконувалось тестування розробленого програмного застосунку. Також виконана інтегральна оцінка умов праці та запропоновані заходи, спрямовані на їх покращення.

4.1 Опис обраного виробничого приміщення, робочих місць, їх обладнання та складання вихідних даних для кількісної оцінки умов праці

Приміщення комп'ютерної аудиторії ЧНУ імені Петра Могили розташовано на третьому поверсі чотириповерхової будівлі, що знаходиться у м. Миколаєві на вулиці Десантників, 68. Розміри приміщення складають $a \times b \times H = 12{,}0 \times 8{,}0 \times 4{,}0$ м. У приміщенні влаштовано п'ять металопластикових вікон (з подвійними склопакетами) розмірами $c \times d = 1{,}2 \times 2{,}4$ м. Приміщення зовнішньою стіною спрямовано за азимутом в діапазоні Аз =136…225 град.

Приміщення має доволі сучасний офісний інтер'єр. Стеля виконана у вигляді підвісної конструкції із синтетичного матеріалу, та складається з окремих квадратних плиток світло-сірого кольору. Стіни оштукатурені, покриті фарбою зеленого кольору. Підлога покрита лінолеумом, що імітує паркет коричневого кольору.

Вікна обладнані світлозахисними пристроями у вигляді горизонтальних регульованих жалюзі.

У приміщенні розташовано 26 робочих місць. всі обладнані робочими стаціонарними комп'ютерами під управлінням ОС Windows. Для доступу кожного з комп'ютерів до локальної обчислювальної мережі на кожні 2 робочих місця розраховано подвійну телекомунікаційну розетку.

Для підключення всього обладнання, використовуються мережеві фільтри. За дозволом адміністрації, іноді використовуються кондиціонер або телевізор. Робочі місця розділені між собою перегородками.

Загальний вид обраного виробничого приміщення представлено на рис.4.1.

Рисунок 4.1 — Загальний вигляд обраного виробничого приміщення

Напруга джерела живлення електроспоживної техніки – 220 В. Електромережа виконана у вигляді трипровідної з дотриманням усіх вимог нормативних документів. За небезпекою ураження електричним струмом приміщення відноситься до приміщень без підвищеної небезпеки ураження електричним струмом.

Мікрокліматичні умови у літній період (частково у перехідний) забезпечується двома кондиціонерами, потужності яких не завжди вистачає для забезпечення комфортних умов праці, з огляду на кліматичні умови в м. Миколаїв. У зимовий період опалення здійснюється центральною системою, яка також не завжди забезпечує необхідний тепловий режим.

Деякий дискомфорт, пов'язаний із виробничим освітленням, може відчуватися у певні часи робочого дня.

Пожежна безпека в обраному виробничому приміщенні забезпечується дотриманням вимог НПАОП 0.00-1.28-10 [4].

За даними контрольних обстежень, виконанням необхідних вимірів за допомогою відповідного обладнання, а саме: термометру(СТР5000) для вимірювання температури повітря, гігрометру(ВИТ-1) – для вологості, анемометру(GM816) для вимірювання швидкості руху повітря, люксометра(S400B) – для освітленості, а також експертних оцінок здійснена оцінка умов праці в обраному виробничому приміщенні (табл. 4.1). Наведені у табл. 4.1 відомості являються вихідними даними для виконання подальших розрахункових робіт даного розділу дипломної роботи бакалавра.

Таблиця 4.1 — Фактори умов праці у відділі розробки програмного забезпечення

№ з/п	**Фактори умов праці на робочому місці**	**Значення показника**	**Тривалість дії фактору, хв.**
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, ℃: - теплий період - холодний період	 27 20	 420 420
2	Відносна вологість повітря на РМ, %	55	420
3	Швидкість руху повітря на РМ, м/с	0,13	420
4	Освітленість на РМ, лк	220	420
5	Мінімальний розмір об’єкта розпізнавання, мм	0,5	360
6	Виробничий шум, дБА	0,7	420
7	Інтенсивність теплового випромінювання, Вт/м²	130	420
8	Токсична речовина, озон, кратність перевищення гранично допустимої концентрації	-	420
9	Виробничий пил (паперовий та ін.), кратність перевищення гранично допустимої концентрації	-	420

10	Робоче місце (РМ), поза та переміщення у просторі	РМ стаціонарне, маса переміщення вантажу $\leq$ 5 кг	420

Продовження табл. 4.1.

11	Кількість важливих об’єктів, що спостерігаються при роботі	2	420
12	Тривалість зосередженого спостереження, % від загального часу тривалості робочої зміни	80	384
13	Тривалість повторюваних операцій, с	120	384
14	Змінність роботи	Ранкова зміна	480
15	Тривалість безперервної роботи за добу, годин	7	420
16	Режим праці та відпочинку	Обґрунтований з вкл.. музики та гімнастики	480
17	Нервово-емоційне навантаження	Складні дії за заданим планом при дефіциті часу	420
18	Кількість рухів пальців на годину	3000	384
19	Монотонність, тривалість операцій, які повторюються , с	10	384

4.2 Інтегральна оцінка умов праці в обраному виробничому приміщенні

Для інтегральної оцінки умов праці [1] в обраному виробничому приміщенні скористаємося даними табл. 4.1 та здійснимо оцінку питомої ваги кожного із представлених там факторів виробничого середовища та трудового процесу.

