Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Дробний Олексій Юрійович*
(ПІБ)

академічної групи *121-19-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: Розробка ігрового застосунку на базі рушія Unity та мові програмування C#

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*Проф. Лактіонов І.С.*	95	відмінно	
розділів:				
спеціальний	*Проф. Лактіонов І.С.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*ст.викл. Мартиненко А.А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19-2* *Дробного О.Ю.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи Розробка ігрового застосунку на базі рушія Unity та мові програмування C#

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *проф. Лактіонов І.С.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Дробний О.Ю.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 84 с., 19 рис., 0 табл., 3 дод., 20 джерел.

Об'єкт розробки: ігровий застосунок в жанрі симуляції транспортування за допомогою ігрового рушія Unity на мові програмування C#

Мета кваліфікаційної роботи: створення ігрового застосунку, який буде має більшу функціональність порівняно зі схожими продуктами на ринку. Застосунок має на меті навчити гравців принципам логістики, управління ланцюгами поставок та обслуговування клієнтів за допомогою інтерактивного та захоплюючого ігрового процесу. Забезпечуючи реалістичне та динамічне середовище, гра спонукає гравців приймати стратегічні рішення та ефективно управляти своїми ресурсами, щоб задовольнити попит і підтримувати високий рівень задоволеності клієнтів

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціонування, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розроблення програмного забезпечення, проведений підрахунок вартості робот по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні ігрового застосунку, який буде мати більшу функціональність порівняно зі схожими продуктами на ринку, а також успішне проходження етапів розробки комп'ютерної гри.

Актуальність даного програмного продукту визначається відсутністю ігрових застосунків з аналогічним функціоналом, а також малою кількістю ігор в жанрі симуляції транспортування.

Список ключових слів: ПРОГРАМА, ІГРОВИЙ ЗАСТОСУНОК, ГРА, ІГРОВИЙ РУШІЙ, UNITY, АЛГОРИТМ, ПРОЕКТУВАННЯ.

ABSTRACT

Explanatory note: 84 pages., 19 fig., 0 table, 3 appendix, 20 sources.

Object of development: a game application in the genre of transportation simulation using the Unity game engine in the C# programming language

The purpose of qualifying work: to create a game application that will have more functionality than similar products on the market. The application aims to teach players the principles of logistics, supply chain management, and customer service through interactive and engaging gameplay. By providing a realistic and dynamic environment, the game encourages players to make strategic decisions and effectively manage their resources to meet demand and maintain high levels of customer satisfaction.

The introduction discusses the analysis and current state of the problem, specifies the purpose of the qualification work and its scope, provides a justification for the relevance of the topic and clarifies the task statement.

The first section analyzes the subject area, determines the relevance of the task and the purpose of the development, formulates the task statement, specifies the requirements for software implementation, technologies and software tools.

The second section analyzes the existing solutions, selects the platforms for development, designs and develops the program, describes the program operation, algorithm, and structure of its functioning, as well as the program call and loading, defines the input and output data, and characterizes the composition of the technical means parameters.

In the economic section, the complexity of software development is deter-mined, the cost of work and the time on creation of the program is calculated.

The practical significance lies in the creation of a game application that will have greater functionality than similar products on the market, as well as the successful completion of the stages of computer game development.

The relevance of this software product is determined by the lack of gaming applications with similar functionality, as well as a small number of games in the genre of transportation simulation.

List of keywords: PROGRAM, GAME APPLICATION, GAME, GAME ENGINE, UNITY, ALGORITHM, DESIGN.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК – персональний ком;

ААА – термін, що позначає клас високобюджетних комп'ютерних ігор;

ЗМІСТ

ВСТУП

Розвиток нашого глобального суспільства призвів до значної трансформації. Технологічний прогрес сьогодні пропонує безліч можливостей і доступ до величезної кількості інформації через Інтернет. Наші мобільні телефони та ноутбуки стали необхідними для того, щоб залишатися на зв'язку, відіграючи важливу роль у нашому повсякденному житті. Фотографії зберігаються в Інтернеті, контакти безпечно зберігаються в хмарі, а додатки соціальних мереж завжди під рукою. Безперечно, наш світ сьогодні є цифровим, а нові технології, такі як соціальні мережі, GPS-системи та штучний інтелект, змінюють нашу планету і створюють абсолютно нову цифрову сферу.

У цьому постійно мінливому ландшафті процвітає ігрова індустрія, вітаючи свіжі ідеї, інноваційні технології та видатні досягнення. Ігри вийшли за рамки ролі простого джерела розваг; тепер вони викликають яскраві емоції, залишають незабутні враження та дарують незабутній досвід. Навколо ігор виникли швидко зростаючі спільноти, професійні змагання і навіть види спорту. Розробка ігор перетворилася на мистецтво, захоплюючи гравців не лише самим ігровим процесом, але й припливом адреналіну, таємничими пригодами та психологічними викликами, які вони дають. Ігри пропонують втечу від повсякденних турбот, даруючи гравцям короткі моменти інтенсивних емоцій і переживань, коли вони долають складні виклики.

Хоча інформаційні технології, безперечно, ускладнили життя сучасних маркетологів, запровадивши нові терміни, поняття та принципи роботи, вони також відкрили нові можливості, розширивши горизонти бізнесу.

Актуальність даної роботи полягає в необхідності створення нових програмних продуктів, які рухають ігровий ринок і індустрію розваг в цілому, у відповідь на постійно мінливий ландшафт інформаційних технологій. Сьогодні існує незліченна кількість технічних можливостей для створення мультиплатформних ігор, які підходять для різних ігрових пристроїв, що

дозволяє виводити програмні продукти на один з найбільш швидкозростаючих ринків світу, що розвивається.

Об'єктом даного дослідження є технології, що використовуються при розробці та впровадженні ігрових програмних продуктів. Зокрема, предметом дослідження є комп'ютерні ігри - жанр відеоігор, які грають на персональних комп'ютерах (ПК), а не на ігрових приставках чи аркадних автоматах. Як і будь-який програмний продукт, гра проходить певні етапи від її концептуалізації до кінцевого продукту, кожен з яких має свої тонкощі у створенні та реалізації.

Метою даної кваліфікаційної роботи є створення ігрового застосунку, який буде мати більшу функціональність порівняно зі схожими продуктами на ринку, а також успішне проходження етапів розробки комп'ютерної гри. Таким чином, вона має на меті зробити свій внесок у постійно мінливий ландшафт ігрової індустрії.

Для досягнення мети кваліфікаційної роботи було окреслено наступні завдання:

- Провести ретельне дослідження ринку ігрової індустрії.
- Надати огляд основних програмних інструментів, що використовуються для розробки ігор.
- Проаналізувати основні етапи створення комп'ютерної гри.
- Розробити та дослідити комп'ютерну гру, використовуючи знання, отримані в результаті аналізу.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Ігрова індустрія - це багатомільярдна галузь, яка продовжує стрімко зростати з року в рік. Станом на 2021 рік світовий ігровий ринок коштує понад 170 мільярдів доларів, а до 2023 року, за прогнозами, сягне понад 218 мільярдів доларів[14]. Таке зростання зумовлене кількома факторами, зокрема зростанням популярності відеоігор як форми розваг, розвитком мобільних ігор та розширенням кіберспортивної індустрії.

Розробка ігор - найважливіший аспект ігрової індустрії. Розробники ігор створюють ігри, в які люди грають і насолоджуються ними, і вони відповідають за все - від дизайну та механіки гри до її художнього оформлення та звукового супроводу. Розробка ігор - це складний процес, який включає кілька етапів, зокрема препродакшн, виробництво та постпродакшн.

На етапі препродакшну розробники працюють над концепцією та дизайном гри. Це включає в себе мозковий штурм ідей, створення ігрових механік і систем, а також створення сюжетних ліній і персонажів. Розробники також можуть створювати прототипи або демо-версії, щоб протестувати свої ідеї та отримати зворотній зв'язок від потенційних гравців.

На етапі виробництва розробники створюють власне гру, включаючи код, художнє оформлення, звук і музику. Цей етап може тривати кілька років, залежно від складності гри та розміру команди розробників. На цьому етапі розробники також можуть працювати над додаванням нових функцій або контенту до гри.

Нарешті, на етапі пост-продакшну розробники працюють над виправленням помилок, оновленнями та патчами, щоб покращити продуктивність гри та вирішити будь-які проблеми, з якими можуть зіткнутися

гравці. Вони також можуть працювати над створенням нового контенту або функцій, щоб підтримувати інтерес гравців до гри.

В останні роки розробка ігор стала більш доступною для інді-розробників та невеликих студій завдяки наявності ігрових рушіїв та інструментів розробки. Це призвело до сплеску креативності та інновацій в індустрії, коли багато незалежних розробників створюють унікальні та цікаві ігри, які здобувають прихильників.

Однак розробка ігор також стикається з низкою викликів, серед яких зростання вартості розробки, необхідність постійно впроваджувати інновації та випереджати конкурентів, а також зростаючий тиск на монетизацію ігор за допомогою мікротранзакцій та інших засобів. Ці виклики призвели до зростання обізнаності про такі проблеми, як кризовий час, вигорання та необхідність покращення умов праці в індустрії.

Загалом, ігрова індустрія та розробка ігор перебувають у стані постійної еволюції та зростання, що зумовлено новими технологіями, зміною споживчих вподобань та інноваційними ідеями. Оскільки індустрія продовжує зростати і розширюватися, для розробників ігор буде важливо адаптуватися до нових тенденцій і викликів, зберігаючи при цьому фокус на створенні цікавих і захопливих ігор, які сподобаються гравцям.

Розробка ігор за допомогою ігрових рушіїв стає дедалі популярнішою останніми роками завдяки наявності потужних і доступних інструментів. Ігрові рушії надають розробникам основу для створення ігор, включаючи інструменти для проектування рівнів, реалізації ігрової механіки та створення ресурсів, таких як 3D-моделі та звукові ефекти.

Серед популярних ігрових рушіїв, які широко використовуються в індустрії, є Unreal Engine, CryEngine та Unity. Кожен з цих рушіїв має свої сильні та слабкі сторони, і розробники часто обирають один з них, виходячи зі своїх конкретних потреб та вподобань.

Unreal Engine[1], розроблений Epic Games, - це потужний ігровий рушій, який відомий своїми розширеними графічними можливостями та використанням у створенні масштабних ААА-ігор. Рушій також широко використовується у розробці додатків віртуальної та доповненої реальності.

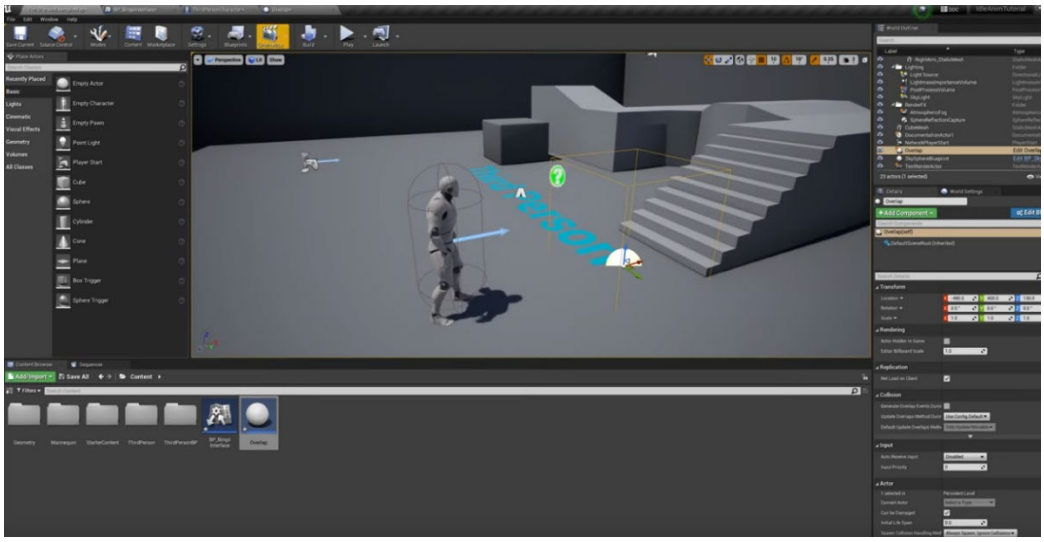

Рис. 1.1 Інтерфейс ігрового рушія Unreal Engine

CryEngine[2], розроблений компанією Crytek, - ще один потужний ігровий рушій, відомий своїми передовими графічними можливостями. Рушій часто використовують у розробці шутерів від першої особи та ігор з відкритим світом.

