МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет менеджменту

Кафедра економічної кібернетики, комп'ютерних наук
та інформаційних технологій

Кваліфікаційна наукова
праця на правах рукопису

ШУТКЕВИЧ Павло Петрович

УДК 004.514.64

КВАЛІФІКАЦІЙНА РОБОТА

РОЗРОБКА ГРИ НА UNITY З ВИКОРИСТАННЯМ ФІЗИЧНОГО РУШІЯ ДЛЯ СИМУЛЯЦІЇ РУХІВ

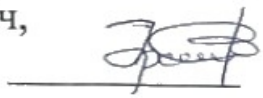
Спеціальність 122 «Комп'ютерні науки»
Галузь знань – 12 «Інформаційні технології»

Подається на здобуття освітнього ступеня «Бакалавр»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Шуткевич П.П.

Науковий керівник: Крайній Володимир Олексійович,
кандидат економічних наук, в.о. доцента



Завідувач кафедри: Тищенко Світлана Іванівна,
кандидат педагогічних наук, доцент

Миколаїв–2025

ЗМІСТ

Анотація

Шуткевич П.П. Розробка стратегічної гри на Unity з використанням фізичного рушія для симуляції руху – Кваліфікаційна наукова праця на правах рукопису. Робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки». – Миколаївський національний аграрний університет, Миколаїв, 2025.

У кваліфікаційній роботі розглянуто процес створення комп’ютерної гри в жанрі стратегії з використанням сучасного рушія Unity. Особливу увагу приділено застосуванню фізичного модулю рушія для моделювання реалістичного руху та взаємодії ігрових об’єктів. Реалізовано динамічні сцени з рухомими юнітами, побудову баз, створення ігрових сценаріїв та системи зіткнень, що враховують фізичні закони.

Проект передбачає впровадження базового штучного інтелекту, який дозволяє ворогам здійснювати прості рішення, орієнтуватися в просторі та реагувати на дії гравця. Крім того, розроблено механізм взаємодії між юнітами гравця та супротивника, включаючи логіку атаки, захисту та пересування в межах ігрового середовища.

Робота ґрунтується на аналізі сучасних підходів до створення стратегічних ігор, а також на практичному використанні можливостей Unity, таких як компоненти Rigidbody, фізичні матеріали, а також API для роботи з фізикою.

Методика включає проєктування архітектури гри, розробку окремих модулів, тестування та оптимізацію. У результаті створено прототип гри, який може слугувати основою для розширення, комерційної реалізації або дослідження у сфері штучного інтелекту в іграх.

Ключові слова: Unity, ігровий штучний інтелект, стратегія, комп’ютерна гра, стратегія в реальному часі.

Abstract

Shutkevych P.P. Development of a Strategy Game in Unity Using a Physics Engine for Movement Simulation – Qualification scientific work in the form of a manuscript. A thesis submitted for the Bachelor's degree in specialty 122 "Computer Science". – Mykolaiv National Agrarian University, Mykolaiv, 2025.

This qualification work carried the process of developing a computer game in the strategy genre using the Unity game engine. The focus of the research lies in the integration of physics-based movement simulation to provide realistic dynamics and object interactions. The game prototype includes moving units, base building, and physics-aware collision mechanics that respond to environmental and gameplay conditions.

A simplified artificial intelligence system was implemented to control enemy units. It enables autonomous decision-making, movement through the game space, and reactive behavior to player actions. Both offensive and defensive unit logic were integrated into the game system to simulate real-time strategy scenarios.

The work is based on current trends in strategy game design and leverages Unity's physics capabilities, such as the Rigidbody component, physical materials, and physics scripting API. The development process included modular programming, testing of core systems, and gameplay optimization.

As a result, a functional prototype of a strategy game was created, demonstrating the feasibility of using Unity's physics engine to enhance strategic gameplay. The findings and the prototype can serve as a basis for further research or commercial development in the field of physics-based game mechanics and AI-driven behaviors.

Key words: Unity, game artificial intelligence, strategy, computer game, real-time strategy.

ВСТУП

Обґрунтування вибору теми дослідження. Сфера розробки комп’ютерних ігор стрімко розвивається, охоплюючи не лише розважальні, а й навчальні, соціальні та симуляційні аспекти. Одним із найбільш динамічних та інтелектуально насичених жанрів виступають стратегічні ігри, що вимагають від гравця логічного мислення, планування та прийняття рішень у складних умовах. У свою чергу, сучасні ігрові рушії, зокрема Unity, надають потужні інструменти для моделювання фізики, що значно розширює можливості реалізації динамічних сценаріїв у стратегічних іграх.

Актуальність теми зумовлена необхідністю створення нових ігрових продуктів з високим рівнем реалізму та занурення, що забезпечується не лише графікою, а й достовірною фізикою та поведінкою об’єктів. Симуляція руху, зіткнень, взаємодії юнітів із середовищем та між собою створює новий рівень глибини геймплею та відкриває нові підходи до розробки ігрових механік у жанрі стратегій.

Мета і завдання дослідження. Метою кваліфікаційної роботи є створення гри в жанрі стратегія з використанням рушія Unity, в якій фізичні взаємодії між об’єктами, симуляція руху та поведінка ворогів реалізовані за допомогою вбудованого фізичного рушія та модулів ігрового штучного інтелекту. Для досягнення поставленої мети необхідно виконати такі завдання:

- проаналізувати технічні можливості рушія Unity щодо реалізації фізичних механік;
- розробити систему вибору, генерації та керування юнітами;
- впровадити механізми зіткнень, рухів та візуального супроводу фізичних процесів;
- реалізувати базові сценарії ігрового ШІ, включаючи поведінку ворожих юнітів;

- протестувати ігрову систему в умовах наближених до реального геймплею;
- оцінити потенціал масштабування проекту.

Об’єкт дослідження – процес створення стратегічної гри в середовищі Unity із застосуванням фізичного рушія для реалістичного моделювання рухів персонажів, взаємодії між об’єктами та адаптивної поведінки ворогів.

Предмет дослідження – практична реалізація фізичних механік у стратегічній грі, що включає симуляцію рухів юнітів, вплив навколишнього середовища на їхню поведінку, а також алгоритми ігрового штучного інтелекту для оптимального управління юнітами гравця та ворогів.

Методи дослідження. У дослідженні використано аналіз літератури та прикладних ігрових проєктів, методи моделювання, компонентний підхід у розробці архітектури, програмні засоби Unity, а також експериментальне тестування з метою оцінки працездатності розроблених ігрових механік.

Практичне значення. Результати роботи можуть бути використані як основа для створення повноцінних стратегічних ігор, навчальних симуляторів або комерційних проєктів. Запропоновані підходи можуть стати базою для подальшого дослідження інтелектуальної поведінки у фізично достовірних ігрових світах.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків і списку використаних джерел. Загальний обсяг – 83 сторінки, включає 63 рисунків. Список використаних джерел налічує 35 найменувань.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ ІГРОВОГО РУШІЯ UNITY

1.1 Опис ігрового рушія Unity

Unity – це один з найбільш потужних і популярних ігрових рушіїв на сьогоднішній день, який надає величезні можливості для розробки ігор для різноманітних платформ. Він дозволяє створювати як 2D, так і 3D ігри, використовуючи мову програмування C#. Завдяки своїй універсальності, Unity забезпечує розробникам гнучкість і простоту при створенні ігор для широкого спектра платформ, таких як Windows, macOS, Linux, Android, iOS, консолі, веб-браузери та навіть віртуальна реальність.

Однією з головних переваг Unity є можливість швидко інтегрувати асети з Unity Asset Store. Це — онлайн-магазин, в якому можна знайти безліч готових ресурсів: 3D-моделей, текстур, анімацій, звукових ефектів, скриптів та багато іншого. Розробники можуть легко додавати ці ресурси в свій проект, що значно скорочує час на розробку. Це дозволяє зосередитися на розробці геймплейної механіки та інших важливих аспектах гри, замість того, щоб створювати базові ресурси з нуля. Також Unity дозволяє створювати власні асети та публікувати їх на Asset Store, що дає можливість заробляти на своїх розробках.

Unity надає величезну кількість інструментів для розробки ігор. Ось деякі з основних:

- Інструменти для 2D і 3D розробки:

➢ 2D-розробка: Unity має потужні інструменти для створення 2D-ігор, включаючи підтримку спрайтів, анімації спрайтів, фізики для 2D-об'єктів, а також інструменти для роботи з камерами та освітленням для 2D.

➢ 3D-розробка: Для 3D-ігор Unity надає високоякісні інструменти для роботи з моделями, текстуруванням, анімацією та рендерингом. Unity підтримує популярні формати моделей, що дозволяє імпортувати їх з таких програм, як Blender або Maya.

- Фізика та фізичні ефекти:

➢ Unity включає вбудовану систему фізики, яка підтримує фізичні взаємодії між об'єктами, такі як зіткнення, гравітація, сили, імпульс та інші фізичні явища.

➢ Для роботи з фізикою в Unity використовуються компоненти Rigidbody та Collider, які дозволяють налаштовувати фізичні властивості об'єктів у 3D та 2D середовищах.

➢ NavMesh — це інструмент для створення навігаційних мереж, що дозволяє персонажам AI орієнтуватися в просторі, уникати перешкод і знаходити оптимальні шляхи для руху.

• Штучний інтелект (AI):

➢ Unity надає потужні інструменти для розробки AI для ігор. Зокрема, можна використовувати NavMesh для створення шляху для AI-агентів, які будуть рухатись в межах обмежених територій, уникати перешкод і здійснювати складні рухи.

➢ Для побудови логіки AI можна застосовувати Finite State Machines (FSM) для створення реакцій AI на різні умови в грі. Unity також дозволяє використовувати зовнішні бібліотеки та плагіни для впровадження більш складних алгоритмів машинного навчання, таких як ML-Agents, для створення адаптивного AI.

• Ефекти та постобробка:

➢ Unity надає можливість створювати візуальні ефекти за допомогою Particle System. Це дає змогу додавати в гру різноманітні ефекти, такі як вибухи, дим, вогонь, дощ та інші.

➢ Post-Processing Stack дозволяє застосовувати фільтри та ефекти до всього зображення після рендеринга, включаючи ефекти, такі як глибина різкості, колірна корекція, блюр, тіні та багато іншого.

• Робота з освітленням:

➢ Unity підтримує різноманітні типи освітлення, такі як Directional Lights, Point Lights, Spot Lights та Area Lights, що дозволяє налаштовувати як статичне, так і динамічне освітлення.

➢ Unity також має Global Illumination для створення реалістичного освітлення, яке симулює відображення світла від поверхонь.

Unity підтримує багато платформ, що дає розробникам величезну гнучкість. Серед підтримуваних платформ:

• PC та Mac: Ігри можуть бути зібрані для Windows, macOS та Linux.

• Мобільні платформи: Unity дозволяє створювати ігри для Android, iOS та інших мобільних платформ.

• Консолі: Unity підтримує популярні ігрові консолі, такі як PlayStation, Xbox та Nintendo Switch.

• Веб: Ігри можна запускати прямо в браузерах через WebGL.

• Віртуальна реальність: Unity підтримує VR-гарнітури, такі як Oculus Rift, HTC Vive та інші, що дозволяє створювати інноваційні ігри в сфері віртуальної реальності.

Це дозволяє розробникам створювати ігри, які можуть працювати на більшості популярних пристроїв, без необхідності адаптувати кожен проект під кожну платформу окремо.

Основною мовою програмування в Unity є C#, що забезпечує високу продуктивність і гнучкість при розробці ігор. C# є однією з найбільш популярних мов програмування, що використовується в індустрії, і вона має велику кількість бібліотек і фреймворків, що спрощує процес розробки. C# дозволяє розробникам створювати складні ігрові механіки, скрипти для керування об'єктами, а також працювати з даними гри.

Unity також підтримує JavaScript і Boo, але C# є основною і найбільш оптимізованою мовою для роботи з Unity, забезпечуючи підтримку широкого спектра інструментів і бібліотек.

1.2 Огляд аналогів ігрового рушія

Термін "ігровий рушій" набув поширення в середині 1990-х років у зв’язку з розвитком 3D-ігор, особливо шутерів від першої особи. Згодом ці технології суттєво еволюціонували, що спричинило розмежування процесів створення та редагування ігор. Серед популярних рушіїв можна виокремити чотири, які мають подібні принципи розробки, але різняться набором інструментів. Далі детальніше розглянемо їхні особливості та унікальні можливості.

- ❖ Rage – це власний ігровий рушій компанії Rockstar Games, що використовується в легендарних іграх GTA, Red Dead Redemption та інших проектах Rockstar. Він має високу продуктивність, реалістичну фізику та AI, а також відмінну графіку.

  Переваги:

- Відкритий світ високої деталізації;
- Високий рівень реалізму;
- Гнучкість та кросплатформність;
- Якісна оптимізація;

  Недоліки:

- Закритий рушій;
- Складність оновлення;
- Високі вимоги до обладнання;

Рисунок 1.1 – Ігровий рушій Rage

Джерело: сформовано автором на основі [4,5,6]

❖ Unreal Engine – ігровий рушій, який розробляється та підтримується компанією Epic Games. Даний рушій написаний мовою програмування C++, рушій дає змогу створювати ігри для більшості операційних систем та більшості консолей. Історія даного рушія триває десятиліттями, проте зараз він здається досяг своєї величі, а також цей рушій є передовим рішенням для створення великих ААА-ігор.

Переваги:

- Потужний редактор попри всі випадки життя;
- Гнучка архітектура грального рушія;
- Готовий до ААА-проектів із коробки;
- Крос-платформний;

Недоліки:

- Вищий поріг входу;
- Більш закрита і не така численна спільнота;
- Акцент – на ААА-проекти;
- Розмір рушія та його вимогливість.

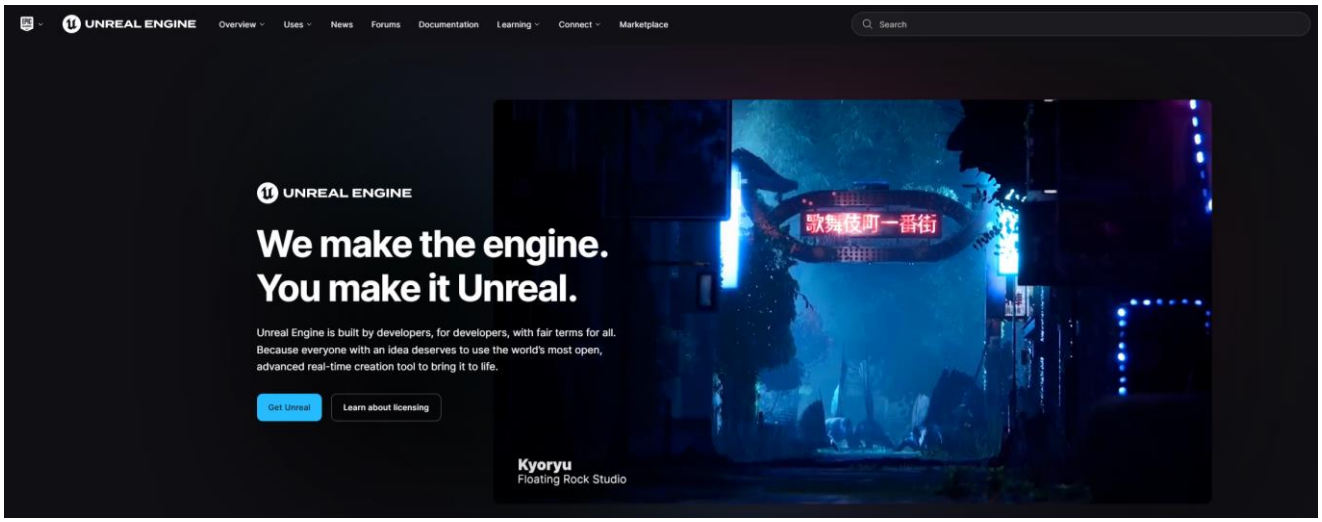


Рисунок 1.2 – Ігровий рушій Unreal Engine

Джерело: сформовано автором на основі [7,8,9,10]

❖ Ego – це власний ігровий рушій, розроблений Codemasters для створення високоякісних гоночних симуляторів. Його вперше використали у 2007 році в грі *Colin McRae: Dirt*, і він став основою для таких популярних

серій, як DiRT, F1 та GRID. Цей рушій є вдосконаленою версією Neon Engine, який Codemasters спочатку використовували для *Colin McRae: Dirt*. Він забезпечує реалістичну фізику, деталізовану графіку, продуману систему AI та оптимізовану продуктивність.

Переваги:

- Високий рівень реалістичності;
- Чудова графіка та спецефекти;
- Висока продуктивність і оптимізація;
- Розвинений AI для суперників;
- Гнучкість у розробці;

Недоліки:

- Обмежене використання для інших жанрів;
- Обмежені можливості моддингу;
- Фокус на гонки, а не на універсальність;
- Закритий рушій.

Рисунок 1.3 – Ігровий рушій Ego

Джерело: сформовано автором на основі [11,12,13]

❖ Anvil Engine — це власний рушій компанії Ubisoft, створений для гри з відкритими світами. Він використовується в Assassin's Creed, Watch Dogs, For Honor та інших іграх Ubisoft. Його сильні сторони — це деталізовані світи, реалістична анімація, велика кількість NPC та продвинута фізика.

Переваги:

- Відкриті світи з високою деталізацією;
- Продвинута система анімації;
- Гарна оптимізація для консолей;
- Велика кількість NPC;
- Розвинені графічні ефекти;

Недоліки:

- Проблеми з продуктивністю та оптимізацією;
- Закритий рушій;
- Складність розробки;
- Проблеми з фізикою та геймплеєм;
- Обмеження для моддингу.

Рисунок 1.4 – Ігровий рушій Anvil Engine

Джерело: сформовано автором на основі [14,15,16]

1.3 Технічне завдання

Для формування технічного завдання визначимо наступні елементи:

Загальні відомості:

Повна назва: дослідити та розкрити використання фізичного рушія для симуляції рухів в грі жанру стратегія за допомогою ігрового рушія, в якому є певні інструменти та команди, що реалізовують фізичну взаємодію об'єктів, моделювання траєкторій руху та механік зіткнення, а також їх вплив на ігровий процес і стратегічні рішення гравця.

Передумови для проведення робіт:

Стратегічні ігри є одним із найрізноманітніших жанрів, оскільки можуть охоплювати різні історичні періоди, альтернативні реальності або навіть фантастичні світи. Вони пропонують широкий спектр механік, включаючи управління ресурсами, будівництво, військову тактику та економічне планування. Однак через велику кількість проектів у цьому жанрі гравці часто стикаються як із якісними, продуманими іграми, так і з менш вдалими реалізаціями, що мають слабкий баланс, застарілу графіку або незручний інтерфейс.

Щоб вирішити цю проблему, важливо впроваджувати в стратегії сучасні технології та інноваційні механіки. Використання вдосконаленої фізики дозволить зробити моделювання бойових дій та руйнувань більш реалістичним, а розширені алгоритми штучного інтелекту сприятимуть динамічній поведінці супротивників, що адаптуються до дій гравця.

Розробка подібних стратегій не лише покращить якість жанру, а й сприятиме його популяризації серед ширшої аудиторії. Досвідчені гравці зможуть знайти для себе нові виклики та цікаві механіки, тоді як новачки отримають доступ до більш інтуїтивного, але водночас глибокого ігрового процесу. Впровадження таких змін допоможе залучити нових шанувальників стратегічних ігор та зробити жанр ще більш захопливим і конкурентоспроможним у сучасній ігровій індустрії.

Характеристика об’ єкту проектування:

Дослідженню підлягає використання таких компонентів гри, як вибору юнітів, механізм генерації юнітів гравця та ворогів та ігровий штучний інтелект в грі жанру стратегія.

Вимоги до функціональних характеристик:

Програмний продукт повинен володіти наступними функціональними характеристиками:

- можливість вибору юнітів гравця завдяки лівої кнопки миші;
- пересування камери за допомогою WASD, поворот камери – Q та E;
- прискорення швидкості руху камери за допомогою лівого Shift;
- генерація свого або своїх ангарів завдяки кнопці Hangar;
- переміщення обраних гравцем юнітів за допомогою правої кнопки миші;
- кнопка Exit відповідає за вихід з гри;
- кнопка Play відтворює тему з класичної гри «Танчики».

Вимоги до надійності:

Надійне функціонування гри в жанрі стратегія забезпечене шляхом:

- безпечне скачування гри з цифрових магазинів;
- оптимальне навантаження на різних типах комп'ютера;
- перевірка на віруси за допомогою програми антивірусу.

Вимоги до програмного забезпечення:

Комп’ютерна гра в жанрі стратегія повинна бути створена на базі ігрового рушія Unity з використанням фізичного рушія для симуляції рухів, а також візуальної та музичної складової, що вимагають декількох типів ліцензування, а саме безкоштовного використання, із зазначенням авторства та із зазначенням авторства – розповсюдження на тих самих умовах. Сам же ігровий рушій Unity написаний на мовах програмування C++ (Runtime) і C# та повинен бути останньої версії, а саме версії 6000.0.38f1, а також повинен мати

декілька ліцензій, а саме вільну, ліцензію MIT, а також загальна громадську ліцензію обмеженого використання GNU.

Вимоги до технічного забезпечення:

Програмний комплекс, який розрахований на функціонування при такому наборі технічних засобів:

мінімальна апаратна конфігурація комп'ютера гравця:

- процесор з тактовою частотою 3.40 ГГц ;
- ОЗУ не менше 32 Гб ;
- відеокарта та монітор з роздільною здатністю не менше 2560x1440;
- твердотілий накопичувач місткістю 1Тб;
- клавіатура;
- маніпулятор типу «миша»;

Вимоги до ергономіки та технічної естетики:

Взаємодія користувачів, а саме гравців з грою в жанрі стратегія повинна реалізовуватись через зручний та інтуїтивно зрозумілий графічний інтерфейс. Він має бути візуально чистим і не перевантаженим зайвими елементами, щоб не ускладнювати сприйняття інформації. Важливо, щоб усі основні функції були доступні без зайвих дій, що дозволить гравцям зосередитися на стратегічному плануванні та управлінні ресурсами.

До інтерфейсу стратегічної гри висуваються наступні вимоги:

- мінімально можливий час завантаження гри;
- зручна навігація, яка дозволяє легко орієнтуватися в ігровому просторі;
- швидкий доступ до важливих інструментів керування без зайвих дій.

Адаптивний інтерфейс забезпечить комфортне керування для різних категорій гравців, сприяючи зручному управлінню ресурсами, військовими силами та іншими стратегічними елементами. Це допоможе зробити ігровий процес більш ефективним і захопливим.