У табл. 4.2 представлені параметри, що необхідні для інтегральної оцінки умов праці:

- $x_{н_i}$ – нормативне значення i – того фактору умов праці;
- $x_{аб_i}$ – дійсне значення i – того фактору умов праці;
- x_{x_i} – оцінка i – того фактору умов праці, балів;
- t_i – тривалість дії i – того фактору умов праці, хв.;
- $t_{пит_i}$ – відносна тривалість дії i – того фактору умов праці хв., тобто:

$$t_{\text{пит}_\text{і}} = \frac{t_i}{t_p} = \frac{t_i}{480};$$

– x_{ϕ_i} – фактична оцінка питомої ваги i – того фактору умов праці:

$$x_{\text{ф}_\text{i}} = x_{x_i} t_{\text{пит}_\text{i}} = x_{x_i} \frac{t_i}{480}.$$

За даними табл. 4.2 та 4.3 визначаємо елемент умов праці, який одержав у балах найбільшу оцінку $x_{\max}$. Таких елементів може бути декілька.

Таблиця 4.2 — Параметри, що необхідні для розрахунку інтегральної бальної оцінки умов праці на робочому місці

№ з/п	Фактор умов праці на робочому місці	Нормоване значення фактора $x_{\text{Н}_\text{j}}$	Абсолютна оцінка $x_{\text{абі}}$	Абсолютна оцінка у балах $x_{\text{хі}}$
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, ℃ - теплий період - холодний період	 23…25 21…23	 26 21	 2 1
2	Відносна вологість повітря на РМ, %	40..60	55	2
3	Швидкість руху повітря на РМ, м/с	<0,2	0,13	1
4	Освітленість на РМ, лк	200	220	3
5	Мінімальний розмір об’єкта розпізнавання, мм	>1	0,5	2
6	Виробничий шум, дБА	50	0,7	1
7	Інтенсивність теплового випромінювання, Вт/м²	≤140	130	1
8	Токсична речовина, озон, кратність перевищення ГДК	≤1	-	1
9	Виробничий пил (паперовий), кратність перевищення гранично допустимої концентрації	≤1	-	1
10	Робоче місце (РМ), поза та переміщення у просторі	РМ стаціонарне,маса переміщення до 5 кг	РМ стаціонарне,маса переміщення до 5 кг	1
11	Кількість важливих об’єктів спостереження	<5	2	1

12	Тривалість зосередженого спостереження, % зміни	<25	80	4

Продовження табл. 4.2

13	Тривалість повторюваних операцій, с	>100	120	1
14	Змінність роботи	Ранкова	Ранкова	1
15	Тривалість безперервної роботи за добу, годин	<8	7	2
16	Режим праці та відпочинку	Обґрунтований з вкл.. музики та гімнастики	Обґрунтований з вкл.. музики та гімнастики	1
17	Нервово-емоційне навантаження	Прості дії за індивідуальним планом	Виконання складних дій за заданим планом при дефіциті часу	4
18	Кількість рухів пальців на годину	<360	3000	4
19	Монотонність, тривалість операцій, які повторюються , с	>100	10	4

Таблиця 4.3 — Додаткові параметри, що необхідні для розрахунку інтегральної бальної оцінки умов праці на робочому місці

№ з/п	**Фактор умов праці на робочому місці**	**Тривалість дії фактора t_i**	**Тривалість дії фактора у долях робочої зміни, $t_{пит\ i}$**	**Фактична оцінка питомої ваги фактора $x_{фi}$**
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, ℃ - теплий період - холодний період	 420 420	 0,875 0,875	 1,75 0,875
2	Відносна вологість повітря на РМ, %	420	0,875	1,75
3	Швидкість руху повітря на РМ, м/с	420	0,875	0,875
4	Освітленість на РМ, лк	420	0,875	2,625
5	Мінімальний розмір об’єкта, мм	360	0,75	1,5
6	Виробничий шум, дБА	420	0,875	0,875

7	Інтенсивність теплового випромінювання, Вт/м²	420	0,875	0,875
8	Токс. речовина, кратність перевищення ГДК	420	0,875	0,875

Продовження табл. 4.3

9	Виробничий пил (паперовий), кратність перевищення ГДК	420	0,875	0,875
10	Робоче місце (РМ), поза та переміщення у просторі	420	0,875	0,875
11	Кількість важливих об'єктів спостереження	420	0,875	0,875
12	Тривалість зосередженого спостереження , у % часу зміни	384	0,8	3,2
13	Тривалість повторюваних операцій, с	384	0,8	0,8
14	Змінність роботи	480	1	1
15	Тривалість безперервної роботи за добу, годин	420	0,875	1,75
16	Режим праці та відпочинку	480	1	1
17	Нервово-емоційне навантаження	420	0,875	3,5
18	Кількість рухів пальців на годину	384	0,8	3,2
19	Монотонність, тривалість операцій, які повторюються , с	384	0,8	3,2

За даними таблиці 4.3, визначимо елемент x_{max}, який набрав найбільшу оцінку. Таким елементом є елемент x_{17}, який пов'язаний із нервово-емоційним навантаженням на робочому місці.