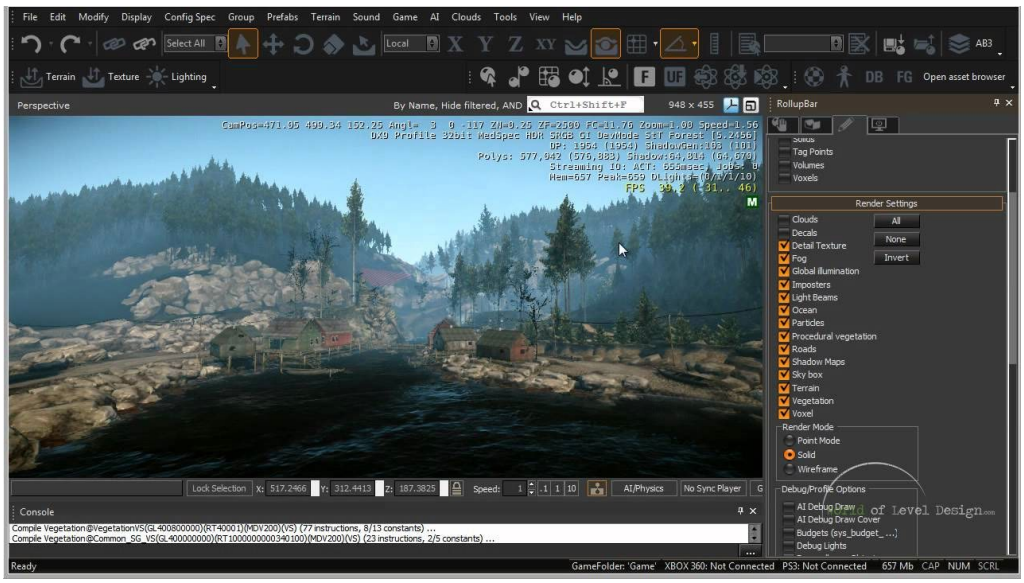

Рис. 1.2 Інтерфейс ігрового рушія Cry Engine

Ігровий рушій Unity[3] є одним з найпопулярніших ігрових рушіїв в індустрії і використовується як інді-розробниками, так і великими студіями. Вперше Unity був випущений у 2005 році компанією Unity Technologies і з того часу став популярним серед розробників завдяки простоті використання, універсальності та доступності.

Рис. 1.3 Інтерфейс ігрового рушія Unity

Однією з ключових переваг ігрового рушія Unity є його кросплатформеність. Unity дозволяє розробникам створювати ігри для широкого спектру платформ, включаючи ПК, консолі, мобільні пристрої та гарнітури віртуальної реальності. Така гнучкість робить його привабливим варіантом для розробників, які хочуть охопити якомога ширшу аудиторію.

Ще однією сильною стороною Unity є візуальний інструмент для написання сценаріїв, який називається Unity Editor. Цей інструмент дозволяє розробникам створювати та маніпулювати ігровими об'єктами та ресурсами за допомогою візуального інтерфейсу, що полегшує створення прототипів та

ітерації ігрових ідей. Редактор Unity також включає в себе ряд інших функцій, таких як моделювання фізики, мікшування звуку та інструменти анімації.

Unity також має велику та активну спільноту розробників, які створюють плагіни, ресурси та навчальні посібники і діляться ними. Ця екосистема, керована спільнотою, полегшує розробникам вивчення та використання рушія, а також можливість ділитися своїми напрацюваннями з іншими.

Однак, у ігрового рушія Unity є певні обмеження. Однією з найпоширеніших критичних зауважень щодо Unity є його робота на певних платформах, зокрема на мобільних пристроях. Крім того, стандартні 2D-інструменти рушія критикують за те, що вони менш потужні та зручні у використанні, ніж в інших ігрових рушіях.

Загалом, ігровий рушій Unity став популярним вибором для розробників ігор завдяки своїй універсальності, простоті використання та доступній ціні. Хоча він має деякі обмеження, його сильні сторони та активна спільнота роблять його чудовим варіантом для розробників, які прагнуть створювати ігри для широкого спектру платформ.

Зараз на ринку комп'ютерних ігор у жанрі ігор симуляцій транспортування представлені такі ігри: Mini motorways, mini metro, Transport fever.

Mini Motorways[4] - це мінімалістична стратегічна гра, розроблена та видана Dinosaur Polo Club. Гра ставить перед гравцями завдання спроектувати ефективну транспортну мережу для міста, що зростає, з кінцевою метою мінімізувати затори на дорогах і підтримувати безперебійну роботу міста. Гравці повинні з'єднати дороги з будівлями та керувати потоком транспорту, щоб не допустити заторів. Гра має просту, але стильну графіку та розслабляючий саундтрек, і отримала позитивні відгуки за захопливий ігровий процес та унікальний погляд на жанр транспортного симулятора.

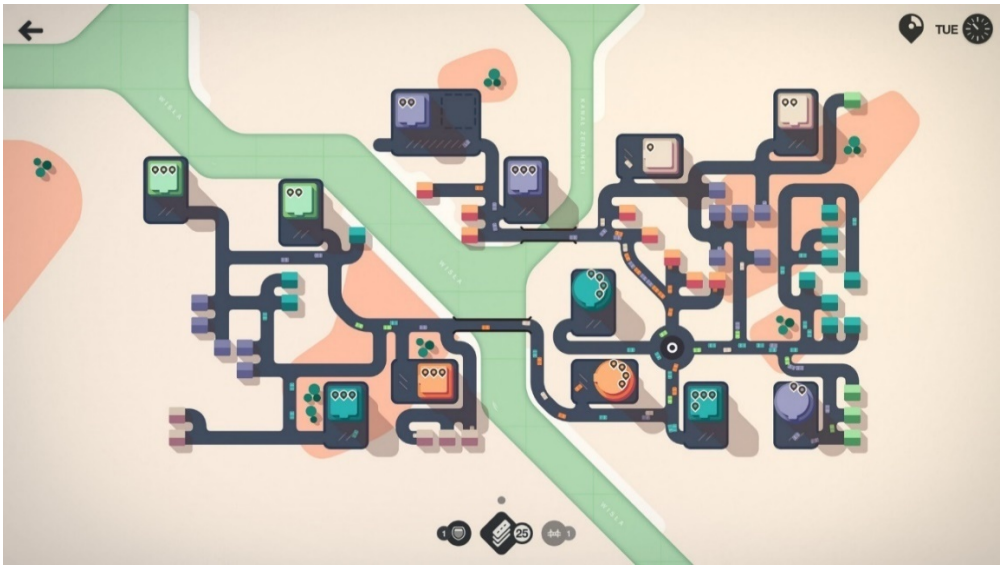

Рис. 1.4 Скріншот з гри Mini Motorways

Mini Metro[4] - ще одна мінімалістична стратегічна гра, розроблена і видана Dinosaur Polo Club. Як і Mini Motorways, гра ставить перед гравцями завдання спроектувати ефективну транспортну мережу, але в цьому випадку основна увага приділяється управлінню системою метрополітену. Гравці повинні з'єднати станції з лініями метрополітену та керувати потоком пасажирів, щоб запобігти переповненню та затримкам. Гра містить безліч різноманітних карт, створених на основі реальних міст, і отримала високу оцінку за інтуїтивно зрозумілий геймплей та стильне візуальне оформлення.

Рис. 1.5 Скріншот з гри Mini Metro

Transport Fever[5] - це транспортний симулятор, розроблений компанією Urban Games і виданий Good Shepherd Entertainment. У грі гравці повинні будувати та управляти транспортними мережами в широкому діапазоні епох, від парових поїздів до сучасних авіалайнерів. Гравці повинні з'єднувати міста та галузі промисловості, керувати потоками товарів і пасажирів, а також збалансовувати потреби різних регіонів і галузей. Гра вирізняється деталізованою графікою та широким спектром транспортних засобів і будівель, а також отримала позитивні відгуки за глибокий ігровий процес і увагу до деталей.

Рис. 1.6 Скріншот з гри Transport Fever

Метою цього дослідження є розробка гри в жанрі симуляції транспортування, а також впровадження інноваційних підходів, які покращують залучення гравців, прийняття стратегічних рішень та загальний ігровий досвід:

- Залучення гравців: Завдяки захоплюючим візуальним ефектам, захоплюючому звуковому оформленню та інтерактивним елементам гра має на меті захопити якомога більше гравцівю. Стилізована графіка та

звукові ефекти сприятимуть посиленню відчуття присутності, дозволяючи гравцям відчути зв'язок з віртуальним середовищем.

- Прийняття стратегічних рішень: У грі буде використано складний алгоритм, який генерує динамічні та складні сценарії для гравців. Завдяки широкому спектру змінних, таких як мінливий попит, транспортні схеми та різноманітні перешкоди, гравців заохочуватимуть мислити стратегічно і приймати обґрунтовані рішення для оптимізації своїх мереж доставки. Включення різних рівнів складності забезпечить відповідний виклик для гравців з різними рівнями навичок.
- Ігровий досвід: Щоб збагатити ігровий процес, гра буде включати елементи дослідження та розвитку. Гравці матимуть можливість розблокувати нові міста, якщо вони успішно виконають поставлені завдання та досягнуть певних етапів. Крім того, введення різноманітних підсилювачів з унікальними можливостями та обмеженнями додасть глибини ігровому процесу, дозволяючи гравцям експериментувати з різними стратегіями та підходами.

1.2 Призначення розробки та галузь застосування

Одне з найголвніших застосувань для гри з управління доставкою з використанням Unity та C# - це розвага. Гра зацікавить гравців, які полюбляють стратегічні та управлінські ігри, а також тих, хто цікавиться логістикою та управлінням ланцюгами поставок. Ігровий процес включає в себе розробку та оптимізацію маршрутів доставки, а також управління логістичними мережами. Гра містить низку викликів і перешкод, таких як перекриття доріг і обмеженість ресурсів, щоб тримати гравців у тонусі і кидати їм виклик.

На додаток до свого потенціалу як форми розваги, така гра може також мати освітнє застосування. Наприклад, її можна використовувати для навчання студентів логістиці та управлінню ланцюгами поставок, або для того, щоб

допомогти їм зрозуміти, як працюють системи доставки пошти. Залучаючи студентів у веселий та інтерактивний спосіб, гра може допомогти зробити складні концепції більш доступними та зрозумілими.

Таку гру також можна використовувати для професійного навчання. Наприклад, логістичні та кур'єрські компанії можуть використовувати гру для навчання нових працівників основам доставки пошти та управління логістикою. Використовуючи ігрове середовище, працівники могли б навчатися в більш цікавий і запам'ятовуючий спосіб, що допомогло б їм ефективніше запам'ятовувати інформацію.

Нарешті, таку гру можна використовувати в дослідницьких цілях. Наприклад, дослідники можуть використовувати гру для вивчення того, як люди приймають рішення щодо логістики та управління ланцюгами поставок, або для тестування різних стратегій оптимізації систем доставки пошти. Аналізуючи дані, зібрані в грі, дослідники можуть отримати уявлення про те, як люди підходять до вирішення складних завдань, і використовувати цю інформацію для вдосконалення реальних систем логістики та управління ланцюгами поставок.