Висновки до розділу 1

Проаналізувавши перший розділ, можна зробити стосовно нього висновок, а саме:

- Був розглянутий опис ігрового рушія Unity в цілому, були розглянуті основні принципи використання рушія, короткий опис терміну стратегія, можливості та інструменти рушія, які використовуються для створення комп'ютерної гри в жанрі стратегія, на яких мовах написаний рушій та як ці мови використовуються при створенні гри, а також як ці мови збільшують потенційні можливості в сеттингу гри, умови використання рушія для некомерційних та комерційних цілей. Також було розглянуто умову роботи рушія на операційних системах в якості основних платформ;
- Був проведений огляд аналогів ігрових рушіїв, серед яких були обрані чотири ігрових рушія, які є найбільш простими та зручним для розробників та гравців. Переглянувши та прочитавши інформацію стосовно кожного з цих рушіїв, виділивши їх сильні та слабкі сторони, був зроблений висновок, що з певністю можна сказати, що найкращим ігровим рушієм серед представлених аналогів є ігровий рушій Unreal Engine, так як цей рушій є популярним та відомим серед розробників, користувачів та гравців і містить в собі корисні інструменти для створення гри, які розробник може використовувати під час створення власного проекту;
- Було розглянуто технічне завдання, в ході якого сформувались загальні відомості тематики гри жанру стратегія, передумови для проведення робіт, характеристика об`єкту проектування, функціональні характеристики, вимоги надійності, а також програмного та технічного забезпечень для комп'ютерної гри в жанрі стратегія і вимоги до ергономіки та технічної естетики графічного інтерфейсу.

РОЗДІЛ 2. ТЕХНОЛОГІЧНІ ТА ДЕТАЛЬНІ АСПЕКТИ РОЗРОБКИ ІГОР ЖАНРУ СТРАТЕГІЯ

2.1 Аналітичний опис фізики та фізичних рушіїв в комп’ютерних іграх

Сучасна індустрія розробки комп’ютерних ігор характеризується високим попитом на ігрові рушії, які здатні ефективно моделювати фізичні явища та забезпечувати широкий спектр інструментів для реалізації складного ігрового процесу. Ігровий рушій — це спеціалізоване програмне забезпечення, що слугує основою для створення комп’ютерних ігор, надаючи розробникам набір готових інструментів і бібліотек для побудови графіки, фізичних взаємодій, звукового супроводу, штучного інтелекту, мережевого функціоналу, а також логіки гри. Завдяки цьому, розробка стає більш гнучкою та ефективною, оскільки дозволяє зосередитися безпосередньо на творчих і дизайнерських аспектах проєкту.

Ігрові рушії активно застосовуються в створенні різноманітних жанрів — від класичних 2D-платформерів і мобільних казуальних ігор до високотехнологічних 3D-шутерів, стратегій у реальному часі й симуляторів. Усе це стало можливим завдяки розвитку рушіїв, які здатні обробляти велику кількість даних у реальному часі, моделювати складні фізичні сцени й забезпечувати високу якість графіки.

Моделювання у сфері комп’ютерних ігор передбачає створення віртуального світу, включаючи 3D-моделі персонажів, предметів, транспорту, архітектури, природного ландшафту та інших елементів. Важливу роль відіграє анімація та налаштування поведінки об’єктів — як у відповідь на дії гравця, так і у взаємодії між собою. Для цього використовуються спеціалізовані програми (наприклад, Blender, Autodesk Maya, 3ds Max), після чого готові моделі інтегруються до ігрового рушія, де отримують свою функціональність.

Один із ключових аспектів такого моделювання — фізика, тобто система, що імітує реальні закони природи. Вона забезпечує правдоподібну поведінку об’єктів у грі: рух, силу тяжіння, зіткнення, інерцію, тертя, взаємодію з рідинами, освітлення, тіні та інші ефекти. Реалістична фізика дозволяє гравцеві

краще зануритися у гру, оскільки взаємодія з ігровим світом відбувається згідно з очікуваними законами.

У більшості сучасних рушіїв, таких як Unity, Unreal Engine або CryEngine, вже вбудовані фізичні модулі (наприклад, NVIDIA PhysX або Chaos Physics), які дозволяють відтворювати фізичні процеси в реальному часі. Такі рушії підтримують складні обчислення фізичних взаємодій, що є критично важливим у проєктах, де динаміка об’єктів безпосередньо впливає на геймплей.

Фізика у відеоіграх є не лише інструментом розваги, а й важливим напрямом технічного розвитку. За статистичними даними, світовий ринок комп’ютерних ігор щороку зростає на приблизно 8%, при цьому сектор мобільних ігор демонструє ще вищу динаміку — близько 15% на рік. Це свідчить про стабільний попит на технології, що забезпечують глибоке й реалістичне занурення користувача.

Наукові й прикладні дослідження в галузі фізики для комп’ютерних ігор здебільшого ґрунтуються на класичних теоріях механіки та кінематики, хоча з розвитком технологій активно досліджуються й нові підходи до моделювання складних фізичних систем. Типова структура ігрового рушія включає графічний, фізичний, звуковий модулі та систему ігрового штучного інтелекту, які разом формують технічну основу сучасного інтерактивного контенту.

Графічний рушій (англ. graphics engine) — це програмний компонент, який відповідає за створення та відображення двовимірної й тривимірної графіки у відеоіграх. Його основне призначення — реалізація графічних ефектів, таких як анімація, освітлення, тіні, текстурування та інші візуальні елементи, що формують загальну картину ігрового середовища. У структурі графічного рушія можуть бути присутні окремі модулі: геометричний рушій, система обробки частинок, підсистема освітлення, фізичний модуль тощо. Усі ці елементи тісно взаємодіють між собою, а також з іншими складовими ігрового рушія — наприклад, модулями звуку чи штучного інтелекту — забезпечуючи цілісне графічне сприйняття гри.

Фізичний рушій (англ. physics engine) — це програмний модуль, який відповідає за моделювання фізичних процесів у віртуальному просторі гри. Він дозволяє обчислювати рух об’єктів, симулювати їхню взаємодію під впливом сили тяжіння, тертя, зіткнень та інших фізичних явищ. Одним із найпоширеніших фізичних рушіїв для 2D-проєктів є Box2D, спочатку розроблений на мові C++ і згодом адаптований для багатьох інших мов програмування. Іншим відомим прикладом є Chipmunk2D, який широко використовується у фреймворках на зразок Cocos2D. Варто зазначити, що в деяких жанрах, таких як шутери або ігри від третьої особи, інтенсивне використання фізичних симуляцій стало популярним відносно недавно. Тут фізика реалізується в широкому спектрі — від базових ragdoll-анімацій та взаємодії з об’єктами до розширеного геймплею, побудованого на фізичних принципах (наприклад, у Half-Life 2 або Psi-Ops).

Звуковий рушій (англ. sound або audio engine) — це програмна система, призначена для генерації, обробки та відтворення звукового контенту у відеоіграх. Аудіо супровід є важливим елементом, що підсилює атмосферу гри та забезпечує глибше занурення гравця в ігровий світ. Завдяки звуковим рушіям розробники можуть реалізовувати широкий набір звукових ефектів — музичний супровід, репліки персонажів, звуки довкілля та інші акустичні елементи. Багато сучасних рушіїв підтримують функції мікшування звуків з різних джерел, просторовий 3D-звук, а також ефекти гучності та напрямку, що змінюються залежно від положення об’єкта в ігровому просторі.

Ігровий штучний інтелект (game artificial intelligence) — це програмна система, яка забезпечує реалізацію поведінкових моделей неігрових персонажів (NPC) у відеоіграх. Основна мета використання такого інтелекту — створити правдоподібну, адаптивну та варіативну поведінку персонажів, що сприяє глибшому зануренню гравця в ігровий світ і посиленню загального ефекту присутності.

Ігровий ШІ також може відповідати за логіку ігрового процесу, управління подіями та створення спеціальних ефектів, що надає грі

динамічності та інтерактивності. До появи універсальних ігрових рушіїв розробка ігор здійснювалася індивідуально для кожної платформи, враховуючи технічні особливості й обмеження апаратного забезпечення. Це вимагало значних витрат часу та ресурсів, оскільки кожен проєкт створювався "з нуля" під конкретну архітектуру.

Запровадження ігрових рушіїв суттєво змінило підхід до розробки ігор. Завдяки стандартизованим інструментам і шаблонам, розробники отримали змогу швидше створювати якісні продукти з використанням готових рішень. Сучасні рушії також підтримують мультиплатформенну розробку — створення ігор для ПК, консолей і мобільних пристроїв на базі одного коду, що значно підвищує продуктивність і знижує витрати часу. Більшість сучасних рушіїв, таких як Unity чи Unreal Engine, мають широкі можливості кросплатформеної підтримки, дозволяючи розробникам працювати з різними типами пристроїв без необхідності переписувати код. Це робить процес розробки значно ефективнішим і доступнішим.

Крім розробки ігор, рушії активно застосовуються в інших галузях, що потребують високоякісної графічної візуалізації: симуляції, технічного моделювання, візуалізації даних, створення інтерактивних інтерфейсів тощо. Їх популярність обумовлена відкритістю та гнучкістю — багато рушіїв мають відкритий вихідний код, що дозволяє адаптувати їх під специфічні потреби проєктів. Завдяки багатофункціональним інструментам, ігрові рушії дозволяють створювати графічно насичені додатки, що включають ефекти освітлення, тіні, анімацію, симуляцію фізики та багато іншого.

Фізичні рушії є окремим елементом ігрових рушіїв, що відповідає за імітацію фізичних процесів. Вони здатні моделювати широкий спектр фізичних явищ: динаміку твердих та деформованих тіл, рідин і газів, поведінку тканин, канатів тощо. Наявність достовірної фізики підвищує реалізм ігрового процесу та дозволяє гравцям взаємодіяти з навколишнім середовищем у природний спосіб.

Залежно від жанру гри, необхідність у фізичній симуляції може варіюватися. Деякі ігри використовують лише базові механіки — прості зіткнення та рух об’єктів, які не потребують складних обчислень. В інших, як-от шутерах чи екшен-іграх від третьої особи, фізика відіграє центральну роль, наприклад, у моделюванні ragdoll-анімацій, руйнування об’єктів або взаємодії з оточенням (Half-Life 2, Psi-Ops тощо).

Спочатку фізичні обчислення здійснювалися виключно за рахунок центрального процесора, що обмежувало їхню точність та кількість одночасно симульованих об’єктів. Із впровадженням апаратної підтримки — зокрема, технології Nvidia PhysX, — з’явилася можливість використовувати ресурси відеокарти для обробки фізичних ефектів у реальному часі. Це дозволило підвищити деталізацію симуляції, зменшити навантаження на CPU та покращити загальну продуктивність.

Фізичні рушії виконують дві ключові функції: виявлення колізій між об’єктами та розрахунок фізичних реакцій — сили, переміщення, обертання тощо. Для точного й ефективного виявлення зіткнень використовуються спеціальні алгоритми просторової оптимізації, зокрема розбиття сцени на сектори за допомогою quadtree (для 2D) чи octree (для 3D).

Системи виявлення зіткнень зазвичай ґрунтуються на дискретній перевірці точок взаємодії, через що швидкі об’єкти можуть минати один одного без реєстрації зіткнення. Щоб запобігти таким неточностям, деякі рушії впроваджують методику неперервного виявлення зіткнень (Continuous Collision Detection, CCD), яка передбачає перевірку взаємодії між витягнутими об’ємами, що відображають траєкторію руху об'єкта за певний часовий інтервал. Це забезпечує більш точне виявлення зіткнень навіть за високих швидкостей об’єктів.

Більшість фізичних рушіїв підтримує моделювання твердої матерії — об’єктів, що зберігають свою форму (наприклад, стіни, столи, цегла). Завдяки хорошому балансу між точністю та продуктивністю фізика твердого тіла широко застосовується у відеоіграх. Для опису форм таких об’єктів

використовують як прості геометричні примітиви (куби, сфери, циліндри, конуси), так і складні полігони — опуклі або неопуклі. У складних випадках об’єкт може бути представлений як набір примітивів для наближеної симуляції, що дозволяє зменшити обчислювальні витрати.

Фізичні характеристики твердого тіла визначаються такими параметрами, як маса, коефіцієнти тертя (спокійного і динамічного), пружність тощо. Його рух описується за допомогою лінійної та обертальної швидкостей і прискорень, які можуть задаватися безпосередньо або виникати під дією прикладених сил та імпульсів. Усе це забезпечує реалістичну динаміку об'єктів у грі.

При моделюванні персонажів часто використовують так звану ляльковy фізику (ragdoll), яка базується на поєднанні твердих тіл і спеціальних з'єднань — джоінтів (joints) або обмежень (constraints). Такі з’єднання обмежують взаємне переміщення або обертання тіл. Наприклад, ball-joint дозволяє вільне обертання по всіх осях, тоді як hinge-joint обмежує рух до однієї осі. Ragdoll-анімації активно використовують ці типи з’єднань для досягнення природного руху тіла при падінні або зіткненні з об'єктами.

Сфера використання джоінтів набагато ширша за симуляцію падіння персонажів. Крім моделі твердої матерії, сучасні рушії часто включають додаткові компоненти: підтримку руху транспортних засобів, симуляцію води, рідин, тканин, частинок, інтерфейсів керування персонажами, анімаційні системи тощо. Це розширює можливості створення складних інтерактивних світів.

Основні етапи створення фізично-орієнтованого ігрового рушія включають:

- визначення фізичних законів (механіки, термодинаміки, електродинаміки), які будуть застосовуватись у грі;
- розробку алгоритмів для симуляції цих процесів;
- реалізацію алгоритмів у програмному середовищі;
- тестування та оптимізацію рушія;

- повноцінну розробку гри з графікою, звуком та іншими елементами.

Розробка ігрових рушіїв, заснованих на фізичних процесах, вимагає детального врахування різних типів взаємодій: між твердими тілами, статичними та динамічними об’єктами, а також процесів деформації та зміни об’ємів. Саме для цього й створюються системи фізичного моделювання, що реалізують принципи фізики твердого тіла.

Одним із найпопулярніших рішень для створення фізично обґрунтованих ігрових середовищ є Unity (Unity3D) — кросплатформене середовище розробки, яке підтримує широкий спектр платформ: Windows, macOS, Linux, Android, iOS, Xbox, PlayStation тощо. Його архітектура побудована на компонентному підході, що дозволяє розробникам зручно конструювати, налаштовувати й комбінувати різні елементи ігрової сцени. Це робить Unity зручним як для початківців, так і для досвідчених розробників, адже дозволяє швидко реалізовувати складні ігрові механіки, завдяки великому набору вбудованих інструментів та можливості налаштування фізики і поведінки об'єктів в ігровому середовищі.

Unity підтримує кілька мов програмування, зокрема C#, JavaScript (у старіших версіях) та Boo, що відкриває доступ до інструментів широкому колу користувачів. Потужна й активна спільнота Unity надає навчальні матеріали, відеоуроки, безкоштовні ресурси та плагіни, що значно полегшують старт у розробці. Безкоштовна версія рушія дозволяє експериментувати й створювати ігрові проєкти без фінансових витрат, що особливо корисно для студентів, інді-розробників і невеликих студій.

Unity підтримує створення як 2D-, так і 3D-ігор, охоплюючи різноманітні жанри — від платформерів до шутерів і масштабних рольових ігор. Крім того, рушій активно використовується у створенні інтерактивних застосунків, симуляторів, додатків доповненої та віртуальної реальності (AR/VR). Прикладами успішних ігор на Unity є Endless Legend, Endless Space, Temple Run, Monument Valley та багато інших.

Іншим потужним рушієм є Unreal Engine, розроблений компанією Epic Games. Це висококласне середовище, популярне серед професійних студій завдяки своїм перевагам, а саме:

- багатофункціональність – рушій підтримує створення ігор різного жанрового спрямування та складності, від інді-проєктів до AAA-ігор;
- кросплатформність – можливість розробки під Windows, Mac, Linux, iOS, Android, Xbox, PlayStation, Nintendo Switch, а також для AR та VR-платформ;
- візуальні можливості – реалізовано передові ефекти: реалістичне освітлення, динамічна вода, частинки (дим, вогонь, туман), тіні тощо, що дозволяють створювати неймовірно детальні й живі ігрові світи;
- висока продуктивність – рушій оптимізовано для швидкої обробки складних графічних сцен та високої кількості об'єктів в реальному часі;
- Blueprints – унікальна система візуального програмування, що дозволяє розробникам створювати логіку гри без необхідності глибоких знань мови C++.

Серед відомих ігор на Unreal Engine – Tom Clancy’s Splinter Cell, BioShock, Might & Magic Heroes VII та інші хітові проєкти. У сфері фізичного моделювання існує низка популярних фізичних рушіїв, які інтегруються в ігрові рушії для забезпечення реалістичної симуляції:

- PhysX – фізичний рушій від NVIDIA, що єдиний має апаратну підтримку для реалізації фізичних ефектів. Це найпоширеніший рушій, що активно використовується в індустрії, зокрема в таких відомих іграх, як Batman: Arkham Asylum і Warframe;

Рисунок 2.1 – Фізичний рушій PhysX

Джерело: сформовано автором на основі [17]

• Bullet Physics Library – відкритий рушій, популярний серед інді-розробників та у вільному ПЗ, відомий своєю швидкістю та гнучкістю в налаштуваннях. Використовується у таких іграх, як Grand Theft Auto V;

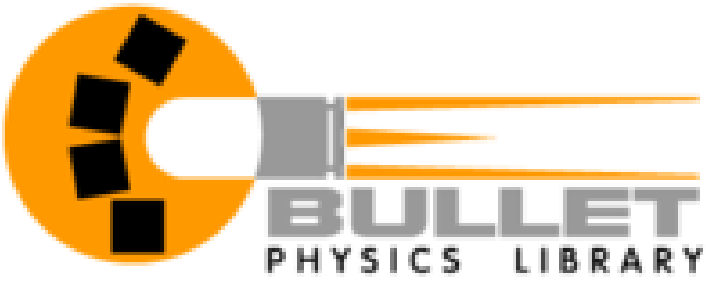


Рисунок 2.2 – Фізичний рушій Bullet Physics Library

Джерело: сформовано автором на основі [17]

• Open Dynamics Engine – ще один відкритий фізичний рушій, що спеціалізується на симуляції твердих тіл і суглобів, здебільшого для 3D-симуляцій. Підходить для розробки спортивних ігор або фізичних головоломок;

Рисунок 2.3 – Фізичний рушій Open Dynamics Engine

Джерело: сформовано автором на основі [17]

• Box2D – легкий і ефективний рушій, призначений для 2D-симуляції фізики твердих тіл, ідеальний для мобільних і казуальних ігор. Його часто використовують у таких іграх, як Angry Birds;

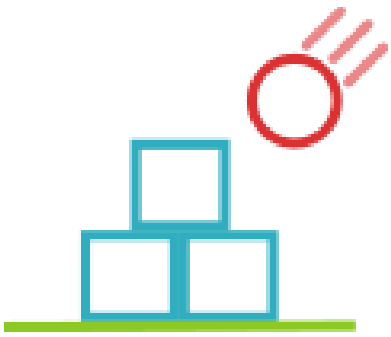


Рисунок 2.4 – Фізичний рушій Box2D

Джерело: сформовано автором на основі [17]

• Havok Physics – потужний комерційний фізичний рушій, який використовується для складних симуляцій у режимі реального часу. Він активно підтримує усі актуальні платформи: ПК, ігрові консолі, гібридні пристрої, і навіть мобільні платформи.

Рисунок 2.5 – Фізичний рушій Havok Physics

Джерело: сформовано автором на основі [17]

Havok Physics був застосований у понад 150 іграх. Частково його технології інтегровані до рушія Source, що використовується в іграх, таких як Half-Life 2 та Portal. Крім того, Havok використовується в таких професійних програмах, як Autodesk 3ds Max та Maya, для моделювання і анімації фізичних процесів. Havok також має широкий спектр застосування в реалістичних симуляторах водіння й авіатренажерах, де важливе точне відтворення фізики руху і зіткнень. Крім рушія для фізики, компанія Havok розробила також графічний рушій Havok Vision Engine та систему штучного інтелекту Havok AI, що розширюють потенціал рушія в напрямку створення складних і інтерактивних віртуальних світів. Ключовими складниками будь-якого фізичного рушія залишаються система колізійного виявлення, симуляція рідин, деструкції та реакції на сили, що дозволяють створювати динамічну і правдоподібну поведінку об'єктів у грі.

2.2 Детальний опис ігрового штучного інтелекту

Ігровий штучний інтелект (ІШІ) відіграє ключову роль у формуванні ігрового процесу, забезпечуючи адаптивність, реалістичність і динамічну взаємодію між гравцем та ігровим світом. Використання ігрового ШІ дозволяє створювати персонажів, які реагують на дії гравця, змінюють свою поведінку та демонструють інтелектуальну діяльність. Такі ігри, як The Last of Us Part II, Red

Dead Redemption 2 та The Legend of Zelda: Breath of the Wild, демонструють, як передові технології ШІ створюють складні сценарії взаємодії, що сприяють зануренню у віртуальний світ.

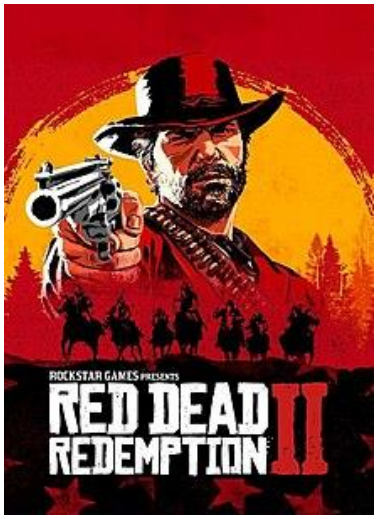


Рисунок 2.6 – гра Red Dead Redemption II

Джерело: сформовано автором на основі [18,19,20]

У комп'ютерних іграх штучний інтелект використовується для створення чуйної, адаптивної або інтелектуальної поведінки головним чином у неігрових персонажів, подібних до людського інтелекту. Штучний інтелект був невід'ємною частиною комп'ютерних ігор з моменту їх створення в 1950-х роках. Штучний інтелект у комп'ютерних іграх – це окрема підгалузь, яка відрізняється від академічного ШІ. Це покращує досвід гравця, а не машинне навчання чи прийняття рішень.

Ігровий ШІ не є «чистим» ШІ в академічному розумінні. Його основна мета – створити ілюзію розумної поведінки, що відповідає динаміці гри. На відміну від самонавчальних нейромереж, ігровий ШІ здебільшого використовує заздалегідь визначені алгоритми, сценарії та евристичні методи. Наприклад, у стелс-іграх, таких як Hitman та Dishonored, неігрові персонажі можуть змінювати маршрути патрулювання або підвищувати рівень тривоги, якщо помічають підозрілу активність.

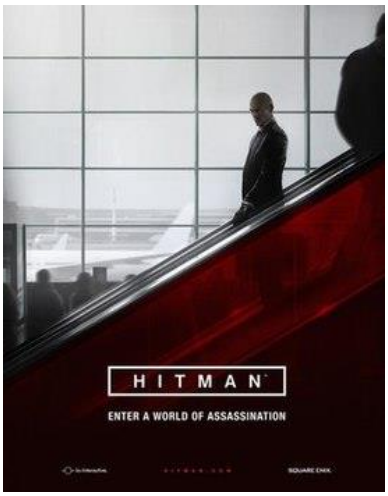


Рисунок 2.7 – гра Hitman

Джерело: сформовано автором на основі [18,19,20]

У The Legend of Zelda: Breath of the Wild ігровий ШІ формує середовище, яке динамічно реагує на вибір гравця, створюючи багатогранний геймплей. З технічної точки зору ранні ігри використовували статичні алгоритми, тоді як сучасні застосовують складніші системи, такі як нейронні мережі та генеративні моделі. Технічна реалізація включає аналіз змінних у реальному часі, патерни поведінки та графи станів.