Розрахуємо середнє арифметичне фактичних оцінок питомої ваги факторів (всіх, окрім визначаючого елемента) за формулою

$$\bar{x} = \frac{\sum_{i=1}^{n-1} x_{\text{ф}i}}{n-1}, \qquad (4.1)$$

де $n = 19$.

В даному випадку, $\bar{x} = 1{,}51$.

Далі, розрахуємо інтегральну оцінку умов праці на робочому місці у відділі розробки програмного забезпечення за формулою

$$I_n = 10 * (x_{max} + \bar{x}\frac{6-x_{max}}{6}). \quad (4.2)$$

В даному випадку, I_n = 41,29.

Порівнявши отримане значення зі значеннями, наведеними в Додатку Д, можна зробити висновок, що умови праці на визначеному робочому місці відносяться до III категорії — Роботи, що відхиляються від ГДК і ГДР та допустимих рівнів психофізіологічних факторів.

Пропонується для деяких факторів, що отримали оцінку, вищу за 2, застосувати заходи для покращення умов праці, а саме:

– Фактор 4 — освітленість на РМ: встановлення більш потужних освітлювальних пристроїв;

– Фактор 12 — тривалість зосередженого спостереження: запровадження іншого режиму роботи, з більшою кількістю (тривалістю) перерв;

– Фактор 17 — нервово-емоційне навантаження: запровадження інших моделей менеджменту проектів, з більш ретельним плануванням.

Фактори 18 та 19 (кількість рухів пальців на годину, та тривалість операцій, які повторюються), не може бути змінена, оскільки ці фактори напряму пов’язані зі специфікою роботи програміста.

Припустимо, що значення оцінок оптимізованих факторів дорівнюють 2. Обчислимо інтегральний показник важкості праці за формулою

$$I_{n_2} = 19{,}7 * a - 1{,}6 * a^2, \quad (4.3)$$

де $a = \sum_{i=1}^{n}\frac{x_i}{n}$.

Для оптимізованих умов праці, інтегральний показник важкості праці дорівнює 25,16. Відповідно до даних, наведених у Додатку Д, цей показник відповідає категорії II — роботам, що виконуються в умовах, які відповідають гранично допустимим концентраціям (ГДК) і рівням (ГДР) санітарно-гігієнічних елементів, а також допустимим рівням психофізіологічних факторів.

Визначимо ступінь втоми працівників до та після прийнятих заходів за формулою 4.4:

$$B = \frac{I_n - 15,6}{0,64} \quad (4.4)$$

До прийнятих заходів, ступінь втоми працівників становив 40,14. Після прийнятих заходів, ступінь втоми працівників становив 14,94.

Розрахуємо також ступінь працездатності людини за формулою

$$\text{П} = 100 - \text{В} \quad (4.5)$$

До впровадження заходів з охорони праці, ступінь працездатності працівників становив 59,86. Після впровадження заходів з охорони праці, ступінь працездатності працівників становив вже 85,06.

Вирахуємо зміну продуктивності праці за формулою

$$\Delta\text{П} = 0,2 * \left(\frac{\text{П}_2}{\text{П}_1} - 1\right) * 100 \quad (4.6)$$

Зміна продуктивності праці становить 8,42%.

Виконані розрахунки довели, що проведені заходи з охорони праці призвели до зменшення важкості праці з III до II категорії, і відповідно, зниженню втоми, підвищенню працездатності працівників відділу програмного забезпечення управління слабоформалізованими системами.

Крім того, вказані заходи можуть призвести до підвищення продуктивності праці співробітників відділу на 8,42 %.

Висновки до четвертого розділу

В ході виконання спеціальної частини з охорони праці до дипломної роботи на тему «Розробка 2D – гри на платформі Unity» проаналізовано умови праці у аудиторії ЧНУ імені Петра Могили, де проводилася частина розробки та тестування розробленого застосунку.

Згідно з розрахунками, інтегральна оцінка умов праці у відділі становила 41,29. Умови праці на визначеному робочому місці відносились до III категорії - Роботи, що відхиляються від ГДК і ГДР та допустимих рівнів психофізіологічних факторів.

Запропоновано впровадити такі заходи для покращення умов праці у відділі:

– встановлення більш потужних освітлювальних пристроїв;
– запровадження іншого режиму роботи, з більшою кількістю (тривалістю) перерв;
– запровадження інших моделей менеджменту проектів, з більш ретельним плануванням.