Отже, ця гра може мати широкий спектр потенційних застосувань і аудиторій. Незалежно від того, чи використовується вона як форма розваги, в освітніх цілях, для професійної підготовки або для досліджень, гра може забезпечити цікаву і складну ігрову механіку, яка перевіряє стратегічне мислення гравців і їхні навички вирішення проблем.

1.3. Підстава для розробки

В кінці навчання, студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему «Розробка ігрового застосунку на базі рушія Unity та мові програмування C#» є наказ по Національному технічному університету «Дніпровська політехніка» від 16.05.2023р. № 350-с.

1.4. Постановка завдання

Метою розробки гри з управління доставкою є створення захопливого та освітнього досвіду, який імітує виклики управління бізнесом з доставки. Гра має на меті навчити гравців принципам логістики, управління ланцюгами поставок та обслуговування клієнтів за допомогою інтерактивного та захоплюючого ігрового процесу. Забезпечуючи реалістичне та динамічне середовище, гра спонукає гравців приймати стратегічні рішення та ефективно управляти своїми ресурсами, щоб задовольнити попит і підтримувати високий рівень задоволеності клієнтів.

Розробка гри потребуватиме програмної платформи, здатної імітувати динаміку системи поштової доставки. Розроблена гра буде використовуватися для управління різними об'єктами, включаючи маршрути доставки пошти, транспортні засоби, розподільчі центри та склади.

Метою вихідної інформації буде забезпечення зворотного зв'язку з учасниками щодо їхньої діяльності та полегшення прийняття рішень. Протягом ігрового процесу учасники здобудуть знання, які дозволять їм відстежувати ключові показники ефективності, виявляти вузькі місця та неефективність, а також приймати стратегічні рішення щодо планування маршрутів та розподілу ресурсів.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для реалізації поставленого завдання потрібно програмно реалізувати:

- Управління транспортними засобами: Гра повинна дозволяти гравцям керувати парком транспортних засобів доставки, включаючи призначення водіїв, планування доставок, відстеження місцезнаходження транспортних засобів.

- Планування маршрутів: Гра повинна дозволяти гравцям планувати маршрути доставки, беручи до уваги такі фактори, як трафік, погода та часові обмеження.
- Підрахунок балів та винагороди: Гра повинна передбачати систему підрахунку балів, яка винагороджує гравців за успішну доставку, задоволення потреб клієнтів та ефективне управління ресурсами.
- Реалістична симуляція: Гра повинна забезпечувати реалістичну симуляцію операцій доставки.

1.5.2. Вимоги до інформаційної безпеки

Для збереження інформаційної безпеки гра повинна реалізовувати такі вимоги:

- Конфіденційність даних: Гра повинна забезпечувати конфіденційність і безпеку даних користувача, включаючи особисту інформацію та дані про ігровий процес.
- Безпечна автентифікація: Гра повинна реалізовувати безпечні механізми автентифікації, щоб гарантувати, що тільки авторизовані користувачі мають доступ до гри.
- Захист від шкідливого програмного забезпечення: Гра повинна бути розроблена таким чином, щоб мінімізувати ризик зараження шкідливим програмним забезпеченням і захистити дані користувача від вірусів та інших форм шкідливого програмного забезпечення.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для повноцінного функціонування програмного забезпечення необхідні:

- персональний комп’ютер,
- клавіатура;

- миша
- процесор з частотою не нижче 2.5 GHz
- 4 гб операційної пам’яті
- відеокарта не нижче NVIDIA GeForce GTX 560 Ti
- 1 гб пам’яті на жорсткому диску

1.5.4. Вимоги до інформаційної та програмної сумісності

Розроблювана гра сумісна з операційними системами Windows 8, 10 чи 11 та macOS 64-розрядних версій з встановленим DirectX 10 чи вище.

Додаток має бути розроблений за допомогою мови програмування C#.

Додаткові файли, що потрібні для роботи генеруються в процесі гри мають знаходитись в одній папці з виконуваним файлом.

Файли для запису поточного стану гри мають знаходитись в директорії постійних даних операційної системи.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення програми

Враховуючи недоліки, в існуючих на ринку аналогах, завдання даного проекту полягає в створенні більш різноманітного на ігрові механіки проекту, в якому більше уваги приділено стратегічному плануванню та управлінню ресурсами.

Гра повинна вимагати від гравців стратегічного планування маршрутів враховуючи такі фактори, як відстань, трафік і часові вікна доставки. Це перевірить здатність гравців думати наперед і приймати оптимальні рішення для забезпечення своєчасної і точної доставки.

Ефективне управління ресурсами має вирішальне значення в будь-якій стратегічній грі. Гра повинна включати управління парком транспортних засобів, ефективний розподіл ресурсів, а також модернізацію або розширення інфраструктури доставки. Щоб досягти успіху, гравці повинні збалансувати витрати, швидкість та ефективність доставки.

Як і в багатьох симуляторах чи головоломках, у грі про доставку пошти гравці можуть зіткнутися з різними проблемами та перешкодами, які їм доведеться долати. Це будуть затори на дорогах, перекриття доріг або часові обмеження. Гравцям потрібно буде творчо мислити і знаходити рішення, щоб забезпечити безперебійну доставку пошти.

Основною метою гри є оптимізація операцій з доставки пошти шляхом мінімізації часу доставки і максимального задоволення потреб клієнтів.

Також, розроблювана гра, може мати освітню цінність, надаючи гравцям уявлення про логістику та роботу системи доставки пошти. Вона також може навчати концепціям, пов'язаним з міським плануванням, управлінням дорожнім рухом і розподілом ресурсів.

Загалом, гра, присвячена доставці пошти, може поєднувати стратегічне планування, управління ресурсами, розв'язання проблем і оптимізаційні завдання, пропонуючи гравцям цікавий і навчальний досвід.

2.2 Опис застосованих математичних методів

При розробленні даної кваліфікаційної роботи були використані наступні математичні методи:

Теорія сітчастої ігрової мапи[6]: Теорія, що лежить в основі сітчастої ігрової мапи, полягає в поділі ігрового світу на сітку однакових за розміром клітинок або тайлів. Кожна клітинка представляє певну область або одиницю простору в ігровому середовищі. Така структура сітки забезпечує структурований та організований спосіб представлення ігрового світу, що полегшує розробникам ігор реалізацію руху, позиціонування та взаємодії між ігровими елементами.

Пошук найкоротшого шляху A*[7]: в інформатиці та математиці, алгоритм пошуку за першим найкращим збігом на графі, що знаходить маршрут із найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої).

Цей алгоритм було вперше описано 1968 року Пітером Гартом, Нільсом Нільсоном і Бертрамом Рафаелем. Це по суті було розширення алгоритму Дейкстри[8], створеного 1959 року. Новий алгоритм досягав більш високої продуктивності (за часом) за допомогою евристики. У їхній роботі він згадується як "алгоритм A". Але оскільки він обчислює найкращий маршрут для заданої евристики, його було названо A*.

2.3 Опис використаної архітектури та шаблонів проектування

При розробці цієї кваліфікаційної роботи використовується мова програмування C#, яка є об'єктно-орієнтованою. Цей підхід активно використовується під час створення програми, надаючи можливість використовувати методи та класи цієї мови програмування.

З метою зручності, класи, відповідальні за конкретні функції, були виділені в окремі файли, які названо відповідно до виконуваних ними функцій.

Також в цій роботі широко застосовуються таки принципи программування[17] як:

1. Модульність:
    - Модульність - це практика розбиття системи на менші, автономні модулі або компоненти.
    - Кожен модуль фокусується на певній функціональності або особливості системи.
    - Модулі розроблені так, щоб бути незалежними, що полегшує розробку, тестування та обслуговування.
    - Модульність сприяє повторному використанню коду та дозволяє краще організувати та співпрацювати між розробниками.
2. Інкапсуляція:
    - Інкапсуляція - це процес приховування внутрішніх деталей реалізації модуля або об'єкта і викриття лише необхідних інтерфейсів або методів.
    - Це допомагає контролювати доступ до внутрішнього стану та поведінки об'єкта, запобігаючи ненавмисним модифікаціям або залежностям.
    - Інкапсуляція покращує ремонтопридатність коду, зменшуючи вплив змін в одній частині системи на інші частини.

- Інкапсулюючи пов'язані дані та методи всередині об'єкта, ви можете досягти більш узгодженої та зрозумілої структури коду.

3. Розділення завдань:
    - Поділ проблем - це практика поділу програмної системи на окремі секції, кожна з яких відповідає за певний аспект або проблему.
    - Кожна секція відповідає за окрему і чітко визначену функціональність або набір функціональних можливостей.
    - Виокремлюючи проблеми, ви можете досягти кращої організації коду, читабельності та ремонтопридатності.
    - Зміни, внесені в одну проблему, мають мінімальний вплив на інші проблеми, що полегшує обслуговування та модифікацію.
    - У контексті стратегічної гри ви можете виокремити такі проблеми, як обробка вхідних даних, ігрова логіка, рендеринг, користувацький інтерфейс.

Архітектура цього проєкту дотримується SOLID[9] принципів. SOLID це абревіатура, яка представляє набір принципів для проектування програмних систем, які є гнучкими, підтримуваними і простими для розуміння. Кожна буква в SOLID означає певний принцип:

- Принцип єдиної відповідальності (Single Responsibility Principle, SRP)
- Принцип відкритості-закритості (Open-Closed Principle, OCP)
- Принцип заміщення Ліскова (LSP)
- Принцип розділення інтерфейсів (Interface Segregation Principle, ISP)
- Принцип інверсії залежностей (Dependency Inversion Principle, DIP)

У кваліфікаційній роботі застосовуються такі патерни програмування[10]: Singleton, Observer та State, при розробці проекту. Паттерн Singleton - це паттерн створення, який забезпечує створення лише одного екземпляру класу та надає глобальну точку доступу до нього. Він корисний у сценаріях, коли один об'єкт

повинен координувати дії всієї системи. Паттерн Observer - це поведінковий патерн, який встановлює зв'язок між об'єктами типу "один до багатьох", коли зміни в одному об'єкті автоматично відображаються в інших залежних об'єктах. Цей патерн є корисним у сценаріях, де декілька об'єктів повинні бути сповіщені та оновлені, коли стан об'єкта змінюється. Нарешті, патерн State - це ще один поведінковий патерн, який дозволяє об'єкту змінювати свою поведінку, коли змінюється його внутрішній стан. Він дозволяє об'єкту виглядати так, ніби він змінив свій клас. Цей патерн корисний у системах, де поведінка об'єкта залежить від його внутрішнього стану, а переходами станів потрібно керувати динамічно. Використовуючи ці патерни програмування, кваліфікаційна робота демонструє, як вони покращують повторне використання коду, його супроводжуваність та гнучкість при розробці проекту.

2.4 Опис використаних технологій та мов програмування

Програмне забезпечення для кваліфікаційної роботи розроблено на мові програмування C#, за допомогою ігрового рушія Unity у середовищі розробки JetBrains Rider 2023.1.

Основною мовою програмування є C#[11], ефективна і широко прийнята мова для створення додатків на різних платформах. C# забезпечує надійний та об'єктно-орієнтований підхід до програмування. C# широко використовується в різних галузях, включаючи розробку програмного забезпечення, розробку ігор, веб-розробку та розробку корпоративних додатків. Вона має велику та активну спільноту, велику документацію, бібліотеки та фреймворки, що полегшує розробникам навчання, співпрацю та створення надійних додатків.

Загалом, C# є універсальною мовою програмування з багатим набором функцій, що дозволяє використовувати зазначену мову програмування під час досягнення поставленої мети. Це було встановлено за критеріями: продуктивності та ремонтопридатності.

Проект розробляється з використанням ігрового рушія Unity, який відомий своєю універсальністю та простотою використання у створенні інтерактивного ігрового досвіду. Unity пропонує повний набір інструментів та функцій для розробки ігор, включаючи рендеринг, фізику, сценарії та управління ресурсами.