Рисунок 2.8 – гра The Legend of Zelda: Breath of the Wild

Джерело: сформовано автором на основі [18,19,20]

Розвиток ігрового ШІ можна розділити на кілька ключових етапів:

- 1970-1980-ті роки (Перші аркадні ігри): Space Invaders (1978) представив базову адаптивність – вороги прискорювали рух при знищенні

союзників. Pac-Man (1980) продемонстрував перші елементи штучного інтелекту, де поведінка кожної примари визначалася окремими алгоритмами.

Рисунок 2.9 – гра Pac-Man

Джерело: сформовано автором на основі [18,19,20]

• 1990-ті роки (Стратегічні ігри): Такі ігрові серії, як Quake, Doom та Warcraft вперше використали алгоритми пошуку шляхів, що дозволило ворогам ефективно переслідувати гравця, а також обходити перешкоди.

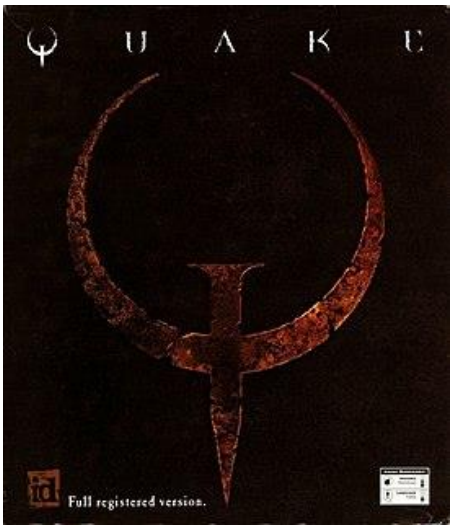


Рисунок 2.10 – гра Quake

Джерело: сформовано автором на основі [18,19,20]

• 2000-ні роки (Розвиток емоційного ШІ): Використання алгоритму A* у Age of Empires покращило навігацію в стратегіях. Fable (2004) дозволив NPC змінювати ставлення до гравця залежно від його рішень.

Рисунок 2.11 – гра Fable

Джерело: сформовано автором на основі [18,19,20]

• Сучасність: Складні сценарії взаємодії NPC у Cyberpunk 2077 або Red Dead Redemption 2 створюють реалістичні світи, де персонажі мають власні ритми життя та індивідуальну поведінку.

Рисунок 2.12 – гра Cyberpunk 2077

Джерело: сформовано автором на основі [18,19,20]

Ефективний ігровий ШІ базується на кількох основних принципах:

• Адаптивність: Наприклад, у Far Cry 5 вороги змінюють тактику залежно від стилю гри гравця.

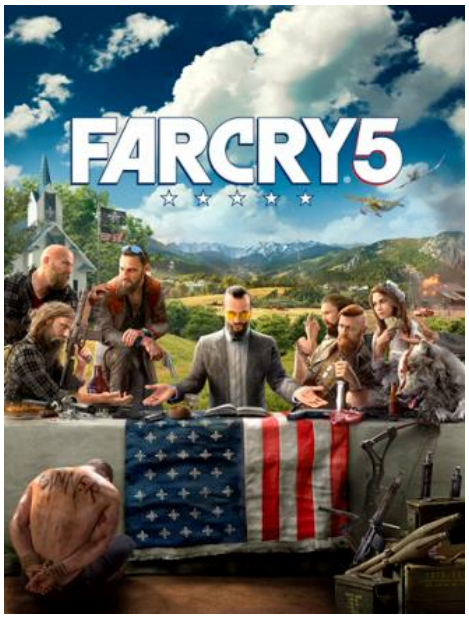


Рисунок 2.13 – гра Far Cry 5

Джерело: сформовано автором на основі [18,19,20]

• Прогнозованість: У серії Dark Souls вороги використовують анімації атак як підказки для гравця.

Рисунок 2.14 – гра Dark Souls

Джерело: сформовано автором на основі [18,19,20]

• Природність: У Red Dead Redemption 2 NPC виконують повсякденні дії, такі як розмови або робота, що створює ілюзію живого світу. Також NPC реагують на бруд або кров на одязі гравця.

Рисунок 2.15 – гра Red Dead Redemption II

Джерело: сформовано автором на основі [18,19,20]

Технічні аспекти реалізації ігрового ШІ охоплюють:

• Finite State Machines (FSM) – алгоритми для зміни станів персонажів (наприклад, патрулювання, переслідування, атака);

• Behavior Trees – використовуються для складної поведінки, як у Horizon Zero Dawn, де роботи-вороги адаптуються до гравця;

Рисунок 2.16 – гра Horizon Zero Dawn

Джерело: сформовано автором на основі [18,19,20]

• Системи сповіщень – у The Last of Us Part II вороги передають інформацію про позицію гравця за допомогою голосових сигналів.

Рисунок 2.17 – гра The Last of Us Part II

Джерело: сформовано автором на основі [18,19,20]

Ігровий ШІ поділяється на дві основні категорії:

• Периферійний ШІ: Виконує локальні завдання, наприклад, у FIFA NPC реагують на положення м’яча.

Рисунок 2.18 – гра EA Sports FC 24

Джерело: сформовано автором на основі [18,19,20]

• Центральний ШІ: Відповідає за прийняття стратегічних рішень, як у XCOM: Enemy Unknown, де вороги аналізують позиції гравця та адаптують свої тактики, або як в Civilization VII він приймає стратегічні рішення щодо розвитку країни.

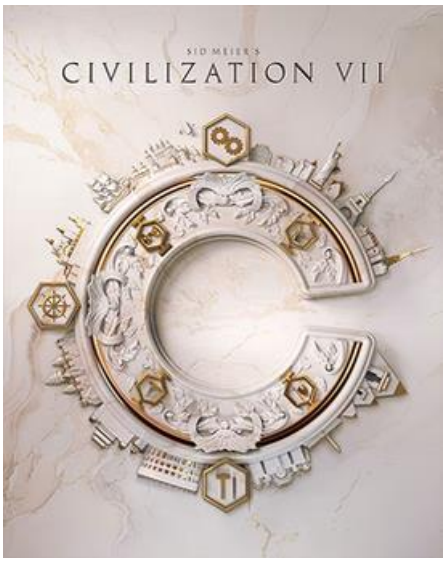


Рисунок 2.19 – гра Civilization VII

Джерело: сформовано автором на основі [18,19,20]

Ігровий ШІ виконує такі ключові завдання:

• Навігація: У Assassin’s Creed система паркуру дозволяє персонажам плавно переміщуватися у відкритому світі. У Hitman вороги обходять перешкоди через алгоритм A*.

Рисунок 2.20 – гра Assassin’s Creed Shadows

Джерело: сформовано автором на основі [18,19,20]

• Прийняття рішень: У StarCraft II вороги адаптуються до тактик гравця, змінюючи свою стратегію. у XCOM ШІ аналізує позицію гравця, укриття та тактичні можливості.

Рисунок 2.21 – гра StarCraft II: Wings of Liberty

Джерело: сформовано автором на основі [18,19,20]

• Поведінка персонажів: У The Sims персонажі демонструють емоції та соціальні взаємодії, а в The Elder Scrolls V: Skyrim - дотримуються рутини дня, а також можуть змінювати поведінку залежно від погоди, часу доби та рішень гравця.

Рисунок 2.22 – гра The Elder Scrolls V: Skyrim

Джерело: сформовано автором на основі [18,19,20]

Серед найпоширеніших алгоритмів для ігрового ШІ є:

- Алгоритм A* – Використовується для пошуку оптимальних маршрутів (наприклад, у The Witcher 3 або StarCraft II).

Рисунок 2.23 – гра The Witcher 3: Wild Hunt

Джерело: сформовано автором на основі [18,19,20]

- Дерева рішень – У Far Cry визначають реакції ворогів на дії гравця.

Рисунок 2.24 – гра Far Cry 6

Джерело: сформовано автором на основі [18,19,20]

• Процедурна генерація – У Minecraft унікальні світи створюються через Perlin Noise.

Рисунок 2.25 – гра Minecraft

Джерело: сформовано автором на основі [18,19,20]

• Нейронні мережі – У AI Dungeon використовуються GPT-подібні моделі для динамічного створення діалогів.

Рисунок 2.26 – гра AI Dungeon

Джерело: сформовано автором на основі [18,19,20]

Майбутнє ігрового ШІ зосереджене на інтеграції генеративних моделей, таких як ChatGPT, для створення більш природної взаємодії між гравцем і NPC. Наприклад, у Cyberpunk 2077, наступні покоління NPC можуть створювати унікальні діалоги залежно від дій гравця. Завдяки впровадженню машинного навчання та нейромереж ігровий ШІ стає дедалі складнішим, що відкриває нові можливості для створення реалістичних і адаптивних ігрових світів.

Розглядаючи алгоритми та методи ігрового ШІ, слід детальніше їх проаналізувати, зважаючи на те, що частина алгоритмів та методів ігрового ШІ разом з охопленням технічних аспектів реалізації ігрового ШІ були раніше описані:

➢ *Алгоритми навігації: пошук шляху та адаптивність*

• Алгоритм A*

Алгоритм A* є стандартом у навігації ігор завдяки ефективності та точності пошуку оптимального маршруту. Він комбінує реальну відстань до цілі з евристичною оцінкою, що дозволяє швидко знаходити найкоротший шлях.

Приклади використання:

• StarCraft II – юніти використовують A* для обходу перешкод і вибору найшвидшого шляху.

• The Witcher 3 – персонажі переміщуються великими відкритими просторами, уникаючи природних перешкод (скелі, дерева).

Технічна реалізація:

1. Розбиття світу: Ігрова карта поділяється на вузли (граф).

2. Функція оцінки: Розраховуються відстані G (від початкової точки) та H (до цілі).

3. Пошук шляху: Алгоритм знаходить маршрут із мінімальною сумою G + H.

Удосконалена версія – Динамічний A* – застосовується у Company of Heroes, де маршрути змінюються через руйнування будівель чи вибухи.

• Навігаційні сітки (Navigation Mesh)

Цей метод дозволяє персонажам природно пересуватися складними середовищами, забезпечуючи реалістичність руху.

Приклади використання:

• Assassin's Creed – персонажі використовують паркур, орієнтуючись на навігаційну сітку.

• Horizon Zero Dawn – роботи адаптують рух, враховуючи складний рельєф.

Технічна реалізація:

1. Попередня генерація: Сітка охоплює всі доступні для переміщення ділянки.

2. Динамічні зміни: Сітка оновлюється в реальному часі, якщо змінюється середовище.

3. Розрахунок маршруту: Поєднується з A* для оптимального пошуку.

У Cyberpunk 2077 навігаційні сітки враховують рух пішоходів та транспортні затори.

➢ *Алгоритми поведінки: від машин станів до дерев поведінки*

• Машини станів (Finite State Machines, FSM)

FSM використовуються для управління поведінкою персонажів через зміну станів залежно від подій.

Приклади використання:

• Half-Life – вороги переходять від патрулювання до атаки при виявленні гравця.

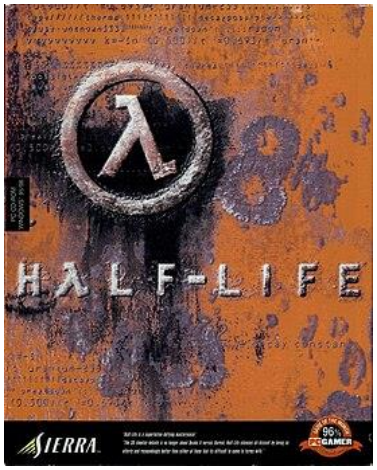


Рисунок 2.27 – гра Half-Life

Джерело: сформовано автором на основі [18,19,20]

• Doom (2016) – демони змінюють поведінку на основі дистанції до гравця.

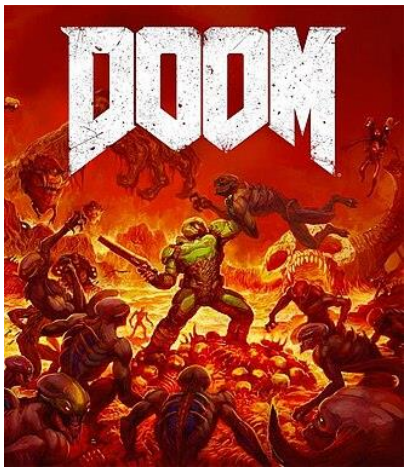


Рисунок 2.28 – гра Doom

Джерело: сформовано автором на основі [18,19,20]

Технічна реалізація:

1. Визначення станів (наприклад, "патруль", "атака", "втеча").

2. Умови переходів (перемикання між станами при певних умовах).

3. Оновлення стану (реакція на події в реальному часі).

• Дерева поведінки (Behavior Trees)

Цей підхід забезпечує складнішу ієрархію прийняття рішень.

Приклади використання:

• Far Cry 5 – вороги можуть викликати підкріплення, займати укриття чи використовувати гранати.

• The Last of Us Part II – NPC координують свої дії, змінюючи тактику відповідно до оточення.

Технічна реалізація:

1. Структура дерева: Вузли відповідають за конкретні дії, а гілки – за умови.

2. Перевірка умов: ШІ аналізує поточну ситуацію від кореня дерева до листків.

3. Прийняття дій: Виконання певної поведінки при виконанні умов.

У Horizon Zero Dawn дерева поведінки поєднуються з машинним навчанням, щоб вороги адаптувалися до стилю гри гравця.

➢ *Генеративний контент: процедурна генерація*

- Шум Перліна (Perlin Noise)

Використовується для створення природних ландшафтів та текстур у відеоіграх.

Приклади використання:

- Minecraft – формування унікальних світів із варіативним рельєфом.
- No Man's Sky – генерація цілих планет, флори та фауни.

Технічна реалізація:

1. Генерація шуму: Псевдовипадкові значення визначають структуру середовища.
2. Формування ландшафту: Перетворення значень у висоти, текстури чи об’єкти.
3. Оптимізація: Дані зберігаються у стислому форматі для ефективного використання ресурсів.

У Subnautica шум використовується для створення унікальних підводних екосистем.

➢ *Соціальні взаємодії та емоційна симуляція*

- Алгоритми емоційної симуляції

Моделі емоцій використовуються для створення більш реалістичних NPC.

Приклади використання:

- The Sims 4 – персонажі змінюють поведінку залежно від емоційного стану.
- Detroit: Become Human – NPC емоційно реагують на вибори гравця.

Технічна реалізація:

1. Вхідні параметри: Аналіз дій гравця, взаємодій із NPC.
2. Обробка емоцій: Визначення позитивних або негативних впливів.
3. Прояв емоцій: Зміни у міміці, діалогах, діях персонажів.

У Red Dead Redemption 2 NPC реагують навіть на дрібні зміни, такі як одяг чи поведінка гравця.

➢ *Перспективи розвитку ігрового ШІ*

У майбутньому можна очікувати інтеграції таких технологій:

- Генеративний ШІ: Використання GPT-моделей для створення реалістичних діалогів NPC.
- Самонавчальний ШІ: NPC у багатокористувацьких іграх адаптуватимуться до стилю гри різних гравців.
- Синтез соціальної поведінки: NPC отримають можливість вести переговори, конфліктувати та формувати альянси у реальному часі.

Ігровий ШІ продовжує розвиватися, підвищуючи рівень занурення у віртуальні світи та роблячи NPC дедалі реалістичнішими.

2.3 Огляд піджанрів жанру стратегія

Як зазначалося у вступі данної кваліфікаційної роботи, стратегічна комп'ютерна гра, або просто «Стратегія», є одним із основних жанрів відеоігор, у якому для досягнення перемоги необхідно застосовувати стратегічне мислення. У таких іграх гравець зазвичай управляє не окремим персонажем, а великими групами, наприклад, займається будівництвом міста чи командує військовими силами під час кампаній.

Хоча елементи стратегії можуть зустрічатися в іграх інших жанрів, саме стратегічні ігри роблять акцент на плануванні, логістиці та управлінні ресурсами. Вони поділяються на дві основні групи: покрокові стратегії та стратегії в реальному часі. Окрім цього, існують численні піджанри, які включають додаткові механіки, такі як тактичні аспекти, дипломатія, економічне управління та дослідження.

Враховуючи широкий спектр механік і піджанрів, стратегічні ігри мають багату історію розвитку, що розпочалася ще у 1960-х роках. Саме в цей період з’явилися перші спроби створити комп’ютерні стратегії, які поступово еволюціонували, додаючи нові елементи та ускладнюючи геймплей.

Перші стратегічні ігри для комп’ютерів з’явилися ще в 1960-х роках. Однією з перших стала The Sumerian Game (1964), яка була текстовим економічним симулятором управління містом. У 1970-х роках з’явилися перші

адаптації настільних стратегій для відеоігор, наприклад, Invasion (1972) для приставки Magnavox Odyssey. З поширенням мікрокомп'ютерів, таких як TRS-80 та Apple II, почався активний розвиток стратегічних ігор, серед яких виділялися Computer Bismarck (1980) і численні варгейми від компаній SSI та Avalon Hill.

У 1980-х роках стратегічні ігри ставали дедалі складнішими, додаючи елементи симуляції та економічного управління. Наприклад, Eastern Front (1941) (1981) враховувала погодні умови та моральний дух солдатів, а Reach for the Stars (1983) заклала основи жанру 4X-стратегій. Японська компанія Koei випускала глобальні історичні стратегії, такі як Nobunaga's Ambition та Romance of the Three Kingdoms, де окрім військових кампаній потрібно було керувати економікою. 1989 року з'явилася Herzog Zwei, яку вважають першою стратегією в реальному часі, що вплинула на подальший розвиток жанру, а Dune II (1992) від Westwood Studios закріпила основні механіки RTS.

У 1990-х і 2000-х роках стратегічні ігри пережили пік популярності, поєднуючи нові технології та інноваційні ігрові механіки. Sid Meier's Civilization (1991) заклала основи сучасних глобальних стратегій, а Warcraft (1994) та Command & Conquer (1995) сформували стандарт RTS. У 2000-х роках жанр зазнав змін через перехід до тривимірної графіки та зміщення інтересів гравців. Серія Total War поєднала покрокове управління імперією з реалістичними битвами, а Warcraft III (2002) стала основою для жанру MOBA. Незважаючи на спад популярності RTS, 4X-стратегії, такі як Civilization IV (2005) і проекти Paradox Interactive, залишалися актуальними. У 2010-х роках MOBA-ігри, зокрема League of Legends і Dota 2, стали новими флагманами жанру, тоді як XCOM: Enemy Unknown (2012) вдало переосмислила тактичні покрокові стратегії.

Таким чином, розвиток стратегічних ігор пройшов довгий шлях — від простих текстових симуляторів до складних багаторівневих систем з високим рівнем деталізації. З огляду на таке розмаїття ігор, у межах жанру сформувалася низка піджанрів, кожен з яких акцентує увагу на певних аспектах

геймплею або має унікальні особливості. Нижче розглянемо основні піджанри стратегічних ігор та їхні характерні риси.

Покрокова стратегія (англ. Turn-Based Strategy, TBS) — піджанр стратегічних відеоігор, у якому ігровий процес поділений на чіткі часові відрізки, або "ходи" (кроки), під час яких гравці по черзі виконують свої дії. Такий підхід контрастує зі стратегіями в реальному часі, де всі дії відбуваються одночасно.

Основною особливістю покрокових стратегій є дискретність ігрового процесу — кожен хід завершується тільки після того, як гравець виконає свої дії. Один хід може охоплювати значний проміжок часу в ігровому світі, протягом якого гравець приймає рішення щодо управління містами, ресурсами, арміями тощо.

У більшості ігор цього типу гравці діють по черзі, як у класичних настільних іграх на кшталт шахів або «Ризику». Прикладами покрокових стратегій є Sid Meier's Civilization та Heroes of Might and Magic.

Окремий різновид покрокових стратегій — ігри з одночасними ходами (англ. tick-based strategy), у яких усі гравці планують дії паралельно, а хід завершується після підтвердження з боку кожного з них. Яскравим прикладом є Age of Wonders.

Деякі відеоігри поєднують елементи покрокових і реалістичних стратегій. Наприклад, серії Lords of the Realm і Total War застосовують покрокову модель на глобальному рівні, а бойові дії відбуваються в реальному часі. У таких іграх часто передбачена можливість автоматичного розрахунку результатів бою для уникнення рутинних зіткнень.

Покрокові стратегії мають витоки в настільних варгеймах. Однією з перших була текстова гра Hamurabi (кінець 1960-х), де гравець керував ресурсами Шумерської держави. У 1970-х з’явилися перші багатокористувацькі покрокові ігри, як-от Empire.

У 1980-х роках на ринку з’явилися серйозні варгейми, зокрема Computer Bismarck (1980) від SSI — перша стратегічна гра для мікрокомп’ютерів.

Відомою також стала Eastern Front (1941) (1981), яка мала великий комерційний успіх. У 1980-х вийшли перші космічні покрокові стратегії — Andromeda Conquest і Reach for the Stars, що заклали основи 4X-піджанру.

Японська компанія Koei випустила серії Nobunaga's Ambition (1983) і Romance of the Three Kingdoms (1985), які стали популярними в Азії та продовжують існувати досі.

У 1990-х покрокові стратегії досягли нових висот. Серед важливих релізів — Warlords (1989), яка підтримувала режим «гарячого місця» (hot seat), Civilization (1991), що започаткувала успішну серію глобальних стратегій, і X-COM: UFO Defense (1993), яка поєднувала стратегію, тактику та рольові елементи. Успішними були також серії Panzer General, Steel Panthers та Heroes of Might and Magic (1995).

Кінець 1990-х і початок 2000-х позначилися розвитком гібридних проєктів, таких як Age of Wonders (1999) та серія Total War (з 2000 року), які поєднали глибоку стратегічну складову з тактичними боями в реальному часі.

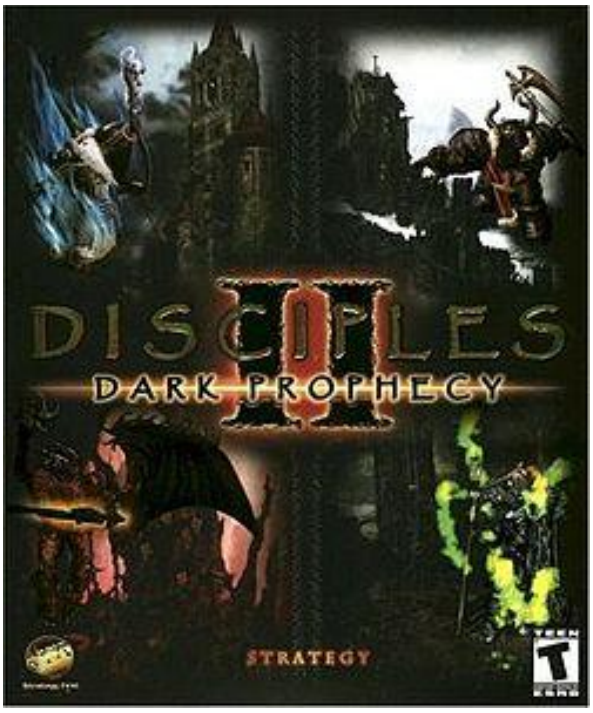


Рисунок 2.29 – гра Disciples II: Dark Prophecy

Джерело: сформовано автором на основі [23,24,31,32]

Стратегія у реальному часі (real-time strategy, RTS) — це піджанр стратегічних відеоігор, у якому всі дії гравця та опонента відбуваються одночасно, без поділу на ходи. На відміну від покрокових стратегій, де гравці виконують дії послідовно, в RTS уся гра триває безперервно, що додає динаміки й вимагає від гравця швидкого прийняття рішень.