За попередніми розрахунками, після впровадження вищенаведених заходів, інтегральний показник важкості праці дорівнюватиме 25,16. Відповідно до даних, наведених у Додатку Б, цей показник відповідатиме категорії II — роботам, що виконуються в умовах, які відповідають гранично допустимим концентраціям (ГДК) і рівням (ГДР) санітарно-гігієнічних елементів, а також допустимим рівням психофізіологічних факторів.

Запропоновані заходи також можуть привести до значного зниження ступеню втоми працівників та підвищення продуктивності їхньої роботи на 8,42%.

ВИСНОВКИ

В ході виконання дипломної роботи були досягнуті наступні цілі:

– проведений аналіз жанру платформеру та доступних аналогів;

– досліджені способи та інструменти створення графічної частини та анімацій для відеоігор;

– проаналізовані доступні ігрові рушії, визначено їх переваги та недоліки;

– досліджено способи використання ШІ в відеоіграх;

– розроблено механіки та графічну частину гри, імплементовано алгоритми штучного інтелекту.

При аналізі жанру платформеру було виділено особливості та окремі унікальні механіки притаманні даному жанру. А саме – вид збоку, велика увага механікам стрибка та постійні виклики гравцеві.

Було проаналізовано способи та інструменти створення графічної частини для відеоігор. Досліджені наступне програмне забезпечення.

1) Photoshop – графічний редактор який дає змогу працювати із зображеннями різних розмірів у тому числі і з піксель – артом але не досить зручний при роботі із зображеннями низького розширення.

2) Aseprite – графічний редактор, спеціально розроблений для роботи із піксельною графікою. Окрім роботи із зображеннями є також можливість роботи із анімаціями.

Для вибору ігрового рушія було проведено аналіз існуючих варіантів, їхні переваги та недоліки. Серед розглянутих рушіїв є Unreal Engine від Epic Games, Unity та GameMaker: Studio. Обраним рушієм став

Unity через його доступність гнучкість та велику кількість навчального матеріалу.

Для написання коду до ігрового застосунку використовувалась мова програмування C#.

Для розробки схем та діаграм використовувався сайт app.diagrams.net. було розроблено схеми зв’язку між класами об’єкта гравця, стани у якому можуть перебувати гравець та боти.

При аналізі алгоритмів штучного інтелекту було досліджено роботу алгоритмів Дейкстри та алгоритму A*.

Для розробки штучного інтелекту для ботів використовувався пакет A* від Unity. Для окремих типів ботів були розроблені окремі сценарії поведінки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wolf M. J. P. Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming / Ed. by Mark J. P. Wolf. Santa Barbara, CA : Greenwood, 2012. 740 с.
2. Донован Т. Грай! Історія видеоігор. М., 2014.
3. Глибовець М. М., Олецький О.В. Штучний інтелект. Київ : «Києво-Могилянська академія», 2002. 364 с.
4. Методи штучного інтелекту : навч. посіб. / В. Б. Гітіс, К. Ю. Гудкова. Краматорськ: ДДМА, 2018. 136 с.
5. Системи штучного інтелекту : навч. посіб. / Н. Б. Шаховська, Р. М. Камінський, О. Б. Вовк. Львів : Львівська політехніка, 2018. 392 с.
6. Stuart J. Russell, Peter Norvig. Artificial Intelligence: A Modern Approach. 3. Pearson, 2015.
7. Bhadeshia H. K. D. H. Neural Networks in Materials Science : ISIJ International 39, 1999. (10): 966–979 с.
8. Duda, Richard O.; Hart, Peter Elliot; Stork, David G. Pattern classification. Wiley, 2001.
9. Bourg; Seemann. AI for Game Developers, 2004. O'Reilly & Associates.
10. Buckland. Programming Game AI By Example, 2004.
11. Funge. AI for Animation and Games: A Cognitive Modeling Approach. A K Peters, 1999.
12. Офіційний сайт Unity3D : вебсайт. URL: https://unity3d.com/ru (дата звернення: 28.04.2022).
13. Офіційний сайт Visual Studio : вебсайт. URL: https://www.visualstudio.com/ru/vs/ (дата звернення: 10.05.2022).
14. Gold J. Object-Oriented Game Development. UK : Pearson Education Limited, 2004. 404 с.