Для написання та управління кодом в кваліфікаційній роботі використовується JetBrains Rider, інтегроване середовище розробки (IDE), спеціально розроблене для розробки на C#. Rider пропонує такі функції, як завершення коду, налагодження, рефакторинг та інтеграцію контролю версій, які підвищують продуктивність та якість коду[16].

Git та GitHub використовуються для контролю версій та спільної розробки. Git - це розподілена система контролю версій, яка дозволяє ефективно відстежувати зміни, розгалуження та об'єднання коду. GitHub слугує хостинговою платформою для Git-репозиторіїв, забезпечуючи безперешкодну співпрацю між розробниками, обмін кодом та управління проектами[17].

Використовуючи ці технології, проект демонструє навички використання стандартних інструментів та фреймворків для створення надійних, ефективних та спільних програмних проектів. Поєднання C#, Unity, JetBrains Rider та Git/GitHub забезпечує міцну основу для розробки та управління проектом, уможливлюючи ефективну командну роботу, керування версіями коду та безперешкодну інтеграцію функцій.

2.5 Опис структури програми та алгоритмів її функціонування

В загальному вигляді, ігровий цикл виглядає наступним чином:

Рис. 2.1 Узагальнена схема ігрового циклу

Логіка роботи програми структурована на кілька компонентів, які взаємодіють один з одним. Основні компоненти наведено нижче.

А) Ігровий менеджер: Ігровий менеджер слугує основним компонентом, який керує логікою гри, зв’язуючи усі інші компоненти.

Б) Інтерфейс користувача: Цей компонент відповідає за графічний інтерфейс гри, дозволяючи гравцям взаємодіяти з ігровими елементами, такими кнопки, побудова маршрутів, вибір підсилювачів.

В) Генерація замовлень: Цей компонент відповідає за створення поштових відправлень з певними адресатами. Він створює різноманітні поштові відправлення, які повинні бути доставлені до відповідних пунктів призначення в грі.

Г) Управління мережею доріг: Цей компонент дозволяє гравцям будувати і змінювати дорожню мережу, включаючи розміщення маршрутів та їх редагування, для створення ефективних маршрутів доставки.

Ґ) Генерація мапи: Цей компонент відповідає за генерацію міста, доріг та навколишнього середовища.

Д) Система підсилювачів: Цей компонент відповідає за винагороду гравця підсилювачами, які урізноманітнюють ігровий процес та додають більше стратегічного планування.

Е) Система підрахунку балів: Компонент системи підрахунку балів розраховує та оновлює бал гравця на основі кількості успішних доставок.

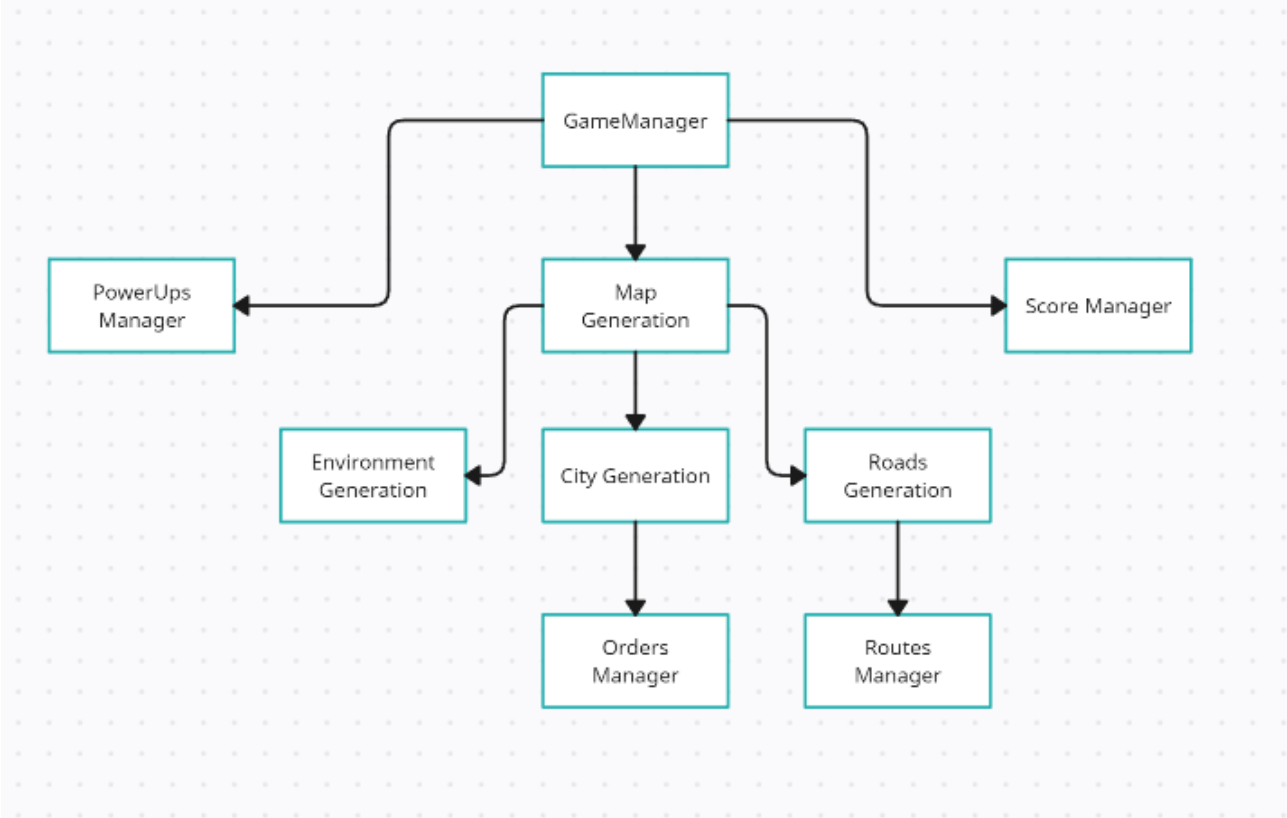

Рис 2.2 Логіка взаємодії компонентів

Для забезпечення безперебійної роботи програми та отримання реалістичного ігрового досвіду застосовується алгоритмічна схема для вирішення проблеми ефективного управління маршрутами доставки пошти:

А) Алгоритм генерації міста: цей алгоритм забезпечує збалансоване генерування міста, залежно від рівня складності. В основі цього підходу лежить Теорія сітчастої ігрової мапи[6].

Б) Алгоритм генерації замовлень: Алгоритм формування пошти забезпечує збалансований розподіл поштових відправлень з різними пунктами призначення. Він враховує такі фактори, як рівень складності, продуктивність гравця та проходження гри, щоб генерувати поштові відправлення, які пропонують відповідний рівень виклику та залучення.

В) Алгоритм оптимізації маршруту: Цей алгоритм спрямований на оптимізацію маршрутів доставки, обраних гравцями. Він використовує алгоритм пошуку A*[7], щоб знайти найкоротші та найефективніші шляхи між відповідними пунктами призначення. Алгоритм використовується для більш зручної побудови маршрутів доставки.

Обрані алгоритми та їхні схеми добре зарекомендували себе і широко використовуються в подібних іграх та симуляторах. Вони забезпечують ефективне та дієве вирішення проблеми управління маршрутами доставки пошти та генерації мапи. Ці алгоритми забезпечують баланс між реалістичним та цікавим ігровим процесом, дозволяючи гравцям розробляти стратегії та оптимізувати свої маршрути, стикаючись із викликом вчасної доставки пошти.

2.5.1 Back-end технології

Не застосовуються

2.5.2 Front-end технології

Не застосовуються

2.6 Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними в розроблюваному проекті, є натискання кнопок користувачем, а саме: натискання кнопок інтерфейсу, побудова маршрутів та вибір відповідних винагород

Вихідними даними є файли збереження гри, які створюються під час першого запуску гри та перед виходом з неї. До цих файлів належать:

- Файл збереження налаштувань аудіо (*.json)
- Файл збереження підсилювачів (*.json)

- Файл збереження поштових автомобілів (*.json)
- Файл збереження обраного режиму гри (*.json)
- Файл збереження очок, набраних гравцем (*.json)
- Файл збереження міста (*.json)
- Файл збереження обраної мови (*.json)
- Файл збереження поштових замовлень (*.json)
- Файл збереження поштових маршрутів (*.json)
- Файл збереження ігрових налаштувань (*.json)

Застосування окремих файлів для збереження полегшує та прискорює роботу з ними. Всі ці файли були зашифровані, для того щоб гравець не міг нічого змінити у файлах гри.

2.7 Опис роботи програмного продукту

2.7.1 Використані технічні засоби

Розроблювана гра оптимізована для роботи на більшості персональних комп'ютерів. Тому для її запуску вистачає мінімальних ресурсів комп'ютера. Для роботи необхідні:

- Персональний комп'ютер, чи ноутбук;
- Клавіатура;
- Миша;
- 4 гб операційної пам’яті;
- 1 гб пам’яті на жорсткому диску.

2.7.2 Використані програмні засоби

Це програмне забезпечення забезпечує стабільну роботу на комп'ютерах, що використовують операційні системи Windows XP, 7, 8 або 10 у версії або 64 біт, а також macOS. Для правильної роботи також необхідно мати встановлений DirectX 10 чи вище.

В папці з виконуваним файлом програми має знаходитись має знаходитись ще 2 файли – UnityCrashHanlder64.exe – файл оброблювач збоїв системи, та UnityPlayer.dll – призначений для коректної роботи гри. Обидва файли генеруються при компіляції програми.

2.7.3 Виклик та завантаження програми

Для запуску програми необхідно виконати файл PackageRush.exe, який знаходиться в папці з програмою. Після запуску програми відкриється головне меню гри(рис. 2.2).

Рис. 2.3 Головне меню гри

2.7.4 Опис інтерфейсу користувача

Після запуску гри відкривається головне меню програми(рис 2.2). Ігровий інтерфейс меню складається з чотирьох пунктів: Продовжити, Ігрові режими, налаштування та вихід.

Продовжити: Ця опція дозволяє гравцеві продовжити гру з того місця, на якому він зупинився.

Режими гри: Ця опція надає вибір різних ігрових режимів або варіантів ігрового процесу, з яких гравець може вибирати.

Налаштування: Ця опція дозволяє гравцеві отримати доступ до різних налаштувань. Вона включає такі опції, як налаштування звуку, графіки, конфігурація елементів керування, вибір мови та інші параметри.

Вийти: Ця опція дозволяє гравцеві вийти з гри і повернутися до операційної системи.

При натисканні кнопки «Ігрові режими» відкриється нове вікно з можливістю вибрати один із запропонованих режимів.