Жанр RTS сформувався внаслідок багаторічного еволюційного розвитку ігрової індустрії, починаючи ще з 1980-х років. Попри те, що перші ігри лише частково відповідали критеріям RTS, саме в них зароджувалися основи жанру: керування військами, управління ресурсами та бойові дії. Серед ранніх проєктів можна згадати Stonkers (1983) для ZX Spectrum — одну з перших спроб реалізувати елементи стратегічного командування у режимі, наближеному до реального часу, та Herzog Zwei (1989), яка поєднувала активне управління бойовими одиницями з тактичними елементами й вважається прямим попередником сучасних RTS.

Ключовим поворотним моментом в історії жанру стала поява Dune II: The Building of a Dynasty (1992) від Westwood Studios. Саме ця гра заклала основоположні механіки жанру: видобуток ресурсів, будівництво бази, створення армії та бойові дії в реальному часі. Dune II вважається першою "класичною RTS" і сформувала зразок для численних наступних проєктів.

У середині 1990-х жанр стрімко розвивався завдяки таким проєктам, як Warcraft: Orcs & Humans (1994), Warcraft II: Tides of Darkness (1995), Command & Conquer (1995) та Total Annihilation (1997). Ці ігри вдосконалили геймплей, розширили сюжетну складову, запровадили мультиплеєр, унікальні раси, фізику бою, тривимірну графіку та інтелектуальні системи керування юнітами.

Піком популярності жанру стала поява StarCraft (1998), яка вразила ідеально збалансованими расами та заклала підвалини професійної кіберспортивної сцени, особливо в Південній Кореї. У наступні роки жанр зміцнив свої позиції завдяки франшизам Age of Empires, Stronghold, Empire Earth, Company of Heroes та Warhammer 40,000: Dawn of War.

Водночас деякі проєкти вийшли за межі традиційної формули RTS, наголошуючи на різних аспектах геймплею: економіці (The Settlers, Zeus: Master of Olympus), тактиці (Ground Control, Myth, World in Conflict), чи обмеженій побудові баз (Dawn of War II). Інші ж поєднували RTS із елементами інших жанрів: наприклад, Dungeon Keeper вміщував елементи симулятора бога, а Total War поєднував глобальну покрокову стратегію з тактичними битвами в реальному часі.

Типова RTS-гра має специфічний інтерфейс: верхній вигляд (часто в ізометрії), розділений на ігрове поле та панель управління. Гравець керує юнітами, які зазвичай діляться на бойові та допоміжні, а сам ігровий процес охоплює кілька ключових елементів:

- Збір ресурсів — через робітників або контроль над точками (як у Company of Heroes);
- Будівництво бази — споруди дають доступ до нових юнітів, технологій або ресурсів;
- Формування армії — створення військ для захоплення територій і знищення ворога;
- Бойові дії — атаки, оборона, контроль карти, використання героїв;
- Туман війни — обмеження видимості на карті, що стимулює розвідку.

Зазвичай RTS мають "дерево технологій", ліміт на кількість юнітів та специфічні механіки управління. Більшість класичних RTS вимагають ефективного поєднання економічної й військової стратегії, а також вміння швидко реагувати на дії противника.

У 2000-х жанр експериментував з тривимірністю (Homeworld, Ground Control), новими механіками видобутку ресурсів (Perimeter), героїчними елементами (Heroes of Annihilated Empires), або навіть повною відмовою від будівництва баз (Dawn of War II). У той же час StarCraft II (2010), R.U.S.E. (2010), Act of Aggression (2015) демонстрували збереження класичних принципів жанру.

У 2010-х жанр зазнав спаду через зміну ігрових трендів та складнощі з монетизацією, зокрема через труднощі впровадження мікротранзакцій. Проте з'явилися нові піджанри, онлайн-RTS, MMO RTS (наприклад, Command & Conquer: Tiberium Alliances), а також хвиля ремастерів класичних ігор: Age of Empires: Definitive Edition, Warcraft III: Reforged, Command & Conquer Remastered Collection.

Попри періоди занепаду, жанр RTS не зник — він адаптувався до сучасних реалій. Сучасні проєкти, як-от Northgard, They Are Billions чи Iron Harvest, демонструють здатність жанру до еволюції. RTS продовжує займати свою нішу, пропонуючи гравцям глибокий, динамічний та стратегічно насичений ігровий досвід.

Рисунок 2.30 – гра Dune II: The Building of a Dynasty

Джерело: сформовано автором на основі [25,26,31,32]

Симулятор менеджменту та будування (англ. construction and management simulation, CMS), або просто симулятор менеджменту, є піджанром стратегічних симуляторів у комп'ютерних іграх. Основна ідея таких ігор полягає у створенні, розвитку та фінансовому управлінні віртуальними спільнотами, організаціями або проектами за умов обмежених ресурсів. Ці ігри зазвичай не мають класичної мети на зразок перемоги над ворогом, як це притаманно стратегіям, а замість цього фокусуються на безперервному процесі побудови та розвитку. Саме тому жанр часто називають управлінськими іграми (management games).

Велике значення в цьому жанрі має економіка. Гравець керує ресурсами, необхідними для будівництва та обслуговування інфраструктури, забезпечуючи постійне зростання і розвиток економіки у заданому середовищі — це може

бути місто, компанія, імперія чи інша організація. Ресурси (гроші, працівники, матеріали тощо) можуть добуватися, обмінюватися, витрачатися, а іноді — трансформуватися (наприклад, з цукру — ром через ферментацію). Вони використовуються або на будівництво нових об’єктів, або на їх утримання, іноді — на демонтаж. Часто реалізовано два типи будівництва: поступове зведення за планом (plan-and-build) або миттєве розміщення об'єктів (purchase and place).

Інтерфейси таких ігор, як правило, складні й нагадують комп’ютерні програми: тут використовуються графіки, таблиці, випадаючі меню, кнопки та помічники, які інформують про стан системи. Через потребу в аналітиці й контролі над складною економікою такі ігри найчастіше мають віконний або панельний інтерфейс, де гравець зазвичай спостерігає за світом з ізометричного ракурсу або через вільну камеру, а прямий контроль над персонажами часто відсутній.

Історія жанру бере початок з гри The Sumer Game для комп’ютера PDP-8, яка працювала у текстовому режимі. Вона стала прототипом майбутніх симуляторів. Проте перший суттєвий крок у формуванні жанру зробила гра Utopia (1982) на приставці Intellivision. У ній гравець керував економікою острова, забезпечував добробут населення і відбивав атаки піратів, що вимагало стратегічного мислення, а не лише швидких реакцій.

У 1983 році компанія Koei випустила історичну гру Nobunaga’s Ambition, де гравець у ролі Оди Нобунаґи мав об'єднати Японію, балансуючи між управлінням економікою, армією та податками. Успіх цієї гри заклав основи жанру симуляторів на консолях і зумовив створення серій Romance of the Three Kingdoms, Bandit Kings of Ancient China та Destiny of an Emperor.

Великим проривом стала гра SimCity (1989), яку вважають першою по-справжньому успішною грою жанру. Вона дозволяла гравцю будувати місто з нуля, керувати бюджетом, задовольняти потреби населення та балансувати між економічним розвитком і громадським добробутом. Гра відмовилася від традиційної системи "перемоги або поразки", натомість пропонуючи відкритий

ігровий процес. SimCity породила безліч сиквелів і натхнення для таких ігор, як SimTower, SimFarm та численних проєктів серії Tycoon. Її творець, Вілл Райт, завдяки цьому здобув визнання як один із найвпливовіших геймдизайнерів в індустрії.

Жанр симуляторів будівництва та управління постійно розвивається, включаючи містобудівні симулятори (Caesar, The Settlers, Anno), економічні моделі (Capitalism) та розважальні проєкти на кшталт Theme Park. Ці ігри приваблюють гравців глибокими системами, економічним моделюванням та можливістю проявити стратегічне мислення в умовах безперервного розвитку.

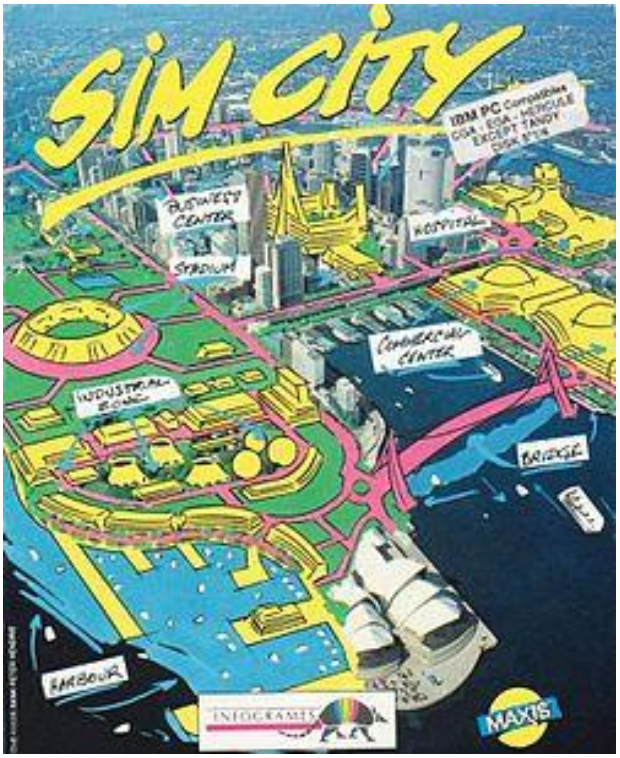


Рисунок 2.31 – гра SimCity

Джерело: сформовано автором на основі [27,28,31,32]

Симулятор містобудування (англ. city-building game) — це піджанр стратегічних симуляторів, в яких гравець займається проєктуванням, будівництвом і управлінням містом. Зазвичай цей жанр відносять до економічних стратегій, оскільки основна увага в таких іграх приділяється розвитку інфраструктури, управлінню ресурсами, підвищенню якості життя мешканців і ефективному функціонуванню міської економіки.

На відміну від багатьох інших жанрів, ігри містобудівного спрямування часто не мають конкретної кінцевої мети — основним завданням є сам процес розвитку й удосконалення міста. Навіть після досягнення умов окремих сценаріїв або кампаній, гравець може продовжити будівництво та управління містом.

Характерними рисами жанру є:

- Фокус на економіці та інфраструктурі: управління бюджетом, покращення житлових умов, розвиток транспорту, забезпечення екологічної стабільності тощо.
- Непряме управління: гравець не керує мешканцями напряму, а задає загальні параметри функціонування міста.
- Рішення логістичних і стратегічних задач: зонування територій, розподіл функціональних зон, планування забудови, енергопостачання та соціальні послуги.
- Другорядність військових елементів: хоча в деяких іграх присутні бойові дії, основна увага приділяється саме економічному розвитку.

Першою грою, що започаткувала жанр містобудівних симуляторів, стала Hamurabi, створена Дугом Дайментом у 1968 році для комп'ютера PDP-8. Незважаючи на примітивну механіку, гра заклала основи управління містом, хоча ще не дозволяла будівництво будівель чи розвиток інфраструктури. Згодом її переписали мовою BASIC, зробивши доступною на інших платформах. Новий етап у розвитку жанру настав у 1981 році з виходом Utopia для консолі Mattel Intellivision, яка вперше поєднала управління містом із елементами реального часу.

Проте справжній прорив відбувся із появою SimCity у 1989 році, яка не лише стала надзвичайно популярною, але й сформувала стандарти жанру, на які орієнтуються розробники й донині. SimCity започаткувала однойменну серію ігор, що залишаються одними з найвідоміших у цій категорії. До містобудівних симуляторів також часто зараховують ігри, що моделюють управління

колоніями, космічними станціями, підприємствами або навіть в’язницями — оскільки їхній геймплей теж базується на принципах розвитку, управління ресурсами та інфраструктурою.

Відомі представники жанру:

- Серія City Building від Impressions Games та Tilted Mill:
- Caesar I–IV (Стародавній Рим)
- Pharaoh, Children of the Nile (Стародавній Єгипет)
- Zeus, Poseidon (Стародавня Греція)
- Emperor: Rise of the Middle Kingdom (Стародавній Китай)
- Серія Anno: Anno 1602, Anno 1503, Anno 1701, Anno 1404, Anno 2070
- Серія Cities XL: City Life, Cities XL 2011–2012, Cities XL Platinum
- Серія SimCity: від оригінальної гри до SimCity (2013)
- Серія Tropico: від Tropico до Tropico 6
- Ігри від Haemimont Games: Glory of the Roman Empire, Imperium Romanum, Grand Ages: Rome

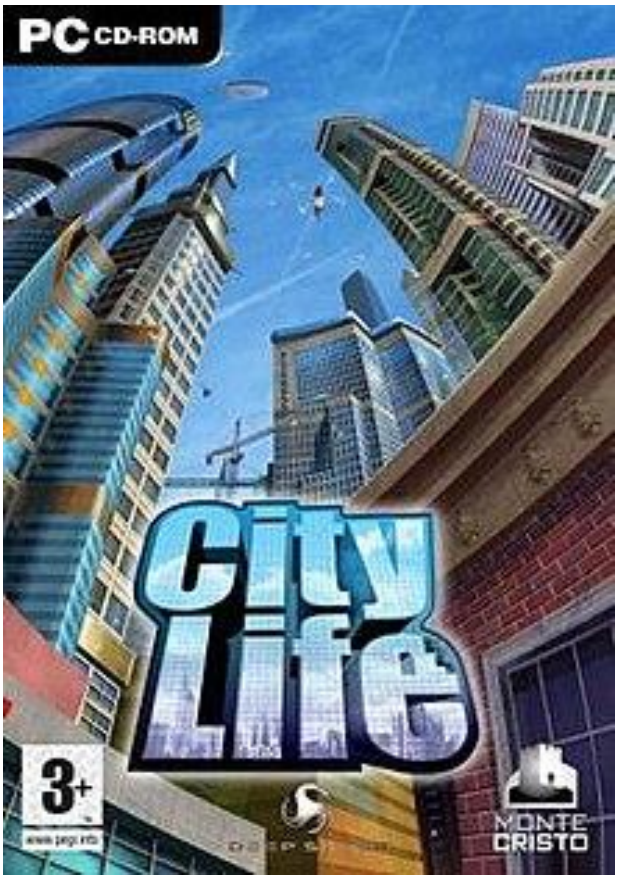


Рисунок 2.32 – гра City Life

Джерело: сформовано автором на основі [29,30,31,32]

Висновки до розділу 2

Проаналізувавши другий розділ, можна зробити стосовно нього висновок, а саме:

- Було розглянуто поняття ігрового рушія як ключового елементу у створенні комп’ютерних ігор, його основні функції та компоненти — графічний, фізичний, звуковий рушії й ігровий ШІ. Підкреслено роль рушія у створенні реалістичного ігрового середовища, моделюванні віртуального світу та взаємодії об'єктів. Окремо акцентовано значення фізичного рушія для симуляції реальних процесів. Проаналізовано еволюцію рушіїв до універсальних платформ з підтримкою VR/AR і мобільних пристроїв. Зазначено, що відкриті рушії, як-от Unity чи Unreal Engine, використовуються й поза ігровою сферою;
- Було розглянуто логіку ігрового ШІ, а саме алгоритми прийняття рішень (дерева рішень, скінченні автомати, дерева поведінки) та поведінкові моделі NPC, що забезпечують їхню адаптивну, реалістичну та стратегічну реакцію на гравця. Ігровий ШІ, на відміну від академічного, є ключовим для створення динамічного ігрового світу, імітуючи інтелектуальну поведінку та поглиблюючи занурення. Проаналізовано еволюцію ігрового ШІ від простих реакцій до складних систем з генеративними моделями та емоційною симуляцією в сучасних іграх, таких як Red Dead Redemption 2, Cyberpunk 2077 та The Legend of Zelda: Breath of the Wild, що формують глибоку взаємодію;
- Було розглянуто, що стратегічні комп'ютерні ігри, як один з ключових жанрів, що вимагає від гравця стратегічного мислення та управління великими групами задля досягнення перемоги, охоплюють широкий спектр механік і піджанрів. Серед основних розрізняють покрокові стратегії, де дії відбуваються почергово, та стратегії в реальному часі, що характеризуються одночасним

прийняттям рішень. Крім того, існують симулятори менеджменту та будівництва, а також симулятори містобудування, які роблять акцент на економічному розвитку та управлінні інфраструктурою, демонструючи таким чином значну різноманітність піджанрів у межах стратегічних ігор.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ГРИ В ЖАНРІ СТРАТЕГІЯ

3.1 Опис програмної реалізації

Вся програмна реалізація для створення комп'ютерної гри в жанрі стратегія була зроблена завдяки таким програмам, як Unity та Visual Studio, які були використанні в процесі створення гри. Розглянемо що з себе представляють собою дані програми. Про Unity в якості ігрового рушія більш детально було описано в пункті 1.1 даної кваліфікаційної роботи.

Якщо розглядати Unity як програму, а не лише як ігровий рушій, то дана програма використовується для створення різноманітних інтерактивних 3D-додатків та симуляцій, що виходять за межі традиційних відеоігор. Unity надає потужний набір інструментів та можливостей для розробки візуалізацій, тренажерів, архітектурних проєктів, освітніх програм та багато іншого.

Розглянемо Unity в якості програми:

➢ *Візуалізація даних та інформації:*

• Інтерактивні 3D-моделі: Unity дозволяє імпортувати та маніпулювати складними 3D-моделями з різних джерел. Користувачі можуть взаємодіяти з цими моделями, розглядати їх під різними кутами, розбирати на компоненти, отримувати детальну інформацію про окремі елементи.

• Динамічне відображення даних: Unity може використовуватися для візуалізації великих обсягів даних у тривимірному просторі. Це може бути корисно для аналізу наукових даних, фінансових показників, геопросторової інформації тощо. Дані можуть бути прив'язані до візуальних елементів, змінюючи їх колір, розмір, положення в залежності від значень.

• Створення інтерактивних дашбордів: Замість статичних графіків та діаграм, Unity дозволяє створювати динамічні 3D-дашборди, де користувачі можуть досліджувати дані в інтерактивному режимі, фільтрувати їх, порівнювати різні параметри.

➢ *Симуляції та тренажери:*

• Промислові симуляції: Unity використовується для створення реалістичних симуляцій роботи складного обладнання, виробничих процесів, логістичних систем. Це дозволяє навчати персонал, оптимізувати процеси, виявляти потенційні проблеми без ризику для реального обладнання.

• Медичні тренажери: У сфері медицини Unity застосовується для розробки інтерактивних тренажерів для хірургів, медсестер та інших медичних працівників. Ці тренажери можуть імітувати складні процедури, надзвичайні ситуації, дозволяючи відпрацьовувати навички в безпечному середовищі.

• Наукові симуляції: Unity може бути використана для моделювання фізичних явищ, хімічних реакцій, біологічних процесів. Інтерактивність дозволяє дослідникам змінювати параметри, спостерігати за результатами в реальному часі, глибше розуміти складні системи.

➢ *Архітектурні та інженерні візуалізації:*

• Інтерактивні прогулянки: Unity дозволяє створювати віртуальні тури по ще не збудованих будівлях або об'єктах інфраструктури. Клієнти та зацікавлені сторони можуть вільно переміщатися по віртуальному простору, оцінювати дизайн, планування, матеріали.

• BIM-візуалізація: Unity може інтегруватися з BIM-даними (Building Information Modeling) для створення інтерактивних 3D-моделей будівель з усією пов'язаною інформацією. Це полегшує комунікацію між різними учасниками проєкту, виявлення колізій, планування будівельних робіт.

• Аналіз освітлення та інсоляції: Unity дозволяє моделювати освітлення в приміщеннях та на відкритих просторах в залежності від часу доби та пори року. Це допомагає архітекторам та інженерам оптимізувати енергоефективність будівель.

➢ *Освітні програми та інтерактивні інсталяції:*

• Віртуальні екскурсії: Unity може використовуватися для створення віртуальних екскурсій по музеях, історичних місцях, природних заповідниках.

Користувачі можуть досліджувати ці місця з будь-якої точки світу, отримуючи додаткову інформацію у вигляді тексту, аудіо, відео.

- Інтерактивні навчальні матеріали: Unity дозволяє розробляти захоплюючі навчальні програми з інтерактивними 3D-моделями, симуляціями, вікторинами. Це підвищує залученість учнів та покращує засвоєння матеріалу.

- Інтерактивні інсталяції для виставок та заходів: Unity може використовуватися для створення унікальних інтерактивних інсталяцій, що реагують на дії відвідувачів, відображають дані в реальному часі, створюють захоплюючий досвід.

➢ *Основні переваги використання Unity як програми для неігрових цілей:*

- Потужна 3D-графіка та візуалізація: Unity забезпечує високу якість рендерингу, підтримку різноманітних візуальних ефектів, що дозволяє створювати реалістичні та привабливі додатки.

- Інтерактивність: Unity надає широкі можливості для створення інтерактивних сценаріїв, реакції на дії користувача, обробки вхідних даних.

- Кросплатформність: Додатки, створені на Unity, можуть бути легко розгорнуті на різних платформах: Windows, macOS, Linux, WebGL, VR/AR-пристрої та інші.

- Велика екосистема асетів та плагінів: Unity Asset Store пропонує величезну кількість готових 3D-моделей, текстур, скриптів, інструментів, що значно прискорює процес розробки.

- Активна спільнота та велика кількість навчальних матеріалів: Існує велика спільнота розробників Unity, яка готова допомогти з вирішенням проблем. Також доступно безліч навчальних матеріалів, документації, курсів.

- Гнучкість та розширюваність: Unity підтримує скриптування на C#, що дозволяє розробникам створювати власну логіку, розширювати функціональність двигуна за допомогою плагінів та інтеграцій з іншими інструментами.