15. Gregory J. Game Engine Architecture. USA : A K Peters / CRC Press, 2009. – 864 с.
16. Unity Manual, Unity Documentation : довідник. URL: https://docs.unity3d.com/Manual/ (дата звернення: 12.05.2022).
17. Swink S. Game Feel: A Game Designer's Guide to Virtual Sensation : ISBN – 13, 2009.
18. Christopher W. T. An architectural approach to level design : George Mason University, 2014.
19. Jared H. Developing 2D Games with Unity: Independent Game Programming with C# : ISBN – 13, 2009.
20. Scott R. Level up! The guide to great video game design, 2010.
21. Методичні вказівки до самостійної роботи з дисципліни «Безпека праці в індустрії інформаційних технологій» для студентів усіх спеціальностей заочної форми навчання / Упоряд: Г.В. Пронюк, Т.Є. Стиценко, Н.М. Сердюк. – Харків: ХНУРЕ, 2017. – 32 с.
22. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу //Охорона праці. – 2001. –№ 12. – С. 12-20.
23. Жидецький В.Ц. Основи охорони праці. Підручник. – Львів: УАД, 2006. – 336 с.
24. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин.
25. Мясоєдова М. С. Важливість охорони праці на підприємствах [Електронний ресурс] / М. С. Мясоєдова. – 2019. – Режим доступу до ресурсу:https://archive.rmr.gov.ua/data/files/us_8zv//%D0%92%D0%B0%D0%B6%D0%BB%D0%B8%D0%B2%D1%96%D1%81%D1%82%D1%8C%20%D0%BE%D1%85%D0%BE%D1%80%D0%BE%D0%BD%D0%B8%20%D0%BF%D1%80%D0%B0%D1%86%D1%96%20%D0