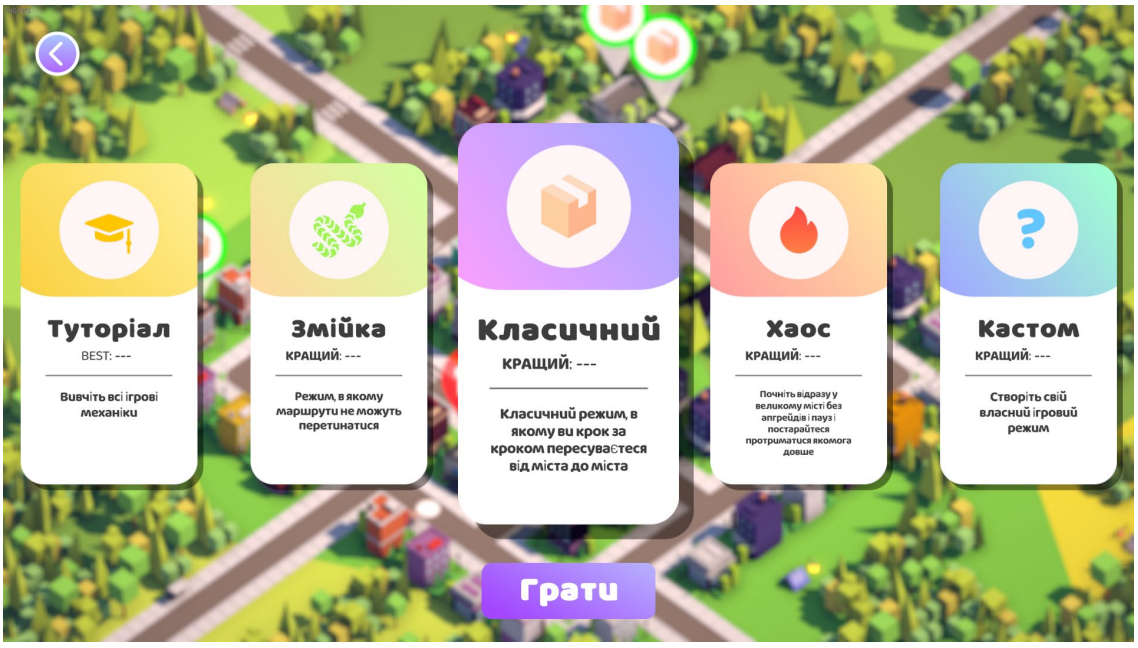

Рис 2.4 Вікно вибору ігрових режимів

Гравцеві на вибір пропонується один з п'яти режимів:

- Туторіал – режим, в якому гравця знайомлять з усіма ігровими механіками.
- Змійка – режим, в якому потові маршрути не можуть перетинатись.
- Класичний – класичний режим з побудовою поштових маршрутів та поступовим пересуванням від міста до міста.

- Хаос – режим, в якому гравець одразу починає в найбільшому місті. В цьому режимі відсутні пауза та підсилювачі.
- Кастом – режим, в якому гравець може сам обрати за якими правилами він хоче грати.

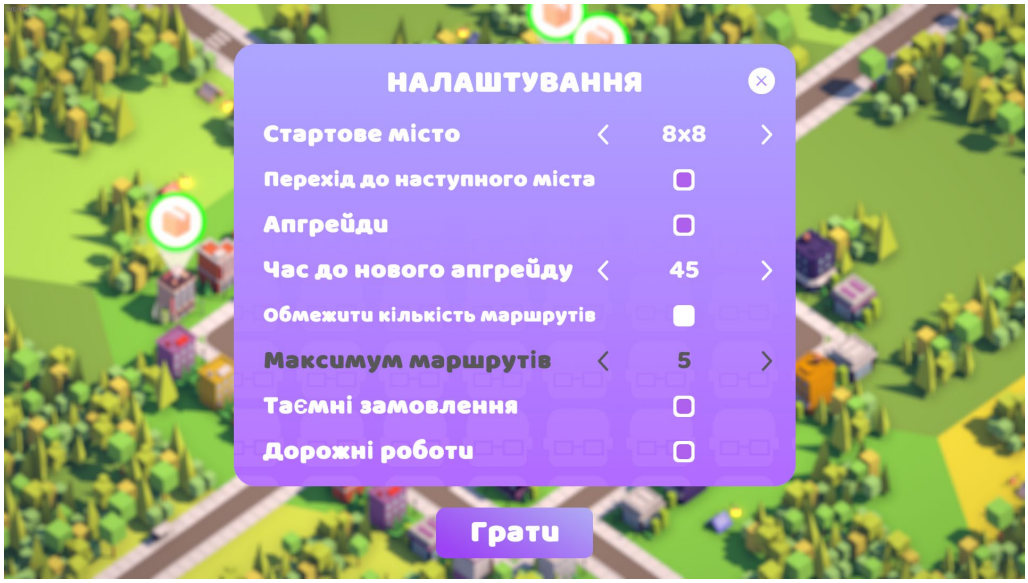

Рис 2.5 Вікно налаштування ігрового режима «Кастом»

При натисканні кнопки «Налаштування» у головному меню гри, відкривається вікно налаштування гри.

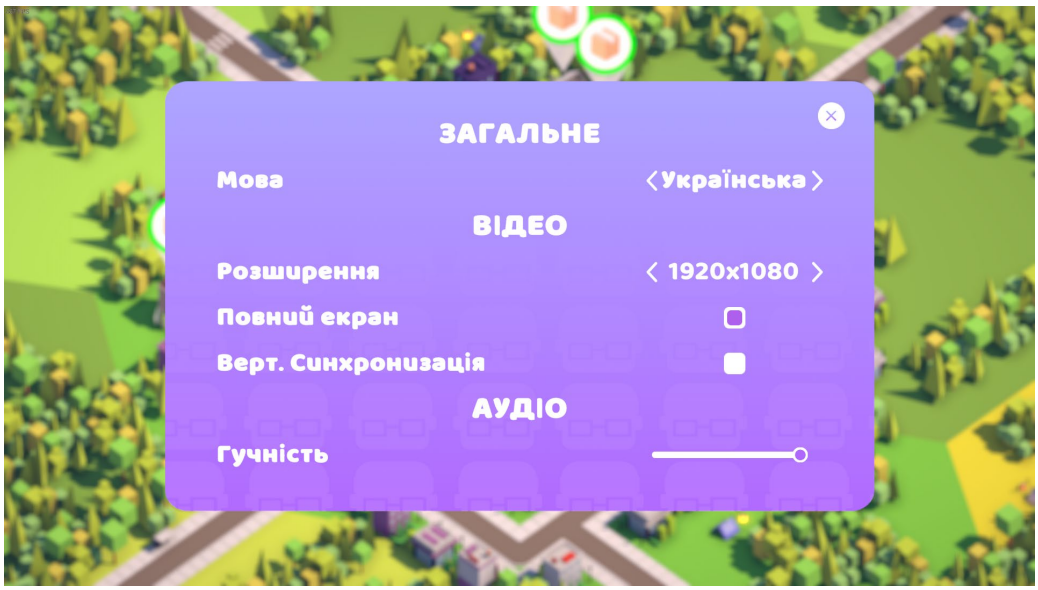

Рис 2.6 Вікно налаштувань гри

Гравець може налаштувати мову програми. Серед доступних мов: українська, англійська, німецька, французька та іспанська.

В графічній секції налаштувань доступне налаштування роздільної здатності, повноекранного режиму а також вертикальної синхронізації.

Також гравець може налаштувати гучність аудіо за допомогою спеціального повзунка.

При натисканні кнопки «Продовжити» у головному меню гри гравець продовжує гру з того місця, де він зупинився.

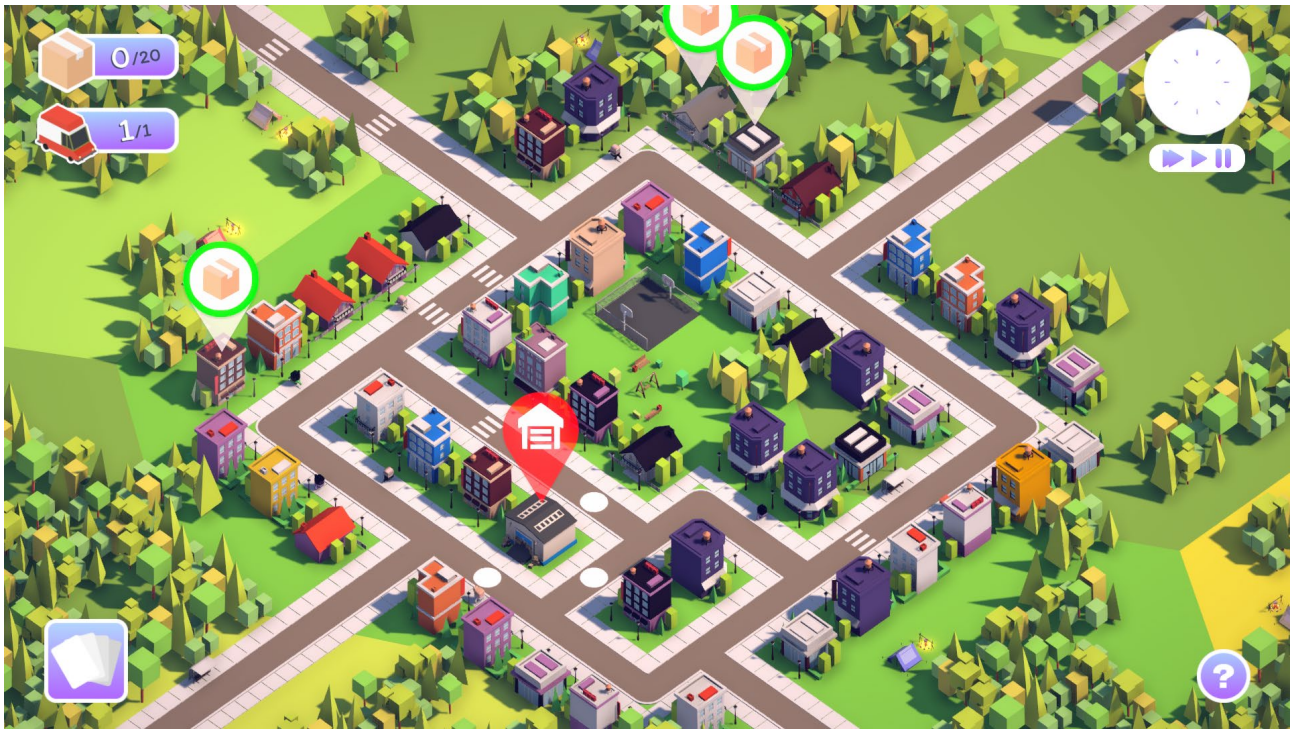

Рис 2.7 Інтерфейс у грі

У грі інтерфейс складається з показників кількості доступних автомобілів доставки, кількості доставлених пакунків, годинника, для контролю часу, кнопки підсилювачів та кнопки підказки, яка відкриває вікно з описом усіх ігрових механік.

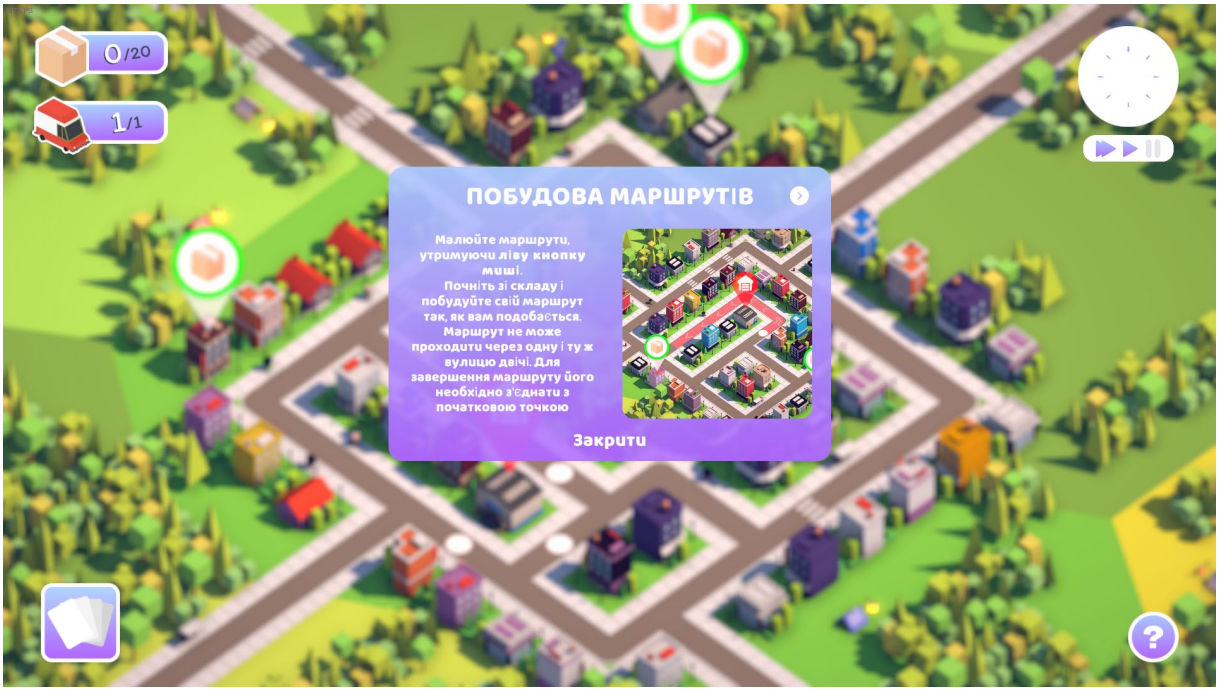

Рис 2.8 Вікно підказок

У міру просування гравця, йому пропонуватимуть підсилювачі. Кожен підсилювач має свій унікальний ефект. Гравець може сам обрати, чи використати підсилювач одразу, чи зберегти.