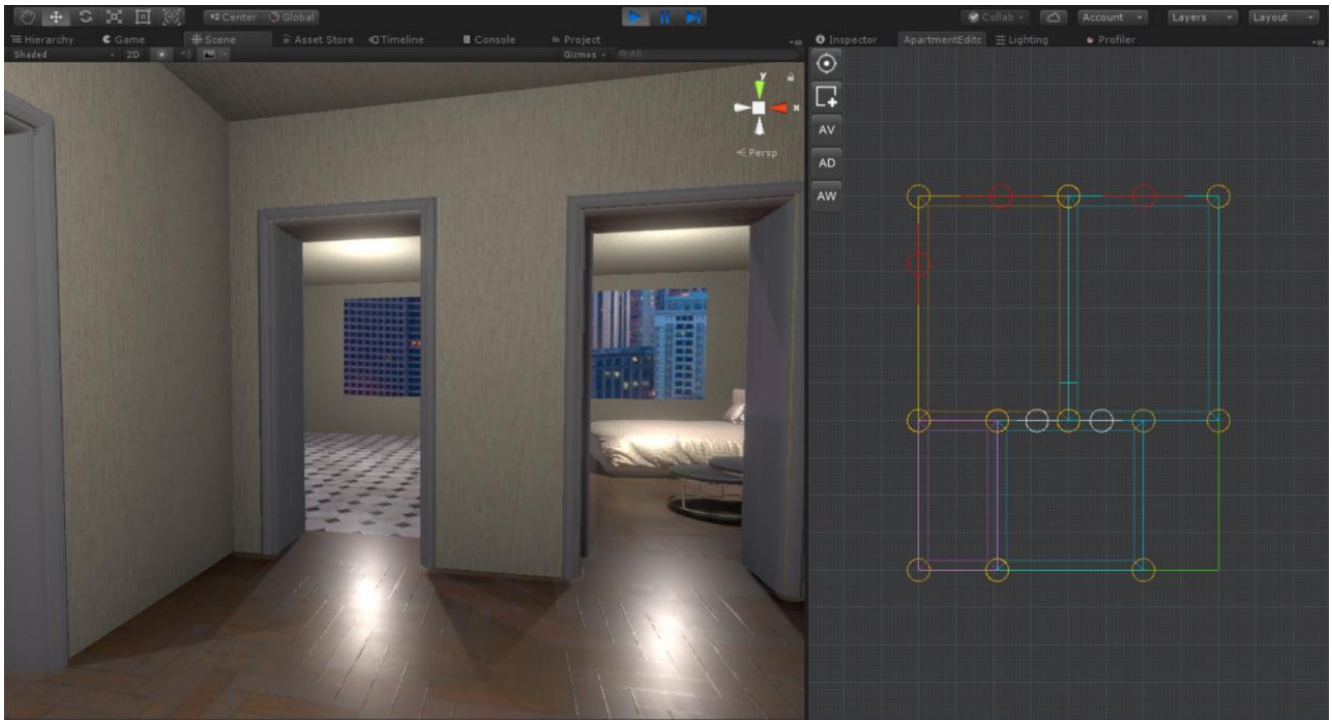


Рисунок 3.1 – Приклад проекту в редакторі Unity

Джерело: сформовано автором на основі [1,2,3]

Visual Studio — це інтегроване середовище розробки (IDE), створене компанією Microsoft. Воно призначене для створення різноманітних комп'ютерних програм, зокрема вебсайтів, вебдодатків, вебсервісів і мобільних застосунків. Visual Studio працює на основі платформ розробки Microsoft, таких як Windows API, Windows Forms, Windows Presentation Foundation (WPF), Microsoft Store і Microsoft Silverlight. Середовище підтримує генерацію як нативного, так і керованого коду.

До складу Visual Studio входить текстовий редактор з підтримкою IntelliSense (функція автодоповнення коду) та рефакторингу. Вбудований зневадник може працювати як на рівні вихідного коду, так і на машинному рівні. Також середовище містить інші інструменти, як-от: профілювальник коду, конструктор графічного інтерфейсу, редактор для вебдизайну, дизайнер класів і редактор схем баз даних. Платформа підтримує підключення плагінів, які дозволяють розширювати функціональність — наприклад, додавати підтримку систем контролю версій (як-от Subversion або Git) чи нові

інструменти для спеціалізованих мов програмування або інших етапів життєвого циклу розробки (наприклад, клієнт Azure DevOps — Team Explorer).

Visual Studio підтримує 36 мов програмування і дозволяє редактору коду та зневаднику працювати з більшістю з них — за умови наявності відповідного мовного сервісу. Вбудована підтримка охоплює такі мови, як: C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML і CSS. Через плагіни можна додати підтримку Python, Ruby, Node.js, M та інших мов. У минулому також підтримувались Java та J#.

Найпростіша версія середовища — Visual Studio Community — доступна безкоштовно. Її гасло: «Безкоштовне повнофункціональне середовище розробки для студентів, індивідуальних розробників і проєктів з відкритим кодом». Станом на 23 березня 2025 року актуальною стабільною версією є Visual Studio 2022, тоді як версії 2015, 2017 та 2019 перебувають на розширеній підтримці.

Visual Studio не має вбудованої підтримки жодної мови програмування, рішень чи інструментів за замовчуванням — замість цього функціональність додається через модулі, відомі як VSPackage. Після встановлення такий модуль функціонує як сервіс. Середовище розробки надає три основні сервіси:

- SVsSolution — забезпечує доступ до проєктів і рішень;
- SVsUIShell — відповідає за інтерфейс і віконну систему (вкладки, панелі інструментів, вікна);
- SVsShell — займається реєстрацією VSPackages.

IDE також координує та забезпечує взаємодію між цими сервісами. Усі редактори, дизайнери, типи проєктів та інші інструменти реалізовані як VSPackages. Visual Studio використовує COM для взаємодії з ними. До складу SDK входить Managed Package Framework (MPF) — набір оболонок для COM-інтерфейсів, що дозволяє створювати пакети на будь-якій мові, сумісній із CLI. Проте MPF охоплює не всі можливості, які надають COM-інтерфейси Visual

Studio. Ці сервіси можна використовувати для створення нових пакетів, які розширюють функціональність IDE.

Підтримка мов програмування реалізується через спеціальні VSPackage, що називаються Language Service. Такий сервіс реалізує інтерфейси, які надають підтримку функцій: підсвічування синтаксису, автодоповнення коду, парні дужки, підказки параметрів, списки членів класів, маркери помилок під час фонової компіляції тощо. Якщо інтерфейс реалізовано, функція буде доступна для відповідної мови. Language Service створюється окремо для кожної мови й може використовувати код з парсера або компілятора. Його можна реалізувати як у нативному коді, використовуючи COM або Babel Framework (частина SDK), так і в керованому коді з використанням MPF.

Visual Studio не має вбудованої підтримки систем контролю версій, але підтримує два способи інтеграції з ними. Один варіант — створення VSPackage з власним інтерфейсом. Інший — використання плагіна, що працює через MSSCCI (Microsoft Source Code Control Interface), який надає набір функцій для реалізації контролю версій через стандартний інтерфейс Visual Studio. MSSCCI спочатку використовувався для інтеграції з Visual SourceSafe у Visual Studio 6.0, а пізніше був відкритий через SDK. Visual Studio .NET 2002 використовував MSSCCI 1.1, а .NET 2003 — MSSCCI 1.2. У версіях 2005, 2008 та 2010 використовується MSSCCI 1.3, що додає підтримку перейменування, видалення та асинхронного відкриття файлів.

Visual Studio дозволяє запускати декілька екземплярів середовища розробки одночасно, кожен з власним набором VSPackage. Вони використовують різні гілки реєстру (registry hives) для збереження конфігурації та мають унікальний AppId. Кожен екземпляр запускається через спеціальний .exe файл, який задає відповідний AppId і ініціалізує середовище. Пакети, зареєстровані для одного AppId, працюють лише в межах цього екземпляру.

Різні редакції Visual Studio створюються на основі різних AppId. Наприклад, продукти Visual Studio Express мають свої AppId, тоді як Standard, Professional і Team Suite використовують один спільний. Тому Express-версії

можна встановлювати паралельно з іншими, а інші редакції оновлюють вже наявну інсталяцію. Професійна версія містить розширений набір VSPackages порівняно зі стандартною, а Team Suite включає всі можливості обох попередніх. Система AppId також активно використовується в Visual Studio Shell, починаючи з версії 2008.

Visual Studio містить редактор коду, який підтримує підсвічування синтаксису та автозаповнення з використанням IntelliSense. Ця функція забезпечує підказки для змінних, функцій, методів, циклів та запитів LINQ. IntelliSense доступна для мов, що входять до складу Visual Studio, а також для XML, CSS та JavaScript під час створення вебсайтів і вебдодатків. Пропозиції автозаповнення відображаються у вікні, яке з'являється поруч із курсором. Починаючи з версії Visual Studio 2008, це вікно можна зробити напівпрозорим, щоб бачити код під ним. Редактор коду підтримує всі мови, які можна використовувати у середовищі.

У редакторі також можна встановлювати закладки для швидкої навігації, згортати блоки коду, використовувати поступовий пошук, звичайний текстовий пошук та пошук за регулярними виразами. Він підтримує буфер обміну з кількома елементами, список завдань та шаблони коду (code snippets), які можна вставляти та налаштовувати під потреби проєкту. Існує інструмент для управління цими шаблонами. Усі ці елементи можуть бути закріплені або приховані автоматично, якщо не використовуються. Також редактор підтримує рефакторинг коду: перейменування змінних та методів, зміну порядку параметрів, виділення інтерфейсів і перетворення полів на властивості.

У склад Visual Studio входить зневадник (debugger), який працює як на рівні вихідного коду, так і на машинному рівні. Він підтримує як керований (managed), так і некерований (native) код, а також дозволяє підключатися до вже запущених процесів для моніторингу та налагодження. Якщо доступний вихідний код, зневадник показує його під час виконання, якщо ні — показує дизасембльований код. Також доступне створення та завантаження дампів

пам’яті, а ще підтримка багатопоточності. За бажанням зневадник може автоматично запускатися під час аварійного завершення роботи застосунку.

Visual Studio Debugger дозволяє встановлювати точки зупину (breakpoints), які можуть бути умовними, та використовувати «спостереження» (watch) для контролю змін значень змінних під час виконання. Можна виконувати покрокову налагоджувальну перевірку коду: заходити всередину функцій або обходити їх. Підтримується функція Edit and Continue — зміна коду під час виконання. Також можна переглядати значення змінних через підказки, наведенням курсора миші, і змінювати їх прямо у процесі налагодження. Додатково доступна консоль Immediate, через яку можна викликати методи та передавати їм параметри вручну. Visual Studio також має ряд візуальних дизайнерів для спрощення розробки застосунків:

- Дизайнер Windows Forms — дозволяє створювати графічні інтерфейси Windows Forms, використовуючи контейнери для компонування елементів і прив'язку їх до країв форми. Компоненти, що відображають дані, можна прив’язати до джерел — баз даних або запитів. Події керують взаємодією між UI та кодом.
- WPF-дизайнер (Cider) — вперше представлений у Visual Studio 2008, дозволяє створювати інтерфейси для Windows Presentation Foundation. Підтримує прив’язку даних, автоматичне компонування та генерує XAML-код, який сумісний із Microsoft Expression Design.
- Веб-дизайнер — забезпечує створення сторінок методом перетягування елементів для ASP.NET-додатків із підтримкою HTML, CSS та JavaScript. Код прив’язується за допомогою моделі code-behind. Починаючи з версії 2008, дизайнер використовує рушій від Expression Web.
- Дизайнер класів — дозволяє створювати та редагувати класи, їхні члени та рівні доступу у вигляді UML-діаграм. Може генерувати код C# та VB.NET.
- Дизайнер даних — дозволяє візуально створювати структури бази даних, визначати таблиці, ключі та обмеження, а також будувати запити.

• Mapping Designer — використовується з LINQ to SQL для створення відповідності між класами й базами даних. У новіших версіях Visual Studio його замінює ADO.NET Entity Framework.

Інші корисні інструменти:

• Редактор властивостей (Properties Editor) — панель для редагування властивостей об’єктів (форм, сторінок тощо).

• Object Browser — перегляд ієрархій класів і просторів імен у .NET.

• Solution Explorer — навігація і керування файлами в межах проєкту.

• Team Explorer — інтеграція з Azure DevOps, включно з управлінням завданнями, помилками, історіями користувачів.

• Data Explorer — інструмент для управління базами даних SQL Server, створення таблиць, запитів, процедур.

• Server Explorer — для керування підключеннями до баз даних і сервісів Windows.

• Dotfuscator Community Edition — безкоштовна версія засобу для обфускації коду.

• T4 (Text Template Transformation Toolkit) — фреймворк для генерації текстових файлів із шаблонів.

• ASP.NET Web Site Administration Tool — засіб для налаштування ASP.NET-сайтів.

• Visual Studio Tools for Office (VSTO) — набір інструментів для розробки рішень для Microsoft Office.

• Інструменти для тестування — включають модульні тести, IntelliTest, Live Unit Testing, CodeLens, аналіз покриття коду, Fakes та інші функції для забезпечення якості коду.

Visual Studio дозволяє розробникам створювати розширення, які розширюють функціональні можливості середовища. Такі розширення інтегруються в середовище розробки й можуть реалізовувати нові функції.

Існує кілька типів розширень: макроси, надбудови (add-ins) та пакети (packages).

Макроси — це дії або завдання, які можна автоматизувати: записати, зберегти, повторити й поширити. Вони створюються за допомогою Visual Basic, але не компілюються. Макроси не можуть додавати нові команди чи створювати вікна інструментів.

Надбудови (Add-Ins) — надають доступ до об'єктної моделі Visual Studio й можуть взаємодіяти з його інструментами. Завдяки цьому можна реалізовувати новий функціонал і додавати власні вікна інструментів. Надбудови підключаються до середовища за допомогою технології COM і можуть створюватися на будь-якій мові, що її підтримує.

Пакети (Packages) — надають найбільші можливості розширення і створюються за допомогою Visual Studio SDK. Вони дозволяють реалізовувати нові редактори, інструменти, інтегрувати додаткові мови програмування. SDK забезпечує доступ як до некерованих API, так і до керованих, хоча останні мають обмежену функціональність.

Починаючи з версії Visual Studio 2005, підтримка розширень доступна у версіях Standard і вище. Версії Express не підтримують встановлення розширень. У Visual Studio 2008 з'явився Visual Studio Shell, який дозволяє створювати індивідуальні версії середовища розробки. Shell визначає набір компонентів (VSPackages), які забезпечують базову функціональність IDE. Поверх цих компонентів можна додати інші пакети для створення кастомізованої версії Visual Studio.

Ізольований режим (Isolated Mode) створює новий AppId (ідентифікатор застосунку), в межах якого встановлюються пакети, і запускається через окремий виконуваний файл. Цей режим призначений для створення власного середовища розробки, наприклад, для певної мови або специфічної задачі.

Інтегрований режим (Integrated Mode) встановлює пакети безпосередньо у вже наявне середовище Visual Studio (Professional, Standard або Team System),

дозволяючи інтегрувати інструменти у ці редакції. Visual Studio Shell доступний для завантаження безкоштовно.

Після виходу Visual Studio 2008 компанія Microsoft створила Visual Studio Gallery — централізований онлайн-сервіс для публікації розширень. Як індивідуальні, так і комерційні розробники можуть розміщувати інформацію про свої розширення для Visual Studio починаючи з версії .NET 2002 і до Visual Studio 2010. Користувачі сайту можуть оцінювати й залишати відгуки на розширення, допомагаючи іншим оцінити їх якість. Розширення публікуються у форматі VSIX. Такий файл насправді є архівом ZIP, що містить XML-файли й за потреби DLL-бібліотеки. Однією з основних переваг є те, що ці розширення не потребують прав адміністратора для встановлення. Також планується підтримка RSS-підписок для сповіщень про оновлення і функція тегів для зручної навігації.

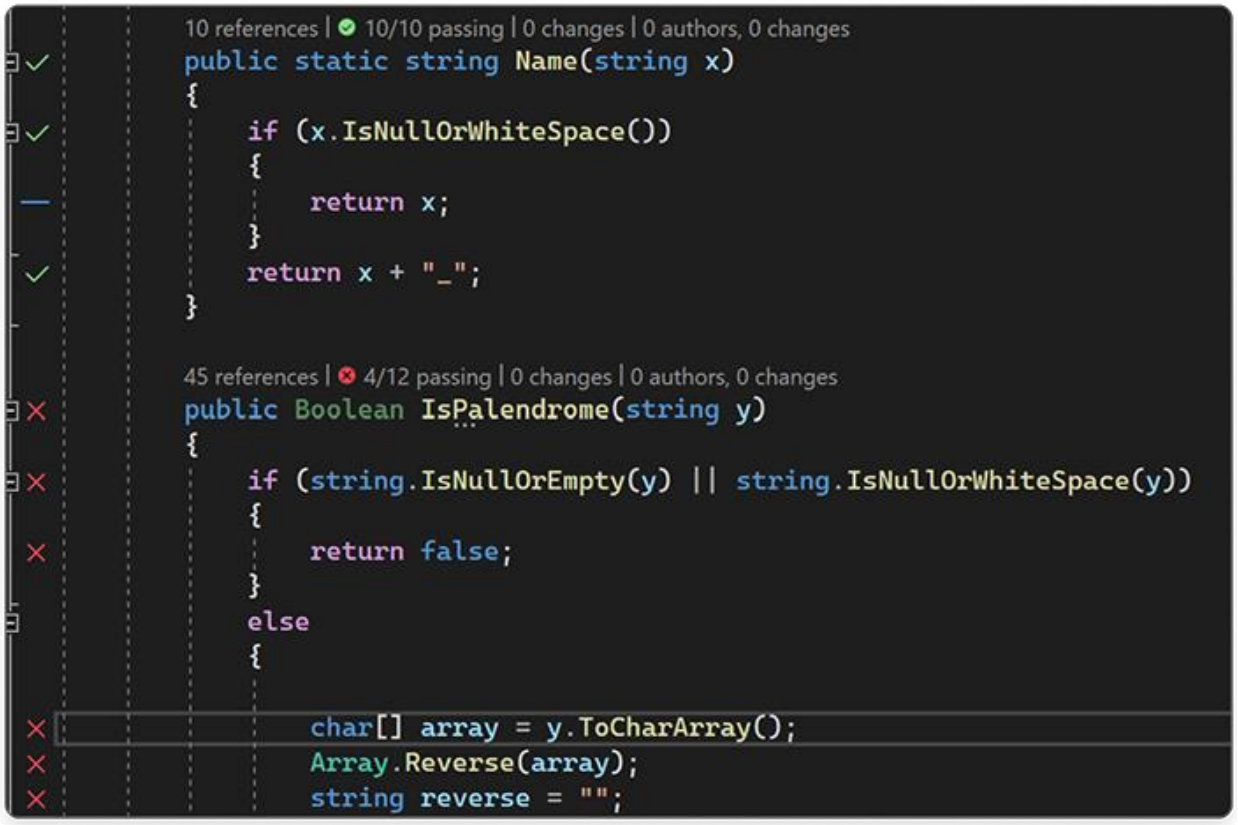


Рисунок 3.2 – Приклад коду в Visual Studio

Джерело: сформовано автором на основі [33,34,35]

3.2 Практичне використання програм під час створення гри

Як було зазначено в попередньому пункті, в процесі створення комп’ютерної гри в жанрі стратегія використовувались програми Unity та Visual Studio.

Ігровий рушій Unity використовувався для побудування карти гри, її ландшафту, природних перешкод з розміткою зон завдяки моделям дерев, будування природних стін по периметру карти завдяки моделям каміння, які, як і моделі дерев, були взяті з паку асетів SimpleNaturePack, який був завантажений з Unity Asset Store і імпортований в асети гри.

Також, як і з побудуванням карти гри і інших ландшафтних елеменітів і моделей, ігровий рушій Unity використовувався для додавання юнітів гравця та ворожих юнітів, а саме їх генерацію на самій карті, а також ангарів, звідки з'являлися юніти гравця та ворожі юніти, завдяки моделі танку типу БТР з паку асетів RTS_Modern_Combat_Vehicle_Pack_Free та моделі будунку з паку асетів Village Pack, які також були завантажені з Unity Asset Store і як в випадку з паком асетів SimpleNaturePack, були також імпортовані в асети гри.