%BD%D0%B0%20%D0%BF%D1%96%D0%B4%D0%BF%D1%80%D0%B8%D1%94%D0%BC%D1%81%D1%82%D0%B2%D0%B0%D1%85.pdf

ДОДАТОК А

Лістинг коду застосунку

Скрипт Player

```
public static Player Instance { get; private set; }
public Rigidbody2D PlayerRigidBody { get; private set; }
[SerializeField] private Animator _squashLanding;
[Header("Run variables")]
[SerializeField] private float _runSpeed = 10f;
[SerializeField] private float _acceleration = 40f;
[SerializeField] private float _decceleration = 40f;
[SerializeField] private float _frictionForce = 1.5f;
[Header("Jump variables")]
[SerializeField] private float _jumpHeight = 70f;
[SerializeField] private float _jumpCutMultiplier = 0.8f;
[SerializeField] private float _jumpBufferTime = 0.2f;
private float _jumpBufferCounter;
[SerializeField] private float _coyoteTime = 0.2f;
private float _coyoteTimeCounter;
[SerializeField] private Transform _groundCheck;
[SerializeField] private LayerMask _isGround;
private float _groundCheckDistance = 0.01f;
[Header("Wall jump variables")]
[SerializeField] private Transform _wallCheck;
[SerializeField] private LayerMask _isWall;
```

```
[SerializeField] private float _wallHopForce = 10f;
[SerializeField] private float _wallJumpForce = 20f;
[SerializeField] private float _wallHangTime = 2f;
private float wallHangTimer = 0f;
private float _wallCheckDistance = 0.7f;
private int _facingDirection = 1;
[Header("Dash variables")]
[SerializeField] private float _dashForce = 40f;
[SerializeField] private float _dashTime = 0.5f;
private float _playerGravityScale;
public bool DashState { get; private set; }
[Header("Particle systems")]
[SerializeField] private ParticleSystem _landingDust;
private bool _playerAirborn;
    private void Awake()
    {
        Instance = this;
        PlayerRigidBody = GetComponent<Rigidbody2D>();
        _playerGravityScale = PlayerRigidBody.gravityScale;
    }
    public bool CheckPlayerDirection(float runDirection, bool facingRight)
    {
        bool facingDirection = facingRight;
        if (runDirection == 1 && !facingRight)
```

```
        {
            Flip();
            facingDirection = true;
        }
        if (runDirection == -1 && facingRight)
        {
            Flip();
            facingDirection = false;
        }
        return facingDirection;
    }

    public bool CheckRunning(float runDirection)
    {
        if (runDirection == 1 || runDirection == -1)
        {
            return true;
        }
        else return false;
    }
    public bool CheckIfGrounded()
    {
        bool isGrounded = Physics2D.Raycast(_groundCheck.position, -
Vector2.up, _groundCheckDistance);
```

```
        CheckPlayerLandedState(isGrounded);
        return isGrounded;
    }
    public bool CheckIfTouchingWall()
    {
        bool isTouchingWall = Physics2D.Raycast(_wallCheck.position,
transform.right, _wallCheckDistance, _isWall);

        return isTouchingWall;
    }
    public float CheckCoyoteTime(bool isGrounded)
    {
        if (isGrounded) _coyoteTimeCounter = _coyoteTime;
        else _coyoteTimeCounter -= Time.deltaTime;
        return _coyoteTimeCounter;
    }
    private void CheckPlayerLandedState(bool isGrounded)
    {
        if(!isGrounded)
        {
            if (!_playerAirborn) _playerAirborn = !_playerAirborn;
        }
        else if(_playerAirborn)
        {
```

```
            CreateLandingDust();
            _squashLanding.SetTrigger("SquashTrigger");
            //_squashLanding.ResetTrigger("SquashTrigger");
            _playerAirborn = false;
        }
    }
    public void PlayerRun(float runDirection, bool facingRight, bool
isGrounded)
    {
        float desiredSpeed = _runSpeed * runDirection;
        float speedDifference = desiredSpeed - PlayerRigidBody.velocity.x;
        float accelDeccel = (Mathf.Abs(desiredSpeed) > 0.01f) ? _acceleration :
_decceleration;
        float runMovement = Mathf.Pow(Mathf.Abs(speedDifference) *
accelDeccel, 0.9f) * Mathf.Sign(speedDifference);
        PlayerRigidBody.AddForce(runMovement * Vector2.right,
ForceMode2D.Force);
    }
    public void PlayerJump()
    {
        PlayerRigidBody.AddForce(_jumpHeight * Vector2.up,
ForceMode2D.Impulse);

        _jumpBufferCounter = 0f;
        _coyoteTimeCounter = 0f;
```

```
}
public float BufferJump(bool jumpKeyPressed)
{
    if (jumpKeyPressed) _jumpBufferCounter = _jumpBufferTime;
    _jumpBufferCounter -= Time.deltaTime;
    return _jumpBufferCounter;
}
public void GrabWall(bool isGrounded, bool isTouchingWall)
{
    if(!isGrounded && isTouchingWall)
    {
        wallHangTimer += Time.deltaTime;
        ResetPlayerYVelocity();
        PlayerRigidBody.gravityScale = 0f;

        if(wallHangTimer >= _wallHangTime)
        {
            wallHangTimer = 0f;
            PlayerRigidBody.gravityScale = _playerGravityScale;
            WallHop();
        }
    }
    else
    {
```

```
            wallHangTimer = 0f;
            PlayerRigidBody.gravityScale = _playerGravityScale;
        }
    }
    private void WallHop()
    {
        float wallHopDirection = ManageWallJumpHopDirection();
        PlayerRigidBody.AddForce(new Vector2(wallHopDirection, 0f) *
_wallHopForce, ForceMode2D.Impulse);

        Debug.Log("wall hop");
    }
    public float WallJump()
    {
        float wallJumpDirection = ManageWallJumpHopDirection();
        PlayerRigidBody.AddForce(new Vector2(wallJumpDirection * 80f,
_wallJumpForce), ForceMode2D.Impulse);
        Flip();
        return wallJumpDirection;
    }
    private float ManageWallJumpHopDirection()
    {
        float direction = 0f;
        if (_facingDirection == 1) direction = -1;
```

```
        else if (_facingDirection == -1) direction = 1;
        return direction;
    }

    public void PlayerFriction(bool isGrounded, float input)
    {
        if(isGrounded && Mathf.Abs(input) < 0.01f)
        {
            float frictionAmount =
Mathf.Min(Mathf.Abs(PlayerRigidBody.velocity.x), _frictionForce);
            frictionAmount *= Mathf.Sign(PlayerRigidBody.velocity.x);
            PlayerRigidBody.AddForce(Vector2.right * -frictionAmount,
ForceMode2D.Impulse);
        }
    }
    public void PlayerFallDown()
    {
        if(PlayerRigidBody.velocity.y > 0)
        {
            PlayerRigidBody.AddForce(Vector2.down *
PlayerRigidBody.velocity.y * _jumpCutMultiplier);
        }
    }
    public IEnumerator PlayerDash(float dashDirection)
```

```
    {
        DashState = true;
        ResetPlayerYVelocity();
        PlayerRigidBody.AddForce(new Vector2(dashDirection * _dashForce, 0f),
ForceMode2D.Impulse);
        PlayerRigidBody.gravityScale = 0f;
        yield return new WaitForSeconds(_dashTime);
        PlayerRigidBody.gravityScale = _playerGravityScale;
        DashState = false;
    }
    public void ResetPlayerYVelocity()
    {
        PlayerRigidBody.velocity = new Vector2(PlayerRigidBody.velocity.x, 0f);
    }

    public void Flip()
    {
        _facingDirection *= -1;
        transform.Rotate(0f, 180f, 0f);
    }
    private void CreateLandingDust()
    {
        _landingDust.Play();
    }
```