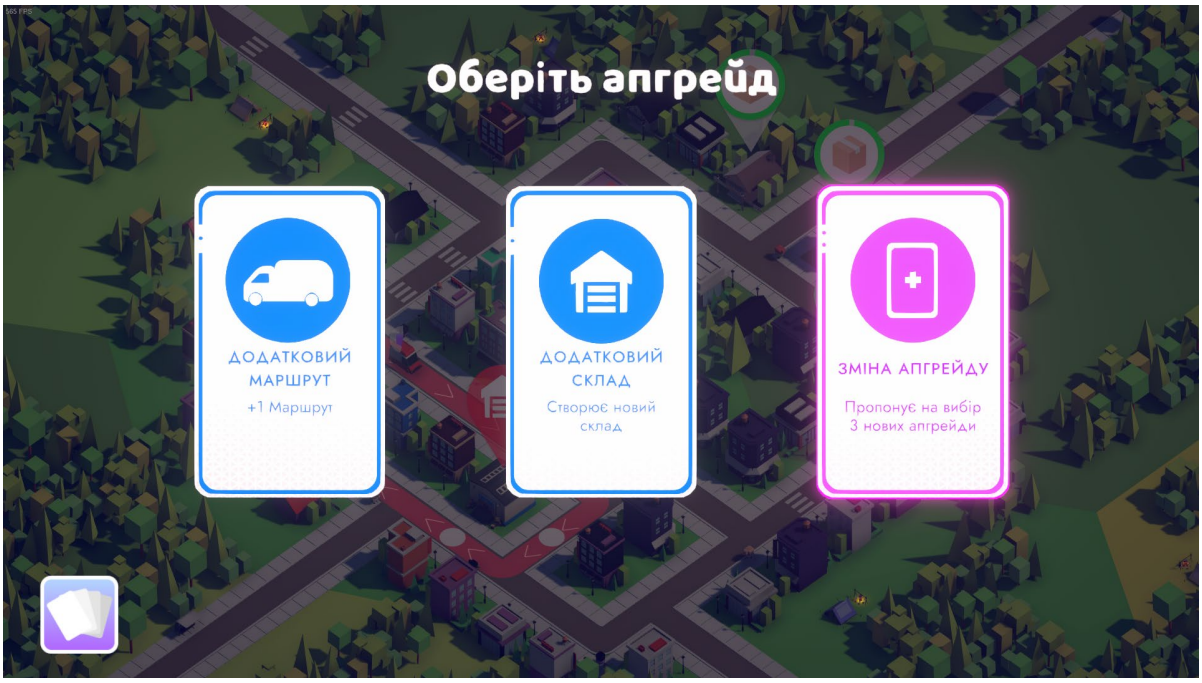

Рис 2.9 Вікно вибору підсилювачів

Загалом у грі 12 різних підсилювачів з різною силою ефекту. Сила ефекту позначається кольором.

Основним завданням гравця є побудова маршрутів поштової доставки, щоб доставити вчасно усі пакунки. Поштові маршрути та автомобілі для зручності розфарбовані в різні кольори, усього можливо мати 10 маршрутів одночасно. Індекатором часу, що залишився на доставку слугує маркер над відповідними домівками. Якщо гравцеві не вдасться доставити пакунок вчасно, гру буде завершено.

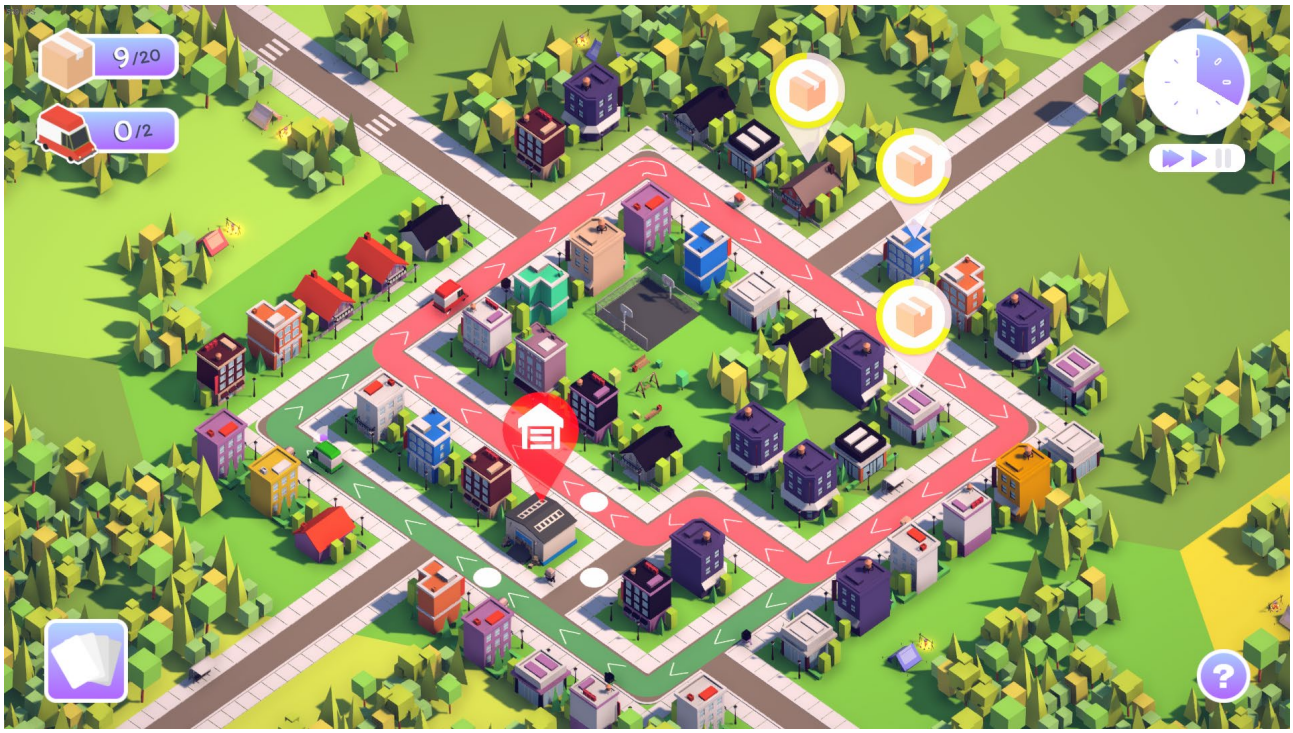

Рис 2.10 Побудова маршрутів

Коли гравець доставить відповідно кількість пакунків, його буде переміщено до нового, більш великого міста. Чим більше місто, тим складнішою стає гра. Найменше місто має розмір 6*6 ігрових клітинок, а найбільше 20*20 ігрових клітинок. Чим місто більше, тим більше в ньому можна побудувати маршрутів доставки.

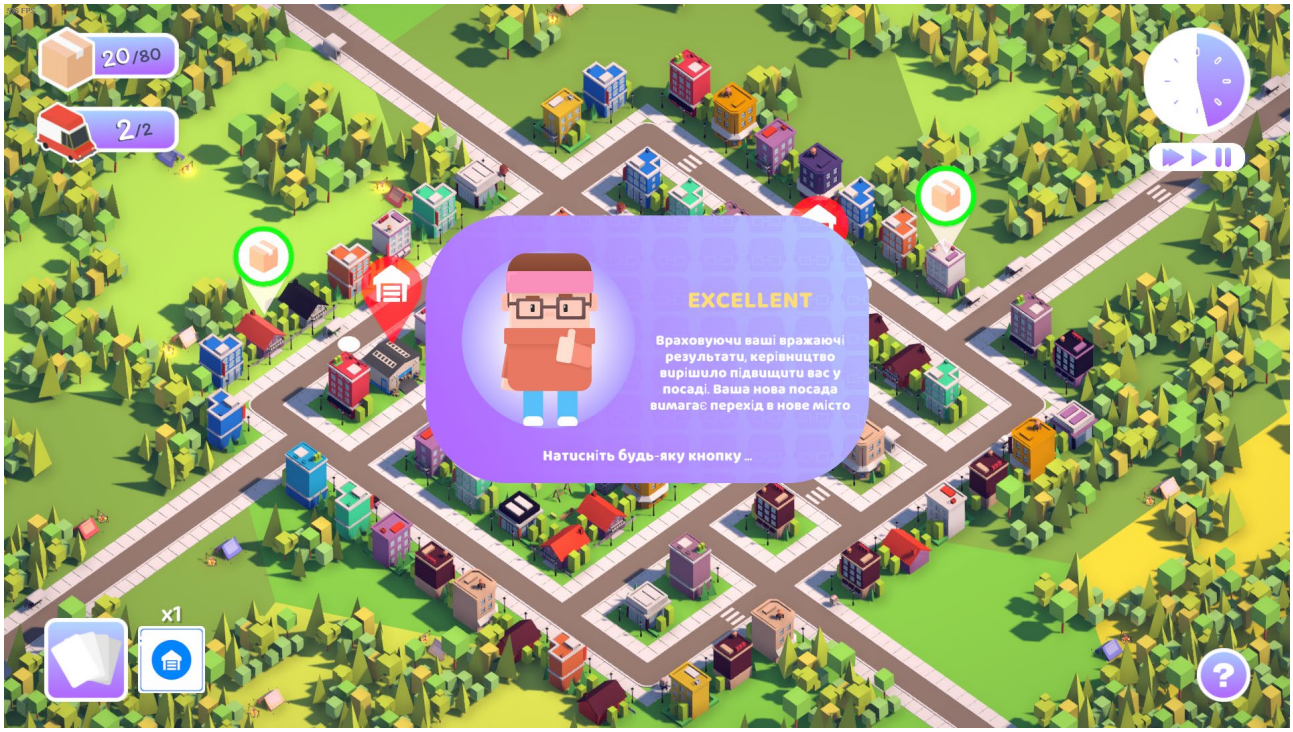

Рис 2.11 Вікно переходу до нового міста

Рис 2.12 Найбільше місто у грі

Якщо гравець не впорається з доставкою і не доставить посилку вчасно, йому буде показано вікно кінця гри, а також запропоновано почати спочатку, або повернутися до головного меню.