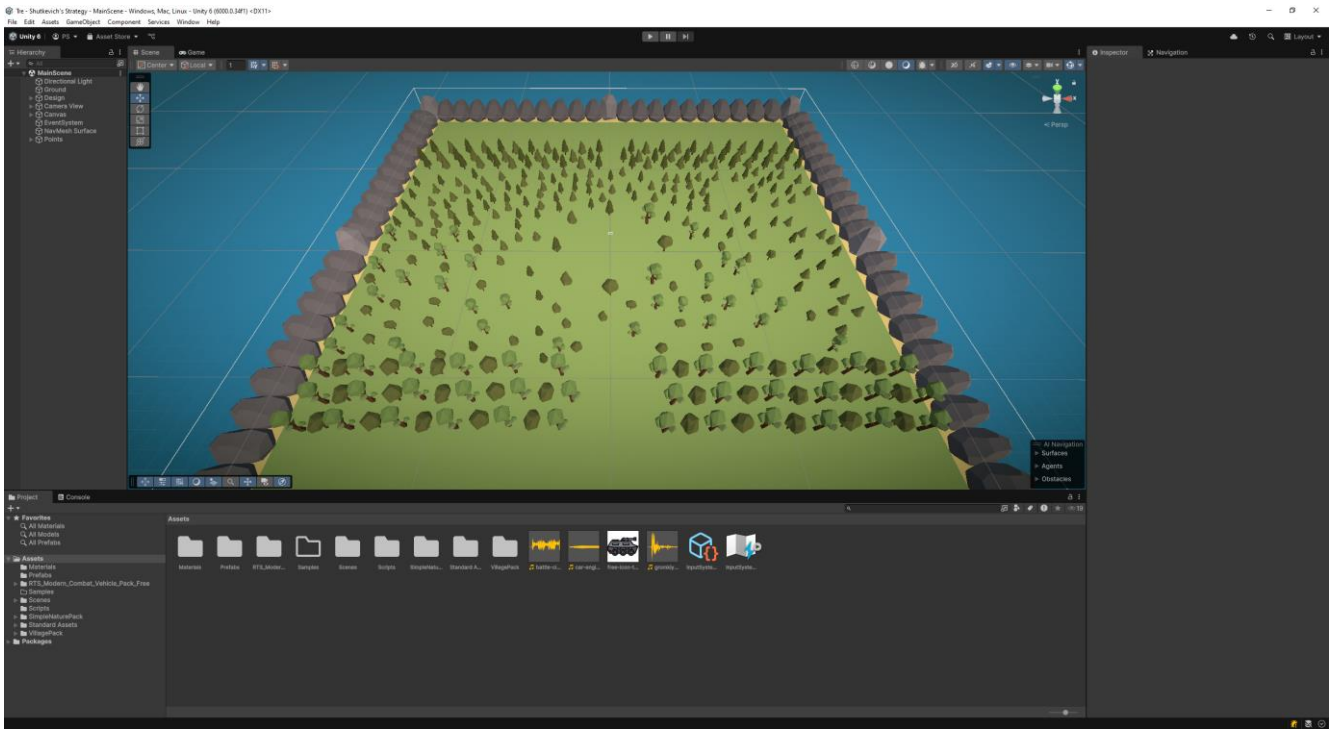
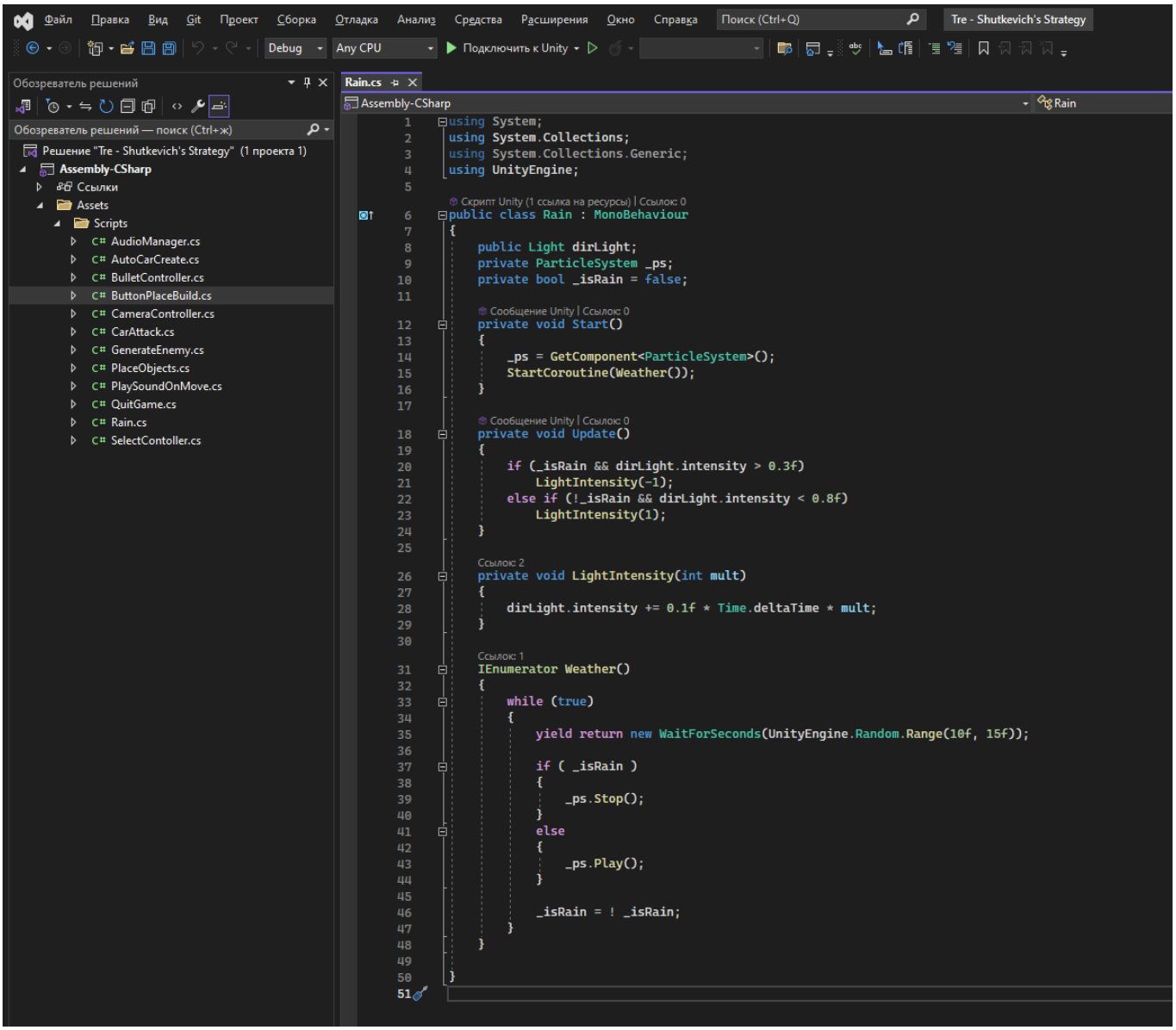
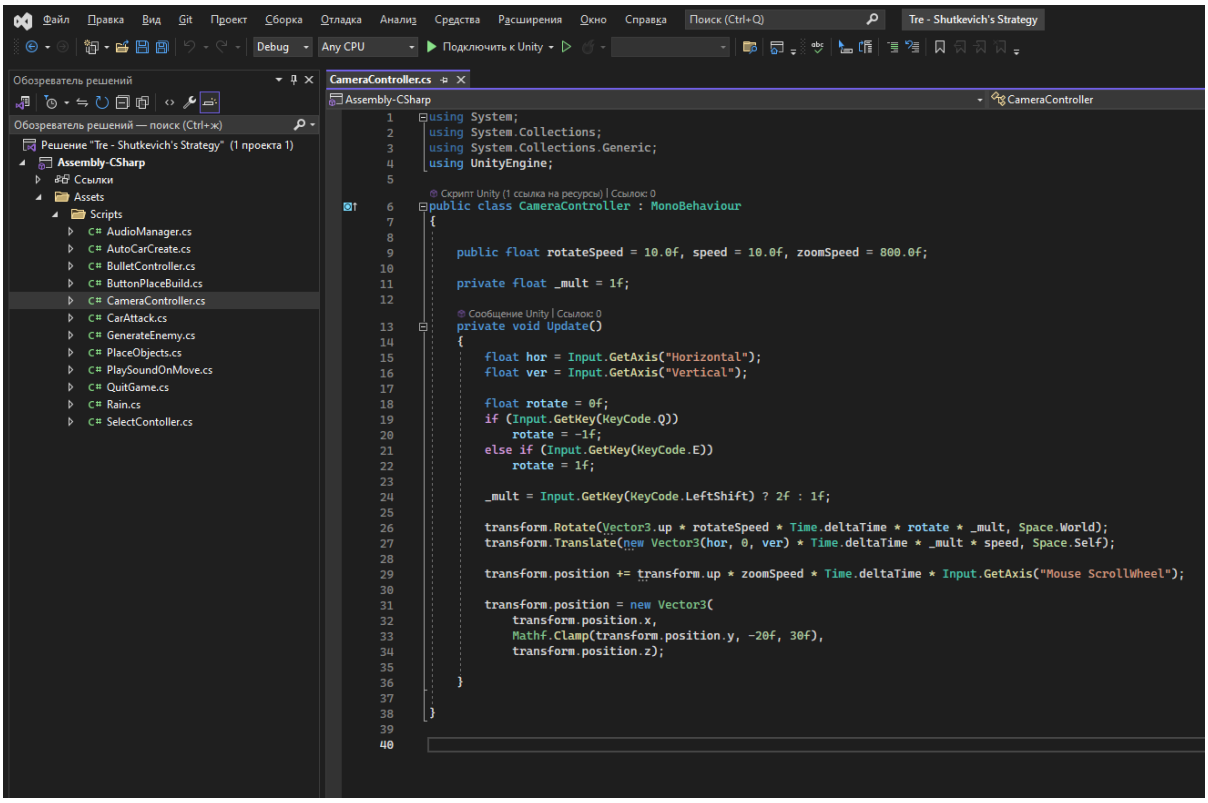


Рисунок 3.3 – Карта гри та список всіх асетів гри в Unity

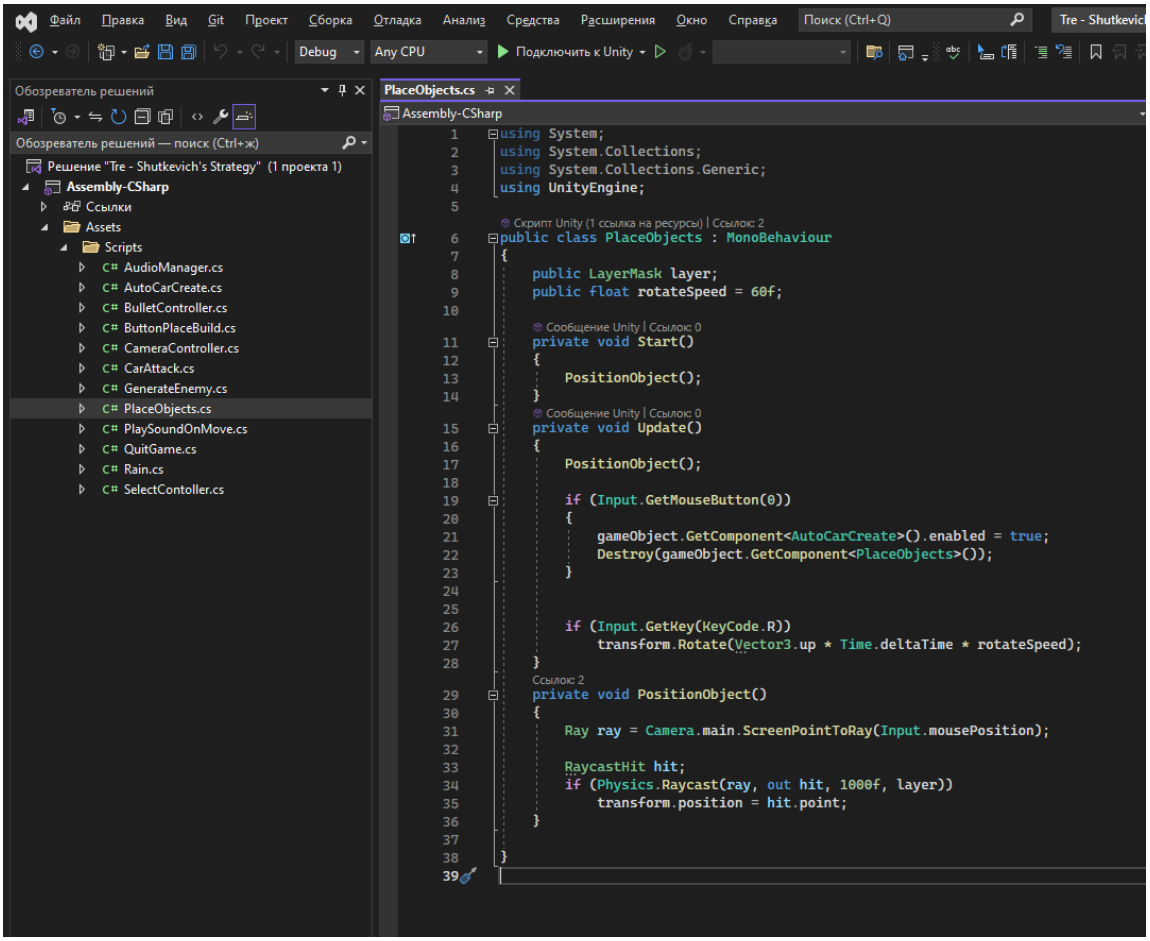
Джерело: сформовано автором

Інтегроване середовище розробки Visual Studio використовувалося для написання всіх скриптів гри, які використовувались для зміни погоди, генерації юнітів гравця та ворожих юнітів, вибору юнітів гравця, будуванню та розташуванню ангарів гравця та ворожих ангарів на карті, відтворення звуку переміщення юніта або юнітів гравця та ворожих юнітів на інші координати на карті, контролю камери гри, взаємної атаки юнітів гравця та опонента, коли вони бачать один одного, знаходячись в взаємному полі зору, контролю снарядів, звукового супроводження та виходу з гри.

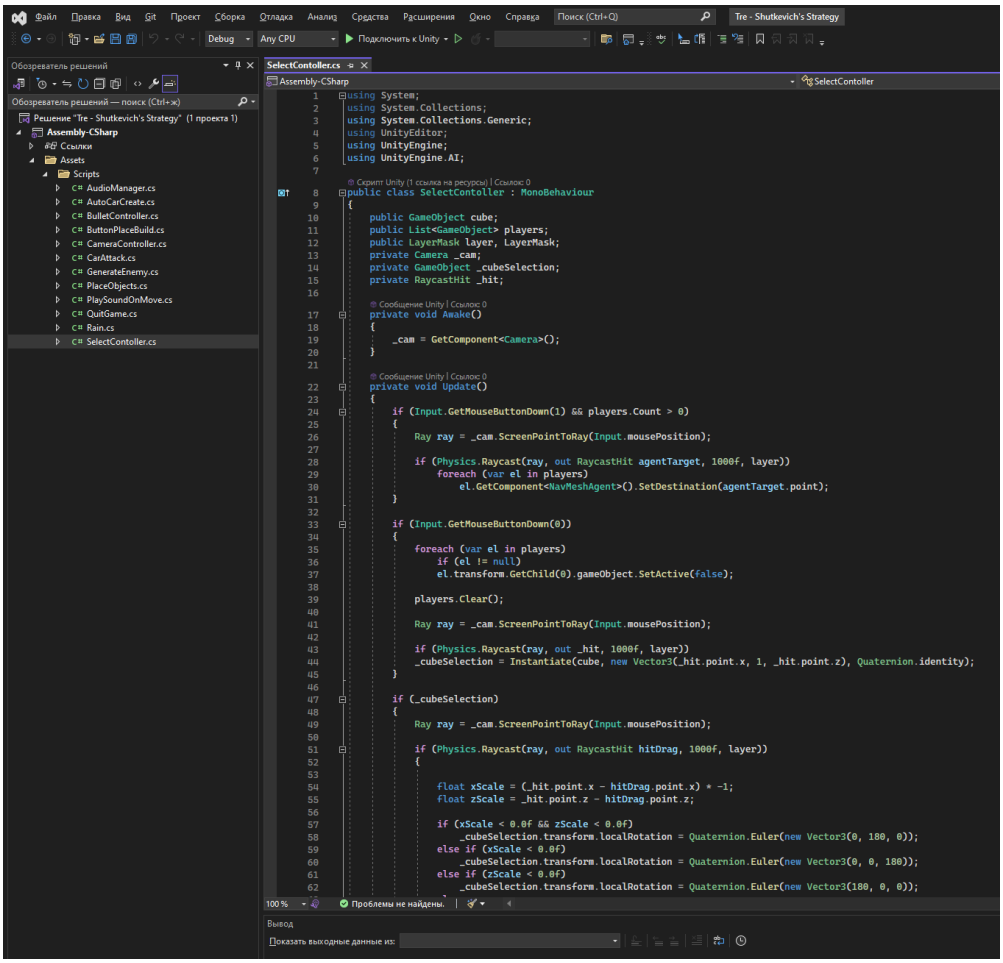

Файл Правка Вид Git Проект Сборка Отладка Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
Tre - Shutkevich's Strategy
Debug
Any CPU
Подключить к Unity
Обозреватель решений
Обозреватель решений — поиск (Ctrl+ж)
Решение "Tre - Shutkevich's Strategy" (1 проекта 1)
Assembly-CSharp
Ссылки
Assets
Scripts
AudioManager.cs
AutoCarCreate.cs
BulletController.cs
ButtonPlaceBuild.cs
CameraController.cs
CarAttack.cs
GenerateEnemy.cs
PlaceObjects.cs
PlaySoundOnMove.cs
QuitGame.cs
Rain.cs
SelectContoller.cs
Rain.cs
Assembly-CSharp
Rain
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class Rain : MonoBehaviour
{
public Light dirLight;
private ParticleSystem _ps;
private bool _isRain = false;
Сообщение Unity | Ссылок: 0
private void Start()
{
_ps = GetComponent<ParticleSystem>();
StartCoroutine(Weather());
}
Сообщение Unity | Ссылок: 0
private void Update()
{
if (_isRain && dirLight.intensity > 0.3f)
LightIntensity(-1);
else if (!_isRain && dirLight.intensity < 0.8f)
LightIntensity(1);
}
Ссылок: 2
private void LightIntensity(int mult)
{
dirLight.intensity += 0.1f * Time.deltaTime * mult;
}
Ссылок: 1
IEnumerator Weather()
{
while (true)
{
yield return new WaitForSeconds(UnityEngine.Random.Range(10f, 15f));
if (_isRain)
{
_ps.Stop();
}
else
{
_ps.Play();
}
_isRain = ! _isRain;
}
}
}



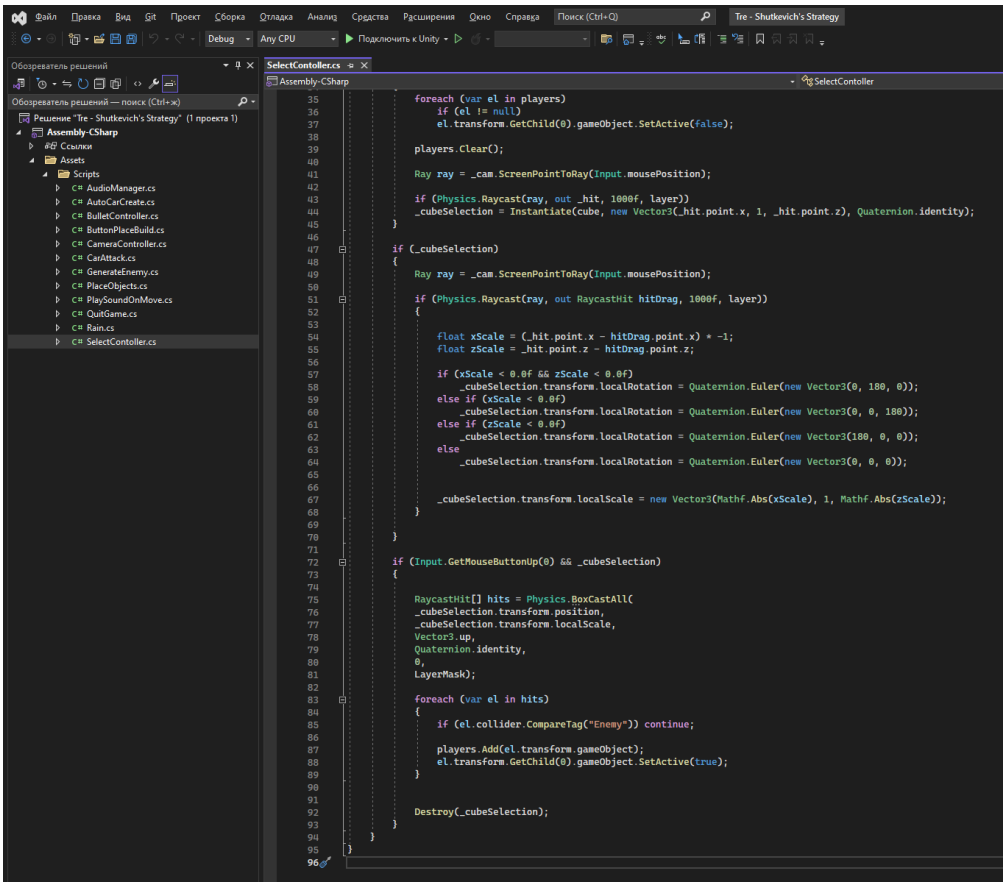
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class CameraController : MonoBehaviour
{

    public float rotateSpeed = 10.0f, speed = 10.0f, zoomSpeed = 800.0f;

    private float _mult = 1f;

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        float hor = Input.GetAxis("Horizontal");
        float ver = Input.GetAxis("Vertical");

        float rotate = 0f;
        if (Input.GetKey(KeyCode.Q))
            rotate = -1f;
        else if (Input.GetKey(KeyCode.E))
            rotate = 1f;

        _mult = Input.GetKey(KeyCode.LeftShift) ? 2f : 1f;

        transform.Rotate(Vector3.up * rotateSpeed * Time.deltaTime * rotate * _mult, Space.World);
        transform.Translate(new Vector3(hor, 0, ver) * Time.deltaTime * _mult * speed, Space.Self);

        transform.position += transform.up * zoomSpeed * Time.deltaTime * Input.GetAxis("Mouse ScrollWheel");

        transform.position = new Vector3(
            transform.position.x,
            Mathf.Clamp(transform.position.y, -20f, 30f),
            transform.position.z);

    }
}
```



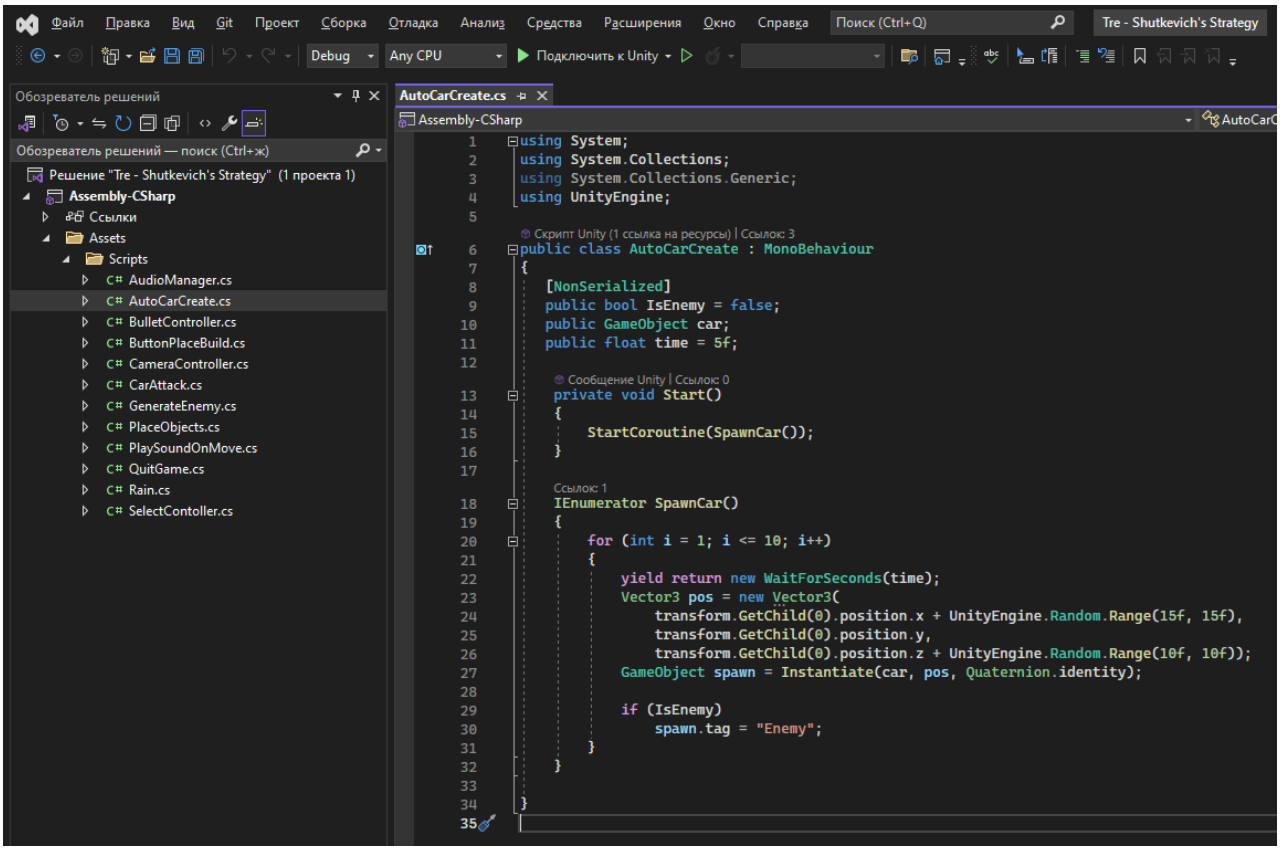
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 2
public class PlaceObjects : MonoBehaviour
{
    public LayerMask layer;
    public float rotateSpeed = 60f;

    Сообщение Unity | Ссылок: 0
    private void Start()
    {
        PositionObject();
    }
    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        PositionObject();

        if (Input.GetMouseButton(0))
        {
            gameObject.GetComponent<AutoCarCreate>().enabled = true;
            Destroy(gameObject.GetComponent<PlaceObjects>());
        }


        if (Input.GetKey(KeyCode.R))
            transform.Rotate(Vector3.up * Time.deltaTime * rotateSpeed);
    }
    Ссылок: 2
    private void PositionObject()
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

        RaycastHit hit;
        if (Physics.Raycast(ray, out hit, 1000f, layer))
            transform.position = hit.point;
    }

}
```