Скрипт InputHandler

```
public delegate void OnHorizontalInput(bool isRunning);
public event OnHorizontalInput OnHorizontalInputEvent;
public delegate void OnVerticalInput(float YVelocity, bool isGrounded);
public event OnVerticalInput OnVerticalInputEvent;
public delegate void OnWallGrab(bool isGrounded, bool isTouchingWall);
public event OnWallGrab OnWallGrabEvent;
public delegate void OnDashInput(bool isDashing);
public event OnDashInput OnDashInputEvent;
private bool _facingRight = true;
   //run variables
private float _runDirection;
   //jump variables
private bool _canJump = false;
private bool _fallDown = false;
private float _jumpBufferTime;
private float _coyoteTime;
   //wall jump variables
private bool _canWallJump = false;
   //animator state variables
private bool _isRunning;
private bool _isGrounded;
private bool _isTouchingWall;
```

```
private bool _isDashing;
    //dash variables
private bool _canDash = false;
private float _dashCoolDown = 0.5f;
private float _dashCDCounter = 0f;
   private void Update()
    {
        //input direction check
        _runDirection = Input.GetAxisRaw("Horizontal");
        //player state check
        _facingRight = Player.Instance.CheckPlayerDirection(_runDirection,
_facingRight);
        _isRunning = Player.Instance.CheckRunning(_runDirection);
        _isGrounded = Player.Instance.CheckIfGrounded();
        _isDashing = Player.Instance.DashState;
        if(_isGrounded) Player.Instance.ResetPlayerYVelocity();
        _isTouchingWall = Player.Instance.CheckIfTouchingWall();
        _coyoteTime = Player.Instance.CheckCoyoteTime(_isGrounded);
        _jumpBufferTime =
Player.Instance.BufferJump(Input.GetKeyDown(KeyCode.Space));
        //jump check
        if (_coyoteTime > 0f && _jumpBufferTime > 0f)
        {
            _canJump = true;
```

```
        }
        //falling down check
        if (Input.GetKeyUp(KeyCode.Space) && _coyoteTime < 0f &&
!_isGrounded)
        {
            _fallDown = true;
        }
        //check dash cool down
        _dashCDCounter -= Time.deltaTime;
        //dash check
        if(Input.GetKeyDown(KeyCode.LeftShift) && _dashCDCounter <= 0f)
        {
            _canDash = true;
        }
        //wall jump check
        if(Input.GetKeyDown(KeyCode.Space) && !_isGrounded &&
_isTouchingWall)
        {
            _canWallJump = true;
        }
        //animation events
        OnHorizontalInputEvent?.Invoke(_isRunning);
      OnVerticalInputEvent?.Invoke(Player.Instance.PlayerRigidBody.velocity.y,
_isGrounded);
```

```
    OnWallGrabEvent?.Invoke(_isGrounded, _isTouchingWall);
    OnDashInputEvent?.Invoke(_isDashing);
}

private void FixedUpdate()
{
    //make player run
    if(!_canDash)
    {
        Player.Instance.PlayerRun(_runDirection, _facingRight, _isGrounded);
    }
    //apply friction
    Player.Instance.PlayerFriction(_isGrounded, _runDirection);
    //make jump and handle jump cut(variable jump height)
    if (_canJump)
    {
        Player.Instance.PlayerJump();
        _canJump = false;
    }

    if(_fallDown)
    {
        Player.Instance.PlayerFallDown();
        _fallDown = false;
```

```
        }
        //make player grab a wall
        if (!_isDashing) Player.Instance.GrabWall(_isGrounded,
_isTouchingWall);
        //make player dash
        if (_canDash)
        {
            StartCoroutine(Player.Instance.PlayerDash(_runDirection));
            _canDash = false;
            _dashCDCounter = _dashCoolDown;
        }
        //make player wall jump
        if (_canWallJump)
        {
            float wallJumpDirection = Player.Instance.WallJump();
            _canWallJump = false;
            if (wallJumpDirection == 1f) _facingRight = true;
            if(wallJumpDirection == -1f) _facingRight = false;
        }
    }
```

Скрипт RobotStateController

```
public Animator animator;
private PatrolState _patrolState;
```

```
private CombatState _combatState;
[SerializeField] public Transform player;
[SerializeField] private float activationZone;
private bool playerInRange = false;
    private void Start()
    {
        _patrolState = GetComponent<PatrolState>();
        _combatState = GetComponent<CombatState>();
    }
    private void Update()
    {
        animator.SetFloat("Speed", Mathf.Abs(_patrolState.rb.velocity.x));
        ChangeState(CheckForPlayer());
        animator.SetBool("Shoot", playerInRange);
    }
    private bool CheckForPlayer()
    {
        float distance = Vector2.Distance(player.position, transform.position);
        if (distance < activationZone) playerInRange = true;
        else playerInRange = false;
        return playerInRange;
    }
    private void ChangeState(bool isInRange)
    {
```

```
        if (isInRange)
        {
            _combatState.enabled = true;
            _patrolState.enabled = false;
        }
        else
        {
            _combatState.enabled = false;
            _patrolState.enabled = true;
        }
    }
    public void Flip(float angle)
    {
        transform.rotation = Quaternion.Euler(0f, angle, 0f);
    }
```

Скрипт PatrolState

```
[SerializeField] private float _patrolDistance;
[SerializeField] private float _walkSpeed;
private float _originLocation;
private float _patrolBorder;
private float _currentWalkSpeed;
public RobotStateController _controller;
private float angle = 0f;
```

```
public Rigidbody2D rb;
public SpriteRenderer spriteRenderer;
  private void Start()
  {
     rb = GetComponent<Rigidbody2D>();
     _originLocation = transform.position.x;
     _patrolBorder = _originLocation - _patrolDistance;
     _currentWalkSpeed = _walkSpeed;
  }
  private void Update()
  {
     Patrol();
  }
  private void FixedUpdate()
  {
     rb.velocity = new Vector2(_currentWalkSpeed, 0f);
  }
  private void Patrol()
  {
     if(transform.position.x < _patrolBorder)
     {
        angle = 180f;
        _currentWalkSpeed = _walkSpeed;
     }
```

```
        if(transform.position.x > _originLocation)
        {
            angle = 0f;
            _currentWalkSpeed = -_walkSpeed;
        }
        _controller.Flip(angle);
    }
```