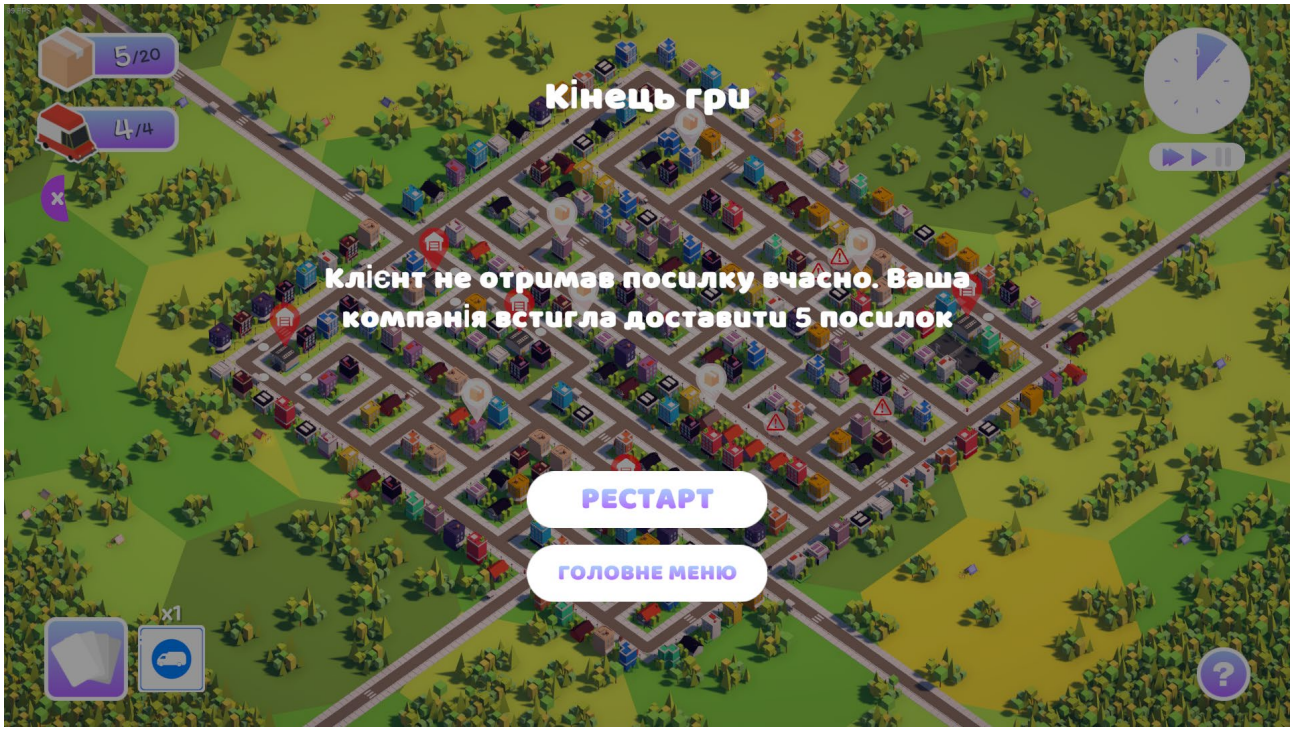

Рис 2.13 Вікно кінця гри

РОЗДІЛ 3.
ЕКОНОМІЧНИЙ РОЗДІЛ

Початкові дані:

1. передбачуване число операторів програми – 2500;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,1;
4. годинна заробітна плата програміста– 115 грн/год; (за версією сайта WorkUA) - https://www.work.ua/salary-dnipro-it/;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,3;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,4;
7. вартість машино-години ЕОМ –16 грн/год(2 грн – е/е, 10 грн – ПЗ, 4грн - амортизація).

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C(1 + p), \text{ де} \qquad (3.2)$$

Q– передбачуване число операторів;

C– коефіцієнт складності програми;

p– коефіцієнт кореляції програми в ході її розробки.

$$Q=2500 \cdot 1,3 \cdot (1+0,1)=3575;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75 \ldots 85)K}, \text{ людино-годин,} \qquad (3.3)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі(1,3);

K – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності(1,4);

$$t_u = \frac{3575 \cdot 1,3}{85 \cdot 1,4} = 39,05, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)K} \quad (3.4)$$

$t_a = \frac{3575}{20 \cdot 1,4} = 127,67$, людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad (3.5)$$

$t_n = \frac{3575}{25 \cdot 1,4} = 102,14$людино-годин.

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)K} \quad (3.6)$$

$t_{отл} = \frac{3575}{5 \cdot 1,4} = 510,71$, людино-годин,

- за умови комплексного налагодження завдання:

$$t^к_{отл} = 1,2 \cdot t_{отл} \quad (3.7)$$

$t^к_{отл} = 1,2 \cdot 707,14 = 612,85$, людино-годин

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до} \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15...20)K} \quad (3.9)$$

$$t_{др}=\frac{3575}{20\cdot 1,4}=127,67, \text{ людино-годин.}$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 \cdot t_{др} \quad (3.10)$$

$$t_{до}=0,75\cdot 127,67=95,75, \text{ людино-годин.}$$

$$t_{д}=127,67+95,75=223,42, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 39,05 + 127,67 + 102,14 + 510,71 + 223,42 = 1052,99, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1052,99 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{по}$ включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година

$$З_{зп}=1052,99\cdot 115=121093,85 \text{ грн.}$$

$З_{МВ}$– Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot СМ, \text{ грн}, \quad (3.13)$$

де *$t_{отл}$* – трудомісткість налагодження програми на ЕОМ, год.

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ}=612,85 \cdot 16=9805,6, \text{ грн.}$$

$$К_{ПО}=121093,2+9805,6=130898,8, \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \quad (3.14)$$

де B_k - число виконавців;

$F_р$ - місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$Т=\frac{1052,99}{1 \cdot 176}=5,98 \text{ міс.}$$

Висновки. Час розробки даного програмного забезпечення складає 1052,99 людино-годин. Таким чином, очікувана тривалість розробки складе 5,98 місяця при 40 годинному робочому тижні (місячний фонд робочого часу 176 годин), а витрати на створення програмного забезпечення складатимуть 130898,8 грн.

ВИСНОВКИ

Ця кваліфікаційна робота заглиблюється в ігрову індустрію, досліджуючи різні її явища та проливаючи світло на її майбутні перспективи. Завдяки ретельному аналізу зібраних даних стає зрозуміло, що ігри розвиватимуться паралельно з розвитком інформаційних технологій. Майбутній розвиток ігрових рушіїв відкриє безпрецедентні можливості для розробників ігор. Ця нова межа уможливить створення захоплюючих ігрових вражень, пропонуючи підвищену реалістичність та інтерактивність завдяки високоякісній графіці та реалістичним ефектам.

В рамках цієї кваліфікаційної роботи було ретельно досліджено аналогічні на ринку продукти, та створено новий, який має більшу функціональність порівняно з аналогами.

По-перше, основну мету гри було змінено на доставку пошти, що дає гравцям виклик, де вони повинні стратегічно проектувати та оптимізувати свої дорожні мережі, щоб ефективно транспортувати пошту в межах встановленого часу. Особливу увагу було приділено складному завданню генерації ігрової мапи на основі сіткової системи та пошуку оптимального шляху за допомогою алгоритму A*.

Крім того, було змінено візуальні та звукові аспекти гри, щоб створити більш реалістичну атмосферу. Графіка була покращена, щоб забезпечити візуально привабливе середовище з яскравими кольорами, та деталізованими містами та поштовими автівками. Звукові ефекти стали більш реалістичними, а фонова музика додає глибини та занурення, покращуючи загальний ігровий досвід та утримуючи гравців у грі.

Також одне з важливих покращень, яке було впроваджено, - це введення системи прогресії. На відміну від Mini Motorways, де рівень складності залишається відносно постійним, наша гра включає різні рівні зі зростаючою складністю. Це гарантує, що гравці постійно мотивовані та отримують виклики

в міру проходження гри. Крім того, було запроваджено різноманітні підсилювачі та бонуси, які дозволяють гравцям долати перешкоди та досягати вищих результатів. Цей аспект гри додає елемент стратегії, забезпечуючи відчуття прогресу та досягнень.

Отже, Unity 3D виявився надійним та економічно ефективним середовищем розробки, що надає повний набір інструментів і функцій, необхідних для успішної реалізації проектів. Використовуючи внутрішні можливості рушія Unity, було оптимізовано реалізацію різних процесів, що призвело до ефективної та безперебійної розробки гри.

Таким чином, середовище розробки Unity є найкращим вибором для створення ігрових продуктів професійного рівня. Завдяки своїм потужним функціям та можливостям Unity дає розробникам змогу створювати передові ігри, які можуть ефективно конкурувати на ринку, задовольняючи постійно зростаючі вимоги та очікування користувачів. Оскільки ігрова індустрія продовжує процвітати, Unity залишається на передовій, забезпечуючи інноваційний та захоплюючий досвід для геймерів по всьому світу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Unreal Engine Manual [Електронний ресурс] – Режим доступу: https://www.unrealengine.com/en-US

2. CryEngine V Manual [Електронний ресурс] – Режим доступу: https://docs.cryengine.com/display/CEMANUAL/CRYENGINE+V+Manual

3. Крейтон, Р.Х. Основи розробки ігор у Unity / Р.Х. Крейтон. – Packt Publishing, 2010. – 83 с.

4. Dinosaur Polo Club Website [Електронний ресурс] – Режим доступу: https://dinopoloclub.com/games/

5. Transport Fever Website [Електронний ресурс] – Режим доступу: https://www.transportfever2.com/

6. Марк Дж. П. Вольф (15 червня 2012). Перед катастрофою: Рання історія відеоігор. Видавництво Університету штату Вейн. р. 173. ISBN 978-0814337226.

7. Стюарт Рассел, Пітер Норвіг: Штучний інтелект: Сучасний підхід, 2004, Prentice Hall, ISBN 3-8273-7089-2

8. Девід А. Гроссман, Офір Фрідер, Пошук інформації: Алгоритми та евристика, 2-е видання, 2004, ISBN 1402030045

9. Мартін, Роберт (2018). Чиста архітектура: Посібник для майстрів зі структури та дизайну програмного забезпечення. с. 58. ISBN 9780134494166.

10. Фаулер, Мартін (2002). Шаблони архітектури корпоративних додатків. Addison-Wesley. ISBN 978-0-321-12742-6.

11. Арчер, Том (2001). Inside C#. Редмонд, штат Вашингтон: Microsoft Press. ISBN 0-7356-1288-9.

12. Методичні рекомендації до виконання кваліфікаційних робіт здобувачів першого рівня вищої освіти спеціальності 121 Інженерія програмного забезпечення / В.В. Спірінцев, І.М. Удовик, О.С. Шевцова; Д : НТУ «Дніпровська політехніка», 2022. – 60 с.

13. Марк Дж. П. Вольф. Вибух відеоігор: історія від PONG до Playstation і далі. ABC-CLIO. с. 73. ISBN 978-0-313-33868-7.

14. Звіт про аналіз розміру, частки та тенденцій ринку відеоігор [Електронний ресурс] – Режим доступу:

https://www.grandviewresearch.com/industry-analysis/video-game-market

15. Чакон, Скотт (24 грудня 2014). Pro Git (2-е вид.). Нью-Йорк, штат Нью-Йорк: Apress. с. 29-30. ISBN 978-1-4842-0077-3.

16. JetBrains Rider електронна документація [Електронний ресурс] – Режим доступу: https://blog.jetbrains.com/dotnet/2017/08/03/rider-2017-1-jetbrains-net-ide-hits-rtm/

17. Бертран Мейєр (2009). Дотик класу: Як навчитися добре програмувати з об'єктами та контрактами. Springer Science & Business Media. с. 329. ISBN 978-3-540-92144-8.

18. Стів Л. Кент (2001), Повна історія відеоігор: від понгу до покемонів і далі: історія захоплення, яке вплинуло на наше життя і змінило світ, с. 64, Прима, ISBN 0-7615-3643-4

19. Лав'єрі, Едвард (2018). Початок роботи з Unity 2018: Посібник для початківців з розробки 2D та 3D ігор за допомогою Unity, 3-тє видання. Packt Publishing. с. 20. ISBN 9781788832915.

20. Зубек, Роберт (18 серпня 2020). Елементи ігрового дизайну. Видавництво Массачусетського технологічного інституту. ISBN 9780262043915.