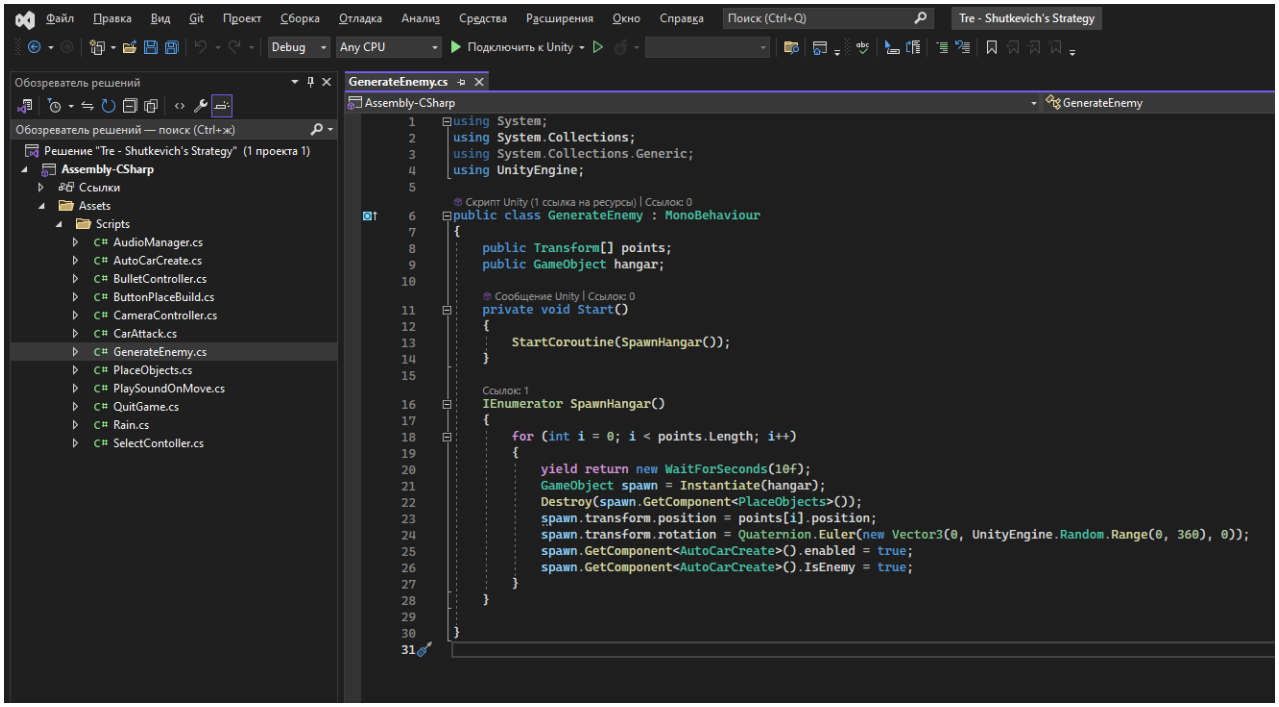


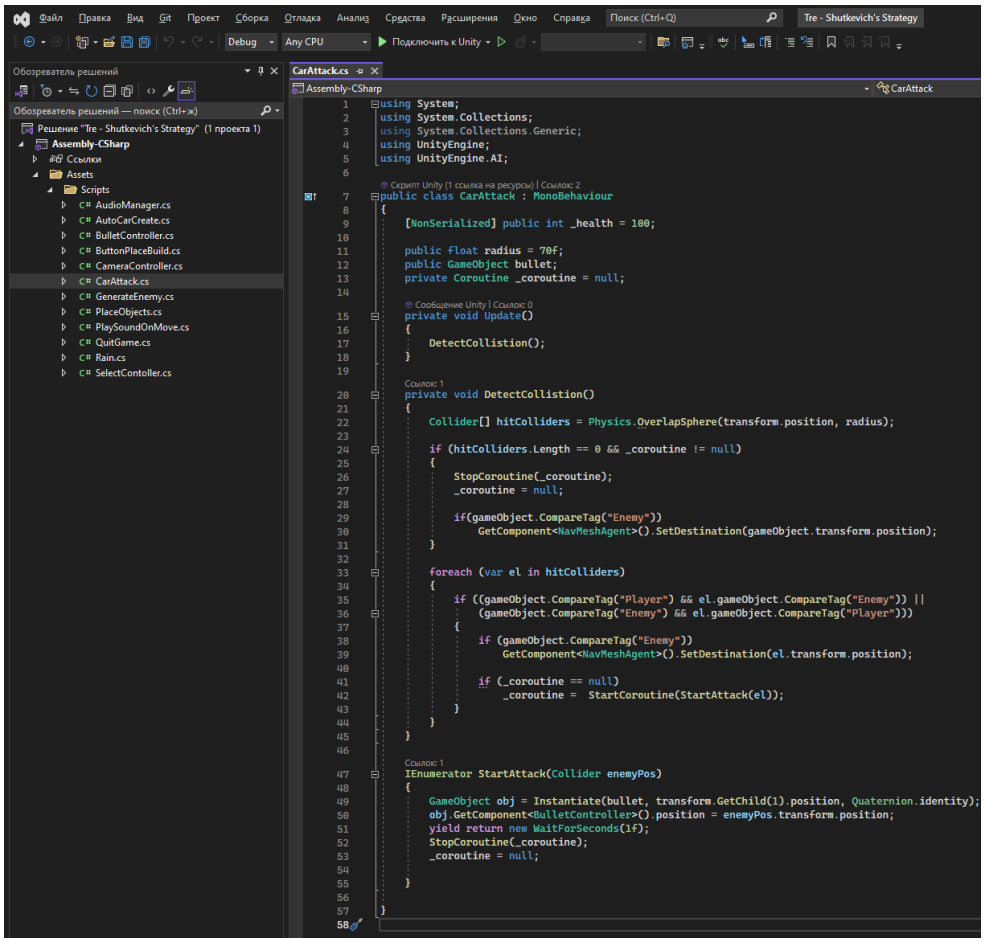
```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;
using UnityEngine.AI;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class SelectContoller : MonoBehaviour
{
    public GameObject cube;
    public List<GameObject> players;
    public LayerMask layer, LayerMask;
    private Camera _cam;
    private GameObject _cubeSelection;
    private RaycastHit _hit;

    Сообщение Unity | Ссылок: 0
    private void Awake()
    {
        _cam = GetComponent<Camera>();
    }

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        if (Input.GetMouseButtonDown(1) && players.Count > 0)
        {
            Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit agentTarget, 1000f, layer))
                foreach (var el in players)
                    el.GetComponent<NavMeshAgent>().SetDestination(agentTarget.point);
        }

        if (Input.GetMouseButtonDown(0))
        {
            foreach (var el in players)
                if (el != null)
                el.transform.GetChild(0).gameObject.SetActive(false);

            players.Clear();

            Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out _hit, 1000f, layer))
            _cubeSelection = Instantiate(cube, new Vector3(_hit.point.x, 1, _hit.point.z), Quaternion.identity);
        }

        if (_cubeSelection)
        {
            Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hitDrag, 1000f, layer))
            {

                float xScale = (_hit.point.x - hitDrag.point.x) * -1;
                float zScale = _hit.point.z - hitDrag.point.z;

                if (xScale < 0.0f && zScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(0, 180, 0));
                else if (xScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(0, 0, 180));
                else if (zScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(180, 0, 0));
```



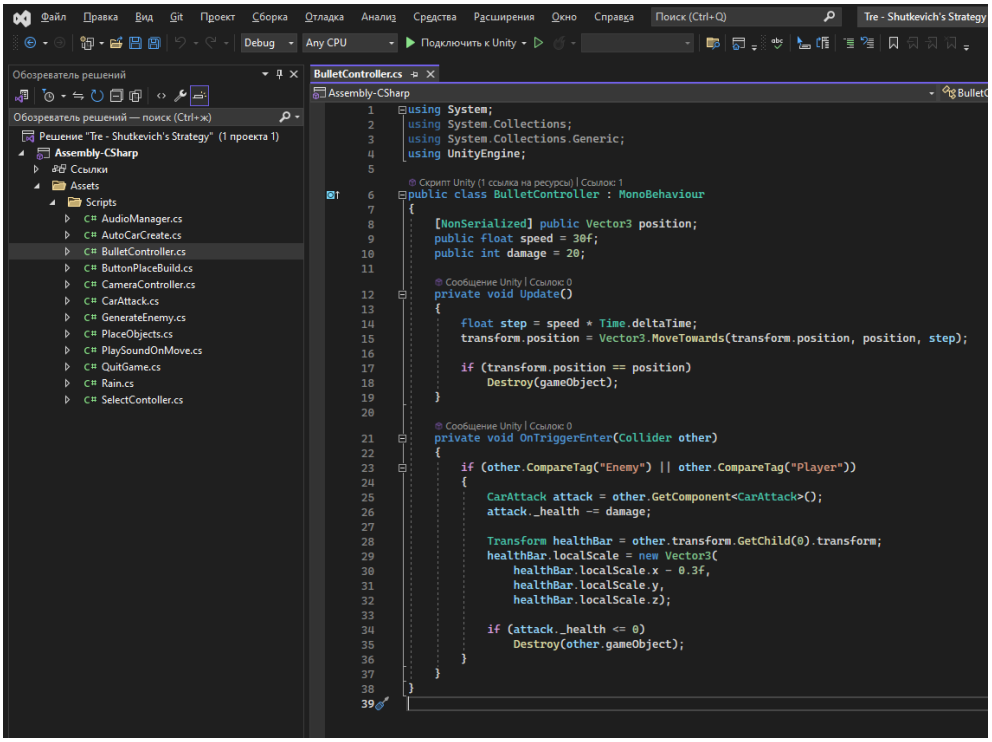
```
            foreach (var el in players)
                if (el != null)
                el.transform.GetChild(0).gameObject.SetActive(false);

            players.Clear();

            Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out _hit, 1000f, layer))
            _cubeSelection = Instantiate(cube, new Vector3(_hit.point.x, 1, _hit.point.z), Quaternion.identity);
        }

        if (_cubeSelection)
        {
            Ray ray = _cam.ScreenPointToRay(Input.mousePosition);

            if (Physics.Raycast(ray, out RaycastHit hitDrag, 1000f, layer))
            {

                float xScale = (_hit.point.x - hitDrag.point.x) * -1;
                float zScale = _hit.point.z - hitDrag.point.z;

                if (xScale < 0.0f && zScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(0, 180, 0));
                else if (xScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(0, 0, 180));
                else if (zScale < 0.0f)
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(180, 0, 0));
                else
                    _cubeSelection.transform.localRotation = Quaternion.Euler(new Vector3(0, 0, 0));


                _cubeSelection.transform.localScale = new Vector3(Mathf.Abs(xScale), 1, Mathf.Abs(zScale));
            }

        }

        if (Input.GetMouseButtonUp(0) && _cubeSelection)
        {

            RaycastHit[] hits = Physics.BoxCastAll(
            _cubeSelection.transform.position,
            _cubeSelection.transform.localScale,
            Vector3.up,
            Quaternion.identity,
            0,
            LayerMask);

            foreach (var el in hits)
            {
                if (el.collider.CompareTag("Enemy")) continue;

                players.Add(el.transform.gameObject);
                el.transform.GetChild(0).gameObject.SetActive(true);
            }


            Destroy(_cubeSelection);
        }
    }
}
```


Файл Правка Вид Git Проект Сборка Отладка Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
Tre - Shutkevich's Strategy
Debug
Any CPU
Подключить к Unity
Обозреватель решений
Обозреватель решений — поиск (Ctrl+ж)
Решение "Tre - Shutkevich's Strategy" (1 проекта 1)
Assembly-CSharp
Ссылки
Assets
Scripts
AudioManager.cs
AutoCarCreate.cs
BulletController.cs
ButtonPlaceBuild.cs
CameraController.cs
CarAttack.cs
GenerateEnemy.cs
PlaceObjects.cs
PlaySoundOnMove.cs
QuitGame.cs
Rain.cs
SelectContoller.cs
AutoCarCreate.cs
Assembly-CSharp
AutoCarC
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 3
public class AutoCarCreate : MonoBehaviour
{
[NonSerialized]
public bool IsEnemy = false;
public GameObject car;
public float time = 5f;
Сообщение Unity | Ссылок: 0
private void Start()
{
StartCoroutine(SpawnCar());
}
Ссылок: 1
IEnumerator SpawnCar()
{
for (int i = 1; i <= 10; i++)
{
yield return new WaitForSeconds(time);
Vector3 pos = new Vector3(
transform.GetChild(0).position.x + UnityEngine.Random.Range(15f, 15f),
transform.GetChild(0).position.y,
transform.GetChild(0).position.z + UnityEngine.Random.Range(10f, 10f));
GameObject spawn = Instantiate(car, pos, Quaternion.identity);
if (IsEnemy)
spawn.tag = "Enemy";
}
}
}

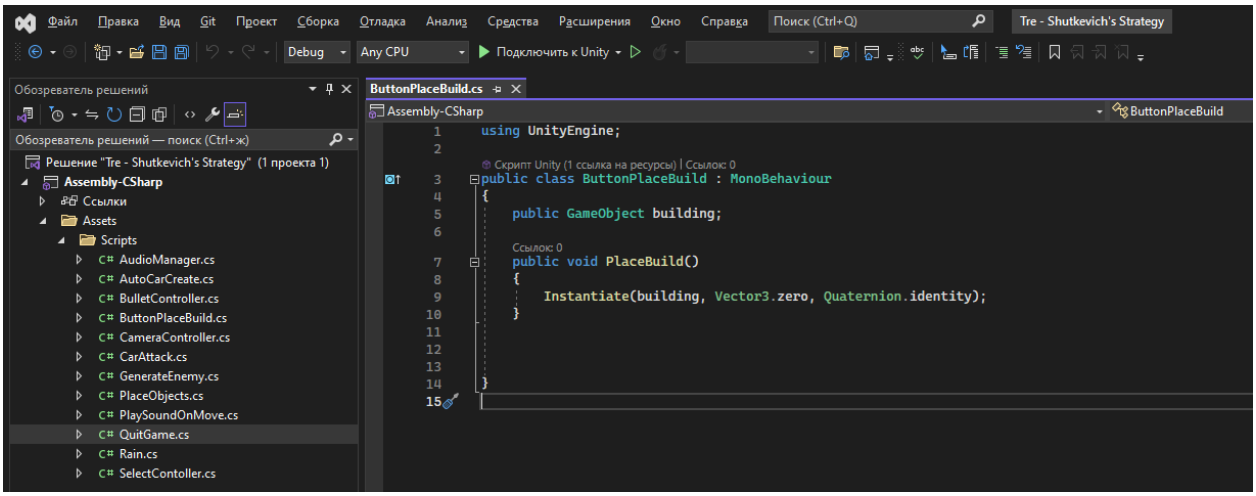

Файл Правка Вид Git Проект Сборка Отладка Анализ Средства Расширения Окно Справка
Поиск (Ctrl+Q)
Tre - Shutkevich's Strategy
Debug
Any CPU
Подключить к Unity
Обозреватель решений
Обозреватель решений — поиск (Ctrl+ж)
Решение "Tre - Shutkevich's Strategy" (1 проекта 1)
Assembly-CSharp
Ссылки
Assets
Scripts
AudioManager.cs
AutoCarCreate.cs
BulletController.cs
ButtonPlaceBuild.cs
CameraController.cs
CarAttack.cs
GenerateEnemy.cs
PlaceObjects.cs
PlaySoundOnMove.cs
QuitGame.cs
Rain.cs
SelectContoller.cs
GenerateEnemy.cs
Assembly-CSharp
GenerateEnemy
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 0
public class GenerateEnemy : MonoBehaviour
{
public Transform[] points;
public GameObject hangar;
Сообщение Unity | Ссылок: 0
private void Start()
{
StartCoroutine(SpawnHangar());
}
Ссылок: 1
IEnumerator SpawnHangar()
{
for (int i = 0; i < points.Length; i++)
{
yield return new WaitForSeconds(10f);
GameObject spawn = Instantiate(hangar);
Destroy(spawn.GetComponent<PlaceObjects>());
spawn.transform.position = points[i].position;
spawn.transform.rotation = Quaternion.Euler(new Vector3(0, UnityEngine.Random.Range(0, 360), 0));
spawn.GetComponent<AutoCarCreate>().enabled = true;
spawn.GetComponent<AutoCarCreate>().IsEnemy = true;
}
}
}

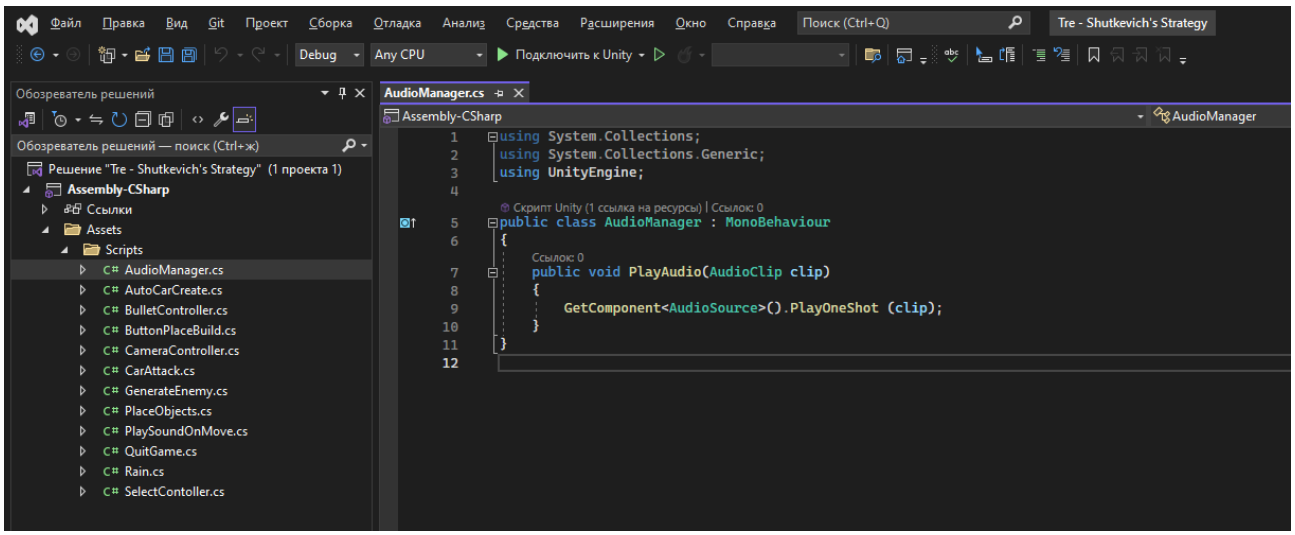


```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 2
public class CarAttack : MonoBehaviour
{
    [NonSerialized] public int _health = 100;

    public float radius = 70f;
    public GameObject bullet;
    private Coroutine _coroutine = null;

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        DetectCollistion();
    }

    Ссылок: 1
    private void DetectCollistion()
    {
        Collider[] hitColliders = Physics.OverlapSphere(transform.position, radius);

        if (hitColliders.Length == 0 && _coroutine != null)
        {
            StopCoroutine(_coroutine);
            _coroutine = null;

            if(gameObject.CompareTag("Enemy"))
                GetComponent<NavMeshAgent>().SetDestination(gameObject.transform.position);
        }

        foreach (var el in hitColliders)
        {
            if ((gameObject.CompareTag("Player") && el.gameObject.CompareTag("Enemy")) ||
                (gameObject.CompareTag("Enemy") && el.gameObject.CompareTag("Player")))
            {
                if (gameObject.CompareTag("Enemy"))
                    GetComponent<NavMeshAgent>().SetDestination(el.transform.position);

                if (_coroutine == null)
                    _coroutine =  StartCoroutine(StartAttack(el));
            }
        }
    }

    Ссылок: 1
    IEnumerator StartAttack(Collider enemyPos)
    {
        GameObject obj = Instantiate(bullet, transform.GetChild(1).position, Quaternion.identity);
        obj.GetComponent<BulletController>().position = enemyPos.transform.position;
        yield return new WaitForSeconds(1f);
        StopCoroutine(_coroutine);
        _coroutine = null;

    }
}
```



```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Скрипт Unity (1 ссылка на ресурсы) | Ссылок: 1
public class BulletController : MonoBehaviour
{
    [NonSerialized] public Vector3 position;
    public float speed = 30f;
    public int damage = 20;

    Сообщение Unity | Ссылок: 0
    private void Update()
    {
        float step = speed * Time.deltaTime;
        transform.position = Vector3.MoveTowards(transform.position, position, step);

        if (transform.position == position)
            Destroy(gameObject);
    }

    Сообщение Unity | Ссылок: 0
    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Enemy") || other.CompareTag("Player"))
        {
            CarAttack attack = other.GetComponent<CarAttack>();
            attack._health -= damage;

            Transform healthBar = other.transform.GetChild(0).transform;
            healthBar.localScale = new Vector3(
                healthBar.localScale.x - 0.3f,
                healthBar.localScale.y,
                healthBar.localScale.z);

            if (attack._health <= 0)
                Destroy(other.gameObject);
        }
    }
}
```





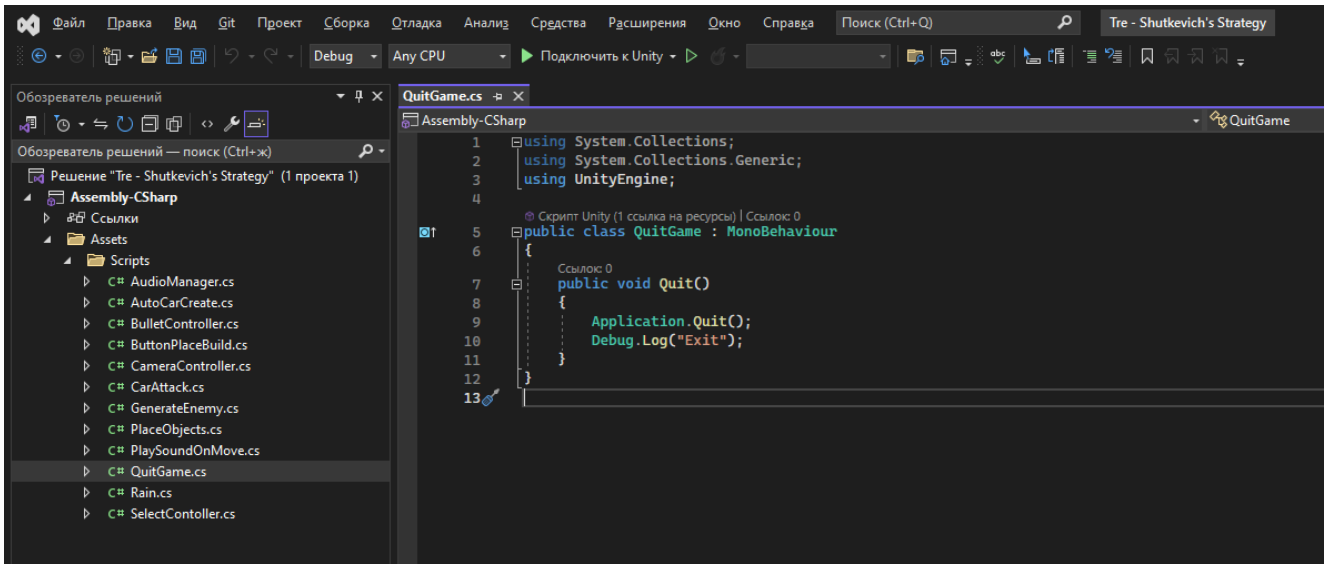


Рисунок 3.4 – 3.15 Скрипти гри в жанрі стратегія

Джерело: сформовано автором

3.3 Програмна реалізація гри в жанрі стратегія

Як зазначалось в першому пункті даного розділу, комп’ютерна гра в жанрі стратегія, а саме піджанр стратегія в реальному часі, про який було розказано в третьому пункті другого розділу даної кваліфікаційної роботи, була зроблена завдяки ігровому рушію Unity та інтегрованому середовищу розробки Visual Studio.