Скрипт CombatState

```
private RobotStateController _controller;
[SerializeField] private Transform _firePoint;
[SerializeField] private float fireDistance;
[SerializeField] private LayerMask playerLayer;
    private void Start()
    {
        _controller = GetComponent<RobotStateController>();
    }
    private void Update()
    {
        //Debug.Log("rotation: " + transform.rotation.eulerAngles.y);
        //Debug.Log("angle: " + FindAngle());
        if(FindAngle() > 90f && transform.rotation.eulerAngles.y == 180f)
_controller.Flip(0f);
```

```
    if(FindAngle() < 90f && transform.rotation.eulerAngles.y == 0)
_controller.Flip(180f);
  }
  private float FindAngle()
  {
    Vector2 dir = _controller.player.position - transform.position;
    float angle = Mathf.Atan2(dir.y, dir.x) * Mathf.Rad2Deg;
    return Mathf.Abs(angle);
  }
  public void InflictDamage()
  {
    if (CheckHit())
    {
      Debug.Log("Damaging");
    }
  }
  public bool CheckHit()
  {
    bool isHitting = Physics2D.Raycast(_firePoint.position, -_firePoint.right,
fireDistance, playerLayer);
    return isHitting;
  }
```

ДОДАТОК Б

Критерії бальної оцінки умов праці

;№ п/п	Фактор умов праці на робочому місці	Оцінка, бали					
		1	2	3	4	5	6
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, °С: - теплий період - холодний період	23…25 21…23	26…28 18…20	29…32 15…17	33…35 12…14	35…37 Нижче +12	>37
2	Відносна вологість повітря на РМ, %	40…50	55…60	61…75	76…85	Понад 85	-
3	Швидкість руху повітря на РМ, м/с	Менше 0,2	0,2…0,5	0,6…0,7	0,8…1,2	1,3…1,7	Понад 1,7
4	Освітленість на РМ, лк	≥ 300	240…300	160…230	100…150	60…90	30…50
5	Мінімальний розмір об’єкта розпізнавання, мм	> 1,0	1…0,3	< 0,3	0,005…0,3	< 0,05	-
6	Виробничий шум, перевищення ГДР, дБА	< 1	Рівно ГДР	1…5	6…10	> 10	> 10 з вібрацією
7	Інтенсивність теплового випромінювання, Вт/м²	≤ 140	141..1000	1001…1500	1501…2000	2001…2500	>2500
8	Токсична речовина, озон, кратність перевищення ГДК	-	≤ 1	1…2,5	2,6…4,0	4,1…6	> 6,0
9	Виробничий пил (паперовий), кратність перевищення ГДК	-	≤ 1	1…5	6…10	11…30	> 30

Продовження таблиці

10	Робоче місце (РМ), поза та переміщення у просторі	РМ стаціонарне, поза вільна, маса переміщене вантажу $\leq$ 5 кг	РМ стаціонарне,поза вільна, маса переміщене вантажу > 5 кг	Робоче місце стаціонарне, поза не вільна, до 25 % часу зміни у нахиленому положенні до 30°	РМ стаціонарне ,поза вимушена – до 50 % робочої зміни	РМ стаціонарне, поза вимушена, незручна – більше 50 % робочої зміни	РМ стаціонарне, поза вимушена, незручна, нахили під кутом до 60 град більше 300 разів за робочу змін
11	Кількість важливих об’єктів спостереження	Менше 5	5…10	11…25	Понад 25	-	-
12	Тривалість зосередженого спостереження, % часу зміни	Менше 25	25…50	51…75	76…85	86…90	Понад 90
13	Тривалість повторюваних операцій, с	Понад 100	31…100	20…30	10…19	5…9	1…4
14	Змінність роботи	Ранкова зміна	Дві зміни	Три зміни	Нерегуляр-ні зміни	-	-
15	Тривалість безперервної роботи за добу, годин	-	< 8	< 12	> 12	-	-

Продовження таблиці

16	Режим праці та відпочинку	Обґрунто-ваний, з включенням музики та гімнастики	Обґрунто-ваний, без включення музики та гімнастики	Відсутність обґрунтованого режиму праці та відпочинку	-	-	-
17	Нервово-емоційне навантаження	Прості дії за індивідуальним планом	Прості дії за заданим планом з можливістю корегування	Складні дії за заданим планом з можливістю корегування	Складні дії за заданим планом при дефіциті часу	Відповідальність за безпеку людей	Індивідуальний ризик
18	Кількість рухів пальців на годину	< 360	360…720	721…1080	1081…3000	> 3000	-
19	Монотонність, тривалість операцій, які повторюються , с	> 100	31…100	20…30	10…19	5…9	1…4