Мета гри в даному піджанрі жанру стратегія полягає в тому, щоб перемогти всіх супротивників. Усього на карті генерується чотири ворожі ангари, кожен з яких містить по десять бойових машин. Кількість техніки для кожного ангара задається у методі IEnumerator SpawnCar().

Гравець має можливість створити власний ангар або кілька ангарів за допомогою кнопки Hangar, яка розташована в нижньому лівому куті екрана.

Керування камерою в грі здійснюється наступним чином:

- переміщення по карті – за допомогою клавіш W, A, S, D;
- обертання камери – за допомогою клавіш Q та E;
- прискорення руху камери – натисканням лівого Shift;
- масштабування (збільшення або зменшення масштабу) – за допомогою коліщатка миші.

Для вибору техніки гравець використовує ліву кнопку миші (ЛКМ). Після вибору однієї або декількох машин, їх можна переміщати по карті, натискаючи праву кнопку миші (ПКМ).

У нижньому правому куті розміщена кнопка Play, яка запускає відтворення головної музичної теми з гри "Танчики" — це своєрідне посилання на класичну гру з консолі Dendy. Кнопка виходу з гри знаходиться у верхньому правому куті екрана.

При запуску даної гри з’являється вікно, на якому зображено логотип самого рушія Unity, який говорить про те, що гра в жанрі стратегія, а саме стратегія в реальному часі була зроблена завдяки даному рушію.

Рисунок 3.16 – Вікно запуску гри Tactical Realistic Experimental - Shutkevich's Strategy

Джерело: сформовано автором

Після запуску гри, гравець бачить перед собою саму гру, де на екрані зображені сама карта гри, ландшафт, кнопки будування ангару чи ангарів гравця «Hangar», кнопка «Play», яка запускає відтворення головної музичної теми з гри "Танчики" та кнопка виходу з гри «Exit», яка знаходиться у верхньому правому куті екрана .

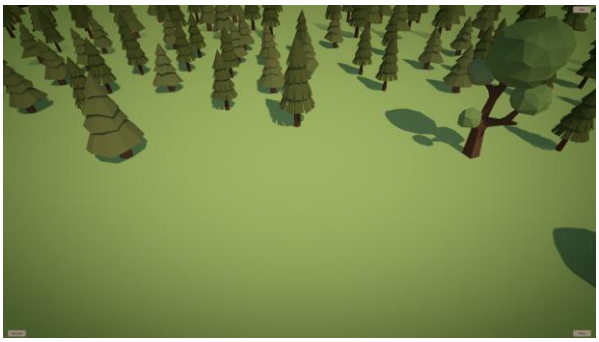


Рисунок 3.17 – Скріншот гри Tactical Realistic Experimental - Shutkevich's Strategy

Джерело: сформовано автором

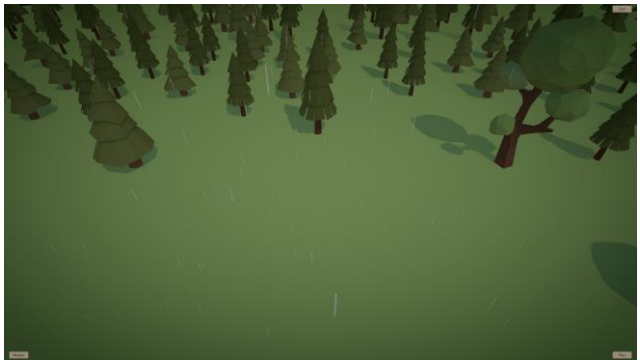


Рисунок 3.18 – Зміна погоди в грі

Джерело: сформовано автором

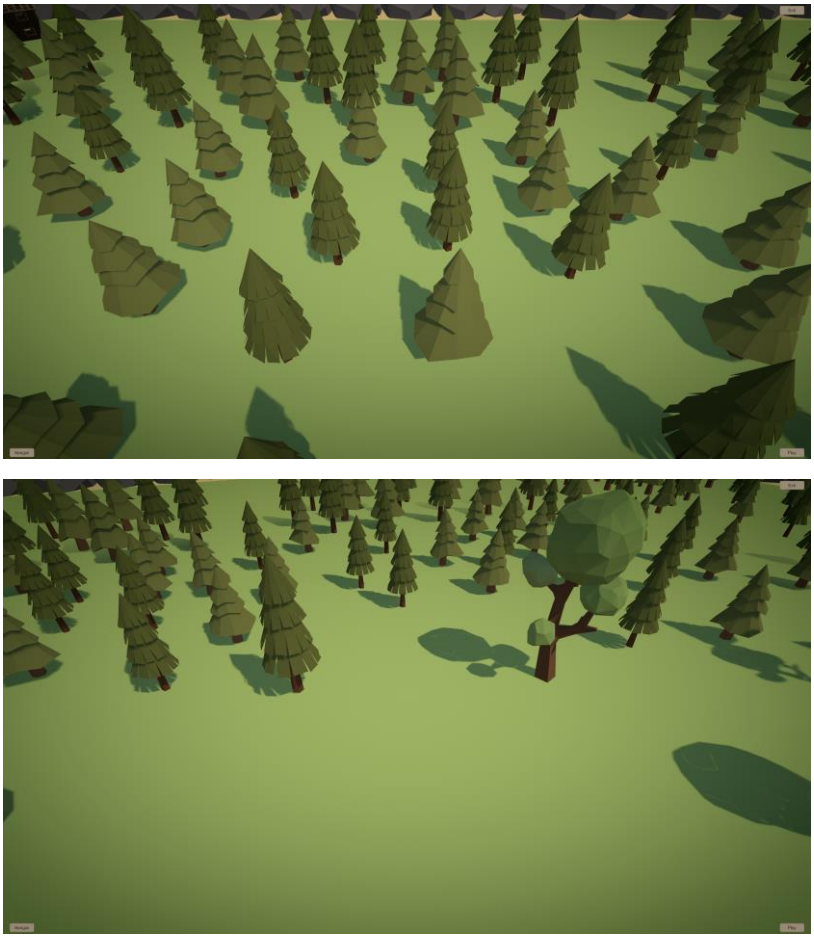
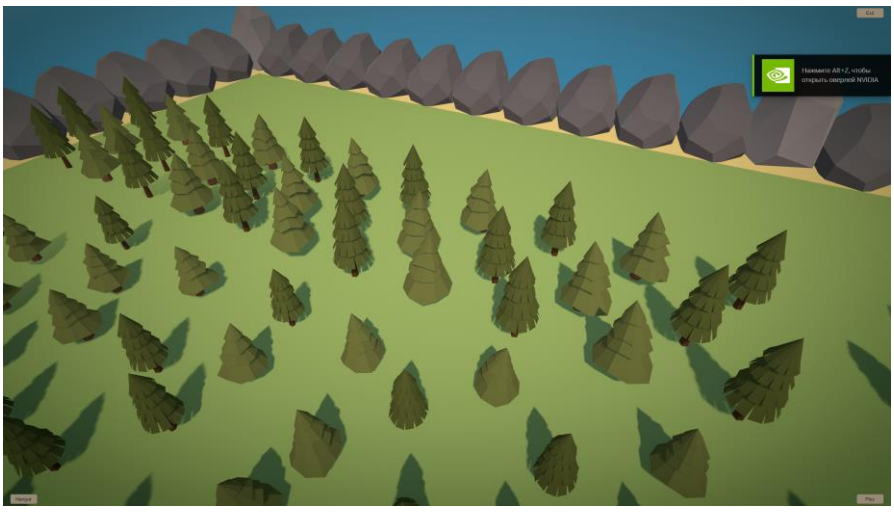


Рисунок 3.19 – 3.20 – Переміщення камери гри, завдяки клавішам W, A, S, D та поворот камери гри, завдяки клавішам Q та E

Джерело: сформовано автором

Далі, як говорилося на початку цього пункту, мета гравця полягає в тому, щоб перемогти всіх супротивників. На карті автоматично створюються чотири ворожі ангари, у кожному з яких розміщується по десять бойових одиниць.



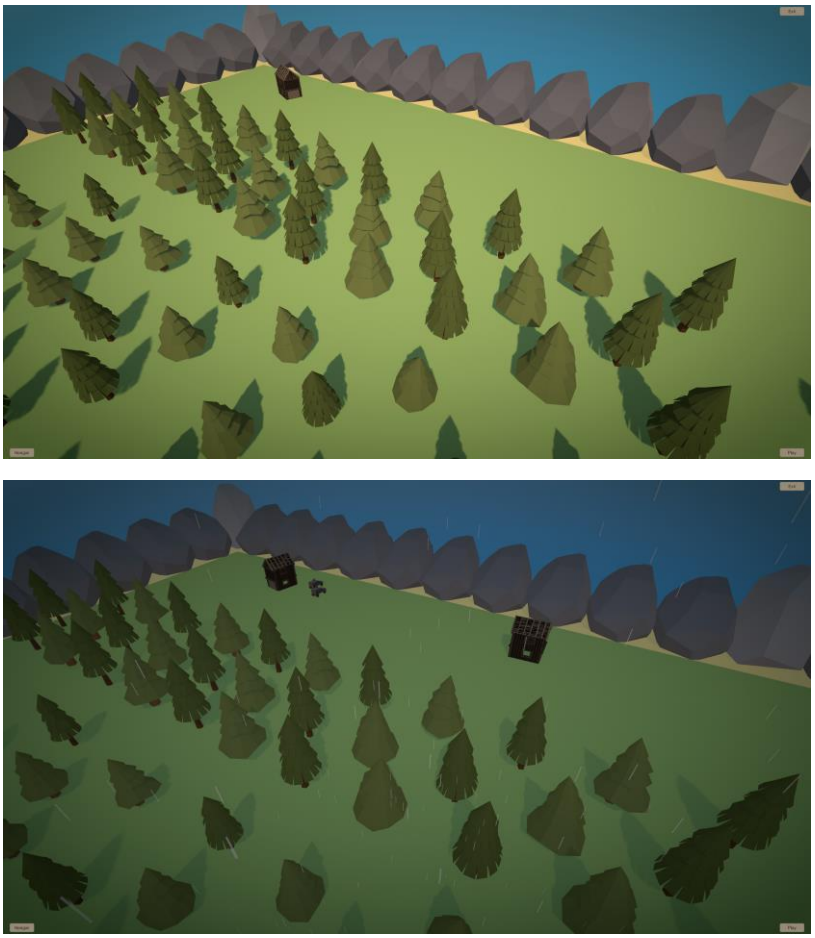
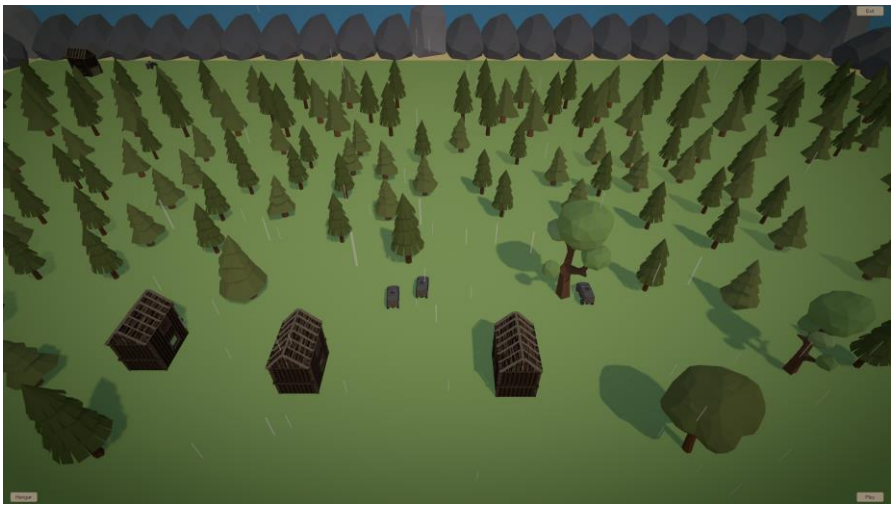


Рисунок 3.21 – 3.23 – Процес генерації ворожих ангарів з юнітами ворога

Джерело: сформовано автором

Коли гравець побачив, що на карті генеруються ворожі ангари, він в свою чергу генерує свої ангари з юнітами і обирає їх, щоб дійти до ворожих юнітів та досягнути головної мети гри, перемогти всіх супротивників і таким чином пройти гру.



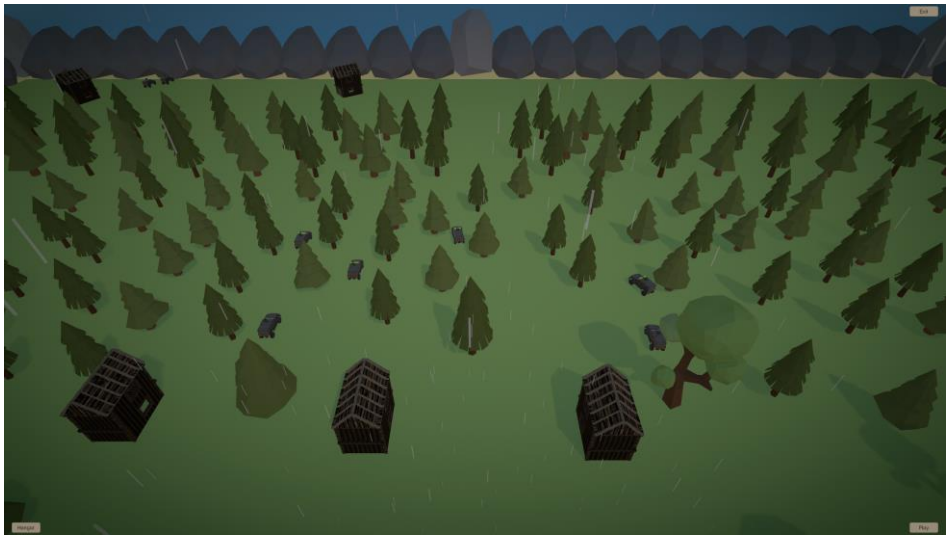


Рисунок 3.24 – 3.27 – Скріншоти геймплею гри Tactical Realistic Experimental - Shutkevich's Strategy

Джерело: сформовано автором

Висновки до розділу 3

Проаналізувавши третій розділ, можна зробити стосовно нього висновок, а саме:

- Були розглянуті основні програмні засоби, використані для розробки комп'ютерної гри в жанрі стратегія — Unity та Visual Studio. Unity слугувала головною платформою для створення ігрового середовища, розміщення об'єктів, реалізації логіки та інтерактивності, а також візуалізації сцени в реальному часі. Visual Studio використовувалась для написання коду мовою C#, налагодження логіки гри та оптимізації її роботи. Завдяки поєднанню гнучкості Unity і широкого функціоналу Visual Studio було досягнуто ефективної взаємодії між графікою та програмною частиною, що забезпечило якісну реалізацію основних елементів геймплею;

- Було здійснено практичне застосування програм під час створення гри: Unity використовувався як основна платформа для побудови ігрового середовища, зокрема розміщення ландшафту, ангарів і юнітів за допомогою моделей з Unity Asset Store. Це дозволило швидко сформувати карту з повноцінним візуальним наповненням. Visual Studio, у свою чергу, використовувалася для написання скриптів, що реалізовували ігрову логіку — керування юнітами, ангарами, погодою, камерою, звуками та взаємодією об'єктів. Завдяки поєднанню можливостей обох інструментів було забезпечено як візуальну, так і програмну частину гри;

- Була продемонстрована програмна реалізація комп'ютерної гри в жанрі стратегія, а саме в піджанрі стратегія у реальному часі (RTS), що передбачає одночасне керування юнітами без пауз. У межах гри реалізовано основні механіки RTS: автоматичне створення ворожих ангарів, генерація власних, керування бойовою технікою, система перемоги через знищення супротивників. Гру доповнено керуванням камерою, інтерактивним інтерфейсом і звуковим супроводом, що створює повноцінну RTS-атмосферу.

Таким чином, основні риси піджанру були успішно втілені як у механіці, так і у візуальній частині гри.

ВИСНОВКИ

Метою бакалаврської кваліфікаційної роботи була розробка гри на ігровому рушії Unity з поглибленим розкриттям можливості його фізичного рушія для створення симуляції рухів і взаємодій ігрових об'єктів.

В процесі виконання даної кваліфікаційної роботи було проведено аналіз ігрового рушія Unity. Розглянуто його функціональні можливості, підтримувані мови програмування, сфери застосування, а також основні інструменти, що використовуються при створенні ігор у жанрі стратегія. Крім того, здійснено порівняння Unity з іншими рушіями, що дозволило виявити їхні переваги та недоліки.

Ще в ході процесі виконання бакалаврської кваліфікаційної роботи було проаналізовано фізику та фізичні рушії в іграх жанру стратегія. Особливу увагу приділено ігровому штучному інтелекту, його ролі в управлінні юнітами та основним алгоритмам (деревам рішень, станів тощо). Також розглянуто піджанри жанру стратегія, з акцентованим аналізом на їх структуру та механіку.

Також в процесі виконання кваліфікаційної роботи було представлено опис розробки гри в жанрі стратегія, а саме стратегія в реальному часі (RTS). Розглянуто застосування Unity та Visual Studio, а також їх використання для реалізації основних механік гри. Детально описано взаємодію гравця з грою, включаючи створення ангарів, керування технікою (юнітами) в режимі реального часу, використання інтерфейсних елементів та управління камерою.

Після виконання кваліфікаційної роботи були вирішенні такі завдання:

1. Було введено опис ігрового рушія Unity, порівняння з іншими ігровими рушіями, знаходження переваг та недоліків;

2. Був проведений аналітичний опис фізики та фізичних рушіїв в іграх, визначення ігрового штучного інтелекту та розбором алгоритмів і методів, які в ньому використовуються, а також огляд та опис піджанрів жанру стратегія;

3. Була зроблена програмна реалізація і тестування комп`ютерної гри в жанрі стратегія, а саме піджанру стратегія в реальному часі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ігровий рушій Unity (українська версія). URL: https://uk.wikipedia.org/wiki/Unity_(%D1%96%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9)

2. Ігровий рушій Unity (англійська версія). URL: https://en.wikipedia.org/wiki/Unity_(game_engine)

3. Головний сайт Unity. URL: https://unity.com/

4. Ігровий рушій Rage (українська версія). URL: https://uk.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine

5. Ігровий рушій Rage (англійська версія). URL: https://en.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine

6. Стаття про ігровий рушій Rage на Linkedin. URL: https://www.linkedin.com/pulse/rage-kavin-a-uq5me/

7. Ігровий рушій Unreal Engine (українська версія). URL: https://uk.wikipedia.org/wiki/Unreal_Engine

8. Ігровий рушій Unreal Engine (англійська версія). URL: https://en.wikipedia.org/wiki/Unreal_Engine

9. Головний сайт Unreal Engine. URL: https://www.unrealengine.com/en-US

10. Офіційний портал розробників Unreal. URL: https://dev.epicgames.com/documentation/

11. Ігровий рушій EGO (українська версія). URL: https://uk.wikipedia.org/wiki/EGO_(%D1%80%D1%83%D1%88%D1%96%D0%B9)

12. Ігровий рушій EGO (англійська версія). URL: https://en.wikipedia.org/wiki/Codemasters#Technology

13. Сайт розробників рушія EGO Codemasters. URL: https://www.ea.com/ea-studios/codemasters

14. Ігровий рушій Anvil (українська версія). URL: https://uk.wikipedia.org/wiki/Anvil

15. Ігровий рушій Anvil (англійська версія). URL: https://en.wikipedia.org/wiki/Ubisoft_Anvil

16. Сторінка на сайті Ubisoft, присвячена технологіям компанії, зокрема Ubisoft Anvil. URL: https://www.ubisoft.com/en-us/company/how-we-make-games/technology

17. Фізика в іграх. Створення ігрового двигуна на основі фізичних процесів [Електронний ресурс]. URL: http://infotech-soccult.knukim.edu.ua/article/view/283971/278302

18. Ігровий штучний інтелект (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%86%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%88%D1%82%D1%83%D1%87%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D1%82

19. Ігровий штучний інтелект (англійська версія). URL: https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games

20. Сайт AiWisdom з статтями та дослідженнями в сфері ігрового штучного інтелекту. URL: http://www.aiwisdom.com/

21. Стратегічна відеогра (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D1%96%D1%87%D0%BD%D0%B0_%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D0%B3%D1%80%D0%B0

22. Стратегічна відеогра (англійська версія). URL: https://en.wikipedia.org/wiki/Strategy_video_game

23. Покрокова стратегія (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BA%D1%80%D0%BE%D0

%BA%D0%BE%D0%B2%D0%B0_%D1%81%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D1%96%D1%8F

24. Покрокова стратегія (англійська версія). URL: https://en.wikipedia.org/wiki/Strategy_video_game#Turn-based_strategy_(TBS)

25. Стратегія в реальному часі (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B0%D1%82%D0%B5%D0%B3%D1%96%D1%8F_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%D1%83_%D1%87%D0%B0%D1%81%D1%96

26. Стратегія в реальному часі (англійська версія). URL: https://en.wikipedia.org/wiki/Real-time_strategy

27. Симулятор менеджмену та будування (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D0%BC%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%BC%D0%B5%D0%BD%D1%82%D1%83_%D1%82%D0%B0_%D0%B1%D1%83%D0%B4%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

28. Симулятор менеджмену та будування (англійська версія). URL: https://en.wikipedia.org/wiki/Construction_and_management_simulation

29. Симулятор містобудування (українська версія). URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D1%8F%D1%82%D0%BE%D1%80_%D0%BC%D1%96%D1%81%D1%82%D0%BE%D0%B1%D1%83%D0%B4%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

30. Симулятор містобудування (англійська версія). URL: https://en.wikipedia.org/wiki/City-building_game

31. Michael Moore. Basics of Game Design. CRC Press, 2016. 400 p [Електронний ресурс]. URL: https://www.academia.edu/25954450/Basics_of_Game_Design

32. Mark J. P. Wolf. Encyclopedia of Video Games: A-L. ABC-CLIO, 2012. 763 p. [Електронний ресурс]. URL: https://www.researchgate.net/publication/328450209_Encyclopedia_of_Video_Games_The_Culture_Technology_and_Art_of_Gaming201368_Encyclopedia_of_Video_Games_The_Culture_Technology_and_Art_of_Gaming

33. Microsoft Visual Studio (українська версія). URL: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

34. Microsoft Visual Studio (англійська версія). URL: https://en.wikipedia.org/wiki/Visual_Studio

35. Сайт Visual Studio. URL: https://visualstudio.microsoft.com/