Додаток А

ЛІСТИНГ ПРОГРАМИ

Файл GameManager.cs

```
using System;
using System.Threading.Tasks;
using UnityEngine;

namespace GeneralSystem
{
    public class GameManager : MonoBehaviour
    {
        public static event Action OnMovedToNewCity;
        public static event Action OnGameReset;

        private static bool _isResetting;

        private void Awake() => ScoreManager.OnScoreForNewCityReached += MoveToNewCity;

        public static async void RestartGame()
        {
            if(_isResetting) return;
            _isResetting = true;

            await Task.Yield();
            OnGameReset?.Invoke();
            _isResetting = false;
        }

        private static async void MoveToNewCity()
        {
            await Task.Yield();
            OnMovedToNewCity?.Invoke();
        }

        private void OnDestroy() => ScoreManager.OnScoreForNewCityReached -=
MoveToNewCity;
```

```
    }
}
```

Файл CarsManager.cs

```
using System.Collections.Generic;
using UnityEngine;
using System;
using GeneralSystem;
using Routes;
using Saving;

namespace Cars
{
    public class CarsManager : MonoBehaviour, ISavable
    {
        public static CarsManager Instance;

        public static event Action OnCarCreated;
        public static event Action OnCarRemoved;

        [SerializeField] private GameObject carPrefab;
        private readonly Dictionary<int, Car> _cars = new Dictionary<int, Car>();
        private readonly List<int> _keysBuffer = new List<int>();

        public Type SaveDataType => typeof(CarsSave);

        private void Awake()
        {
            Instance = this;

            GameManager.OnGameReset += ResetCarsController;
            GameManager.OnMovedToNewCity += ResetCarsController;
            RoutesBuilder.OnRouteCreated += CreateCar;
        }

        private void ResetCarsController()
        {
            foreach(int routeID in _cars.Keys)
            {
                GameObjectPool.Instance.ReturnGameObject(_cars[routeID].gameObject);
                OnCarRemoved?.Invoke();
            }

            _cars.Clear();
            _keysBuffer.Clear();
        }

        public void Load(object data)
        {
            CarsSave save = (CarsSave)data;
            for(int i = 0; i < save.routesIndex.Count; i++)
            {
                int routeIndex = save.routesIndex[i];
                _cars[routeIndex].LoadCar(save.routePointIndexes[i],  save.stopTimes[i],
save.keepRoutes[i], save.carsPositions[i]);
            }
        }
```

```
        private void Update()
        {
            foreach(int routeID in _cars.Keys)
                _cars[routeID].MoveCar();

            for (int i = 0; i < _keysBuffer.Count; i++)
            {
                int routeID = _keysBuffer[i];
                if (TryRemoveCar(routeID)) i--;
            }
        }

        private void CreateCar(Route route)
        {
            if(_cars.ContainsKey(route.RouteID)) return;
            if(route.Length == 0) return;

            Car carToAdd = GameObjectPool.Instance.GetObject(carPrefab, route.Coords[0],
Vector3.zero).GetComponent<Car>();
            _cars.Add(route.RouteID, carToAdd);
            _keysBuffer.Add(route.RouteID);
            carToAdd.InitializeCar(route);
            OnCarCreated?.Invoke();
        }

        private bool TryRemoveCar(int routeID)
        {
            if(_cars[routeID].finishedRoute == false) return false;
            GameObjectPool.Instance.ReturnGameObject(_cars[routeID].gameObject);
            _cars.Remove(routeID);
            _keysBuffer.Remove(routeID);
            RoutesManager.Instance.RemoveRoute(routeID);
            OnCarRemoved?.Invoke();

            return true;
        }

        public void SetCarsSpeed(float speed)
        {
            foreach (int routeID in _cars.Keys)
                _cars[routeID].SetCarsSpeed(speed);
        }

        public object Save()
        {
            CarsSave save = new CarsSave();

            foreach(int routeID in _cars.Keys)
            {
                Car carToSave = _cars[routeID];
                save.routesIndex.Add(routeID);
                save.routePointIndexes.Add(carToSave.GetRoutePointIndex());
                save.stopTimes.Add(carToSave.GetStopTime());
                save.keepRoutes.Add(carToSave.GetKeepRoute());
                save.carsPositions.Add(carToSave.GetCarPosition());
            }

            return save;
        }
```

```
        private void OnDestroy()
        {
            GameManager.OnGameReset -= ResetCarsController;
            GameManager.OnMovedToNewCity -= ResetCarsController;
            RoutesBuilder.OnRouteCreated -= CreateCar;
        }
    }
}
```

Файл Car.cs

```
using System;
using System.Collections;
using Audio;
using UnityEngine;
using Orders;
using Routes;
using UtilitySystems;

namespace Cars
{
    public class Car : MonoBehaviour
    {
        public static event Action OnCarFinishedRoute;
        public static event Action OnKeepRoutePressed;
        public static event Action OnCarCrashed;
        private Transform _cachedTransform;
        [SerializeField] private CarVisuals carVisuals;

        private Route _route;
        private int _routePointIndex;

        private int RoutePointIndex
        {
            get => _routePointIndex;
            set
            {
                _routePointIndex = value;
                _route.CarPointIndex = _routePointIndex;
            }
```

```
}

public const float BASE_SPEED = 10f;
private float _currentSpeed = 10f;

public const float DELIVERY_TIME = 2.5f;
private float _stopTime;

private const float ROTATION_SMOOTHNESS = 5f;
private Vector3 _lookDirection = Vector3.zero;
private Quaternion _toRotation = Quaternion.identity;

[HideInInspector] public bool finishedRoute;

private const float FADE_INTERVAL = 0.25f;
private const int CRASH_TICKS_COUNT = 20;
public const float CRASH_TIME = FADE_INTERVAL * CRASH_TICKS_COUNT;

private bool _stopForRepair;
private bool _keepRoute;

private bool KeepRoute
{
    get => _keepRoute;
    set
    {
        _keepRoute = value;
        _route.KeepRoute = _keepRoute;
    }
}

private void Awake() => _cachedTransform = transform;

public void InitializeCar(Route route)
{
    _route = route;
```

```
            finishedRoute = false;
            _stopForRepair = false;
            RoutePointIndex = 0;
            _stopTime = 0f;
            _currentSpeed = BASE_SPEED;
            KeepRoute = false;

            _cachedTransform.LookAt(_route.Coords[1]);
            _lookDirection = new Vector3(_route.Coords[1].x, 0, _route.Coords[1].z) -
_cachedTransform.position;

            carVisuals.ResetCarVisuals(_route.RouteID);
        }

        public void LoadCar(int routePointIndex, float stopTime, bool keepRoute,
Vector3 position)
        {
            RoutePointIndex = routePointIndex;
            transform.position = position;
            _stopTime = stopTime;

            int nextPoint = RoutePointIndex == _route.Length - 1 ? 0 : RoutePointIndex
+ 1;
            _cachedTransform.LookAt(_route.Coords[nextPoint]);
            _lookDirection = new Vector3(_route.Coords[nextPoint].x, 0,
_route.Coords[nextPoint].z) - _cachedTransform.position;

            if(_stopTime > 0) carVisuals.TurnOnPopUp(CarPopUpType.Delivery, stopTime);
            KeepRoute = keepRoute;
            if(keepRoute) carVisuals.TurnOnPopUp(CarPopUpType.KeepRoute, 0);
        }

        public void MoveCar()
        {
            if(_route.Length == 0) return;
            if(_route.IsRouteLooped() == false) return;

            if(_stopTime > 0)
```

```
            {
                _stopTime -= Time.deltaTime;
                if (_stopTime <= 0)
                    carVisuals.TurnOffPopUp(CarPopUpType.Delivery);
                return;
            }

            if (_stopForRepair) return;

            _toRotation = Quaternion.LookRotation(_lookDirection);
            _cachedTransform.rotation = Quaternion.Lerp(_cachedTransform.rotation,
_toRotation, ROTATION_SMOOTHNESS * Time.deltaTime);

            _cachedTransform.position = Vector3.MoveTowards(transform.position,
_route.Coords[RoutePointIndex], _currentSpeed * Time.deltaTime);

            if (Vector3.Distance(_cachedTransform.position,
_route.Coords[RoutePointIndex]) != 0) return;
            if(TryDeliverOrder()) return;

            if(RoutePointIndex < _route.Length - 1)
            {
                RoutePointIndex ++;
                _lookDirection = new Vector3( _route.Coords[RoutePointIndex].x, 0,
_route.Coords[RoutePointIndex].z) - _cachedTransform.position;

                return;
            }

            OnCarFinishedRoute?.Invoke();

            if (KeepRoute)
            {
                RoutePointIndex = 0;
                _lookDirection = new Vector3( _route.Coords[1].x, 0,
_route.Coords[1].z) - _cachedTransform.position;
                return;
            }
```

```
            if(_stopTime <= 0)
                finishedRoute = true;
        }

        private bool TryDeliverOrder()
        {
            if
(OrdersManager.Instance.TryCompleteOrder(Utility.GetGridCoord(_route.Coords[RoutePointI
ndex])))
            {
                _stopTime = DELIVERY_TIME;
                carVisuals.TurnOnPopUp(CarPopUpType.Delivery, DELIVERY_TIME);
                return true;
            }

            return false;
        }

        public void OnCarSelected() =>
RoutesManager.Instance.SelectRoute(_route.RouteID, true);

        public void OnRouteKeepPressed()
        {
            OnKeepRoutePressed?.Invoke();
            KeepRoute = !KeepRoute;

            if(KeepRoute) carVisuals.TurnOnPopUp(CarPopUpType.KeepRoute, 0);
            else carVisuals.TurnOffPopUp(CarPopUpType.KeepRoute);
        }

        public void SetCarsSpeed(float speed)
        {
            carVisuals.SetParticlesActiveState(speed > BASE_SPEED);
            _currentSpeed = speed;
        }
```

```
        public int GetRoutePointIndex() => RoutePointIndex;
        public float GetStopTime() => _stopTime;
        public bool GetKeepRoute() => KeepRoute;
        public Vector3 GetCarPosition() => _cachedTransform.position;

        private void OnTriggerEnter(Collider other) => StartCoroutine(CarCrash());

        private IEnumerator CarCrash()
        {
            OnCarCrashed?.Invoke();
            AudioManager.Instance.PlaySound(AudioClipType.CarCrash);
            carVisuals.TurnOnPopUp(CarPopUpType.Repair, CRASH_TIME);
            _stopForRepair = true;
            for(int i = 0; i < CRASH_TICKS_COUNT; i++)
            {
                carVisuals.ApplyCarMaterial(i % 2 == 0);
                yield return new WaitForSeconds(FADE_INTERVAL);
            }

            _stopForRepair = false;
            carVisuals.TurnOffPopUp(CarPopUpType.Repair);
        }
    }
}
```

Файл CarVisuals.cs

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

namespace Cars
{
    public class CarVisuals : MonoBehaviour
    {
```

```
        [SerializeField] private List<Material> normal = new List<Material>();
        [SerializeField] private List<Material> transparent = new List<Material>();
        [SerializeField] private MeshRenderer meshRenderer;

        [SerializeField] private GameObject particles;
        [SerializeField] private SpriteRenderer carCrashPopUp;
        [SerializeField] private SpriteRenderer carDeliveryPopUp;
        [SerializeField] private SpriteRenderer keepRoutePopUp;

        private const float POP_UP_Y_POS = 4.75f;
        private const float POP_UP_OFFSET_X = 1.5f;

        private readonly List<SpriteRenderer> _activeEffects = new
List<SpriteRenderer>();
        private readonly List<CarPopUp> _effectsTime = new List<CarPopUp>();

        private static readonly int Arc1 = Shader.PropertyToID("_Arc1");

        private int _routeID;

        public void ResetCarVisuals(int routeID)
        {
            particles.SetActive(false);
            carCrashPopUp.enabled = false;
            carDeliveryPopUp.enabled = false;
            keepRoutePopUp.enabled = false;

            _routeID = routeID;
            meshRenderer.material = normal[routeID];

            _effectsTime.Clear();
            _activeEffects.Clear();
        }

        private void FixedUpdate()
        {
            for (int i = 0; i < _effectsTime.Count; i++)
```

```
            {
                if(_effectsTime.Count == 0) return;


                _effectsTime[i] = new CarPopUp(_effectsTime[i].Type,
_effectsTime[i].Time - Time.deltaTime);


                if (!(_effectsTime[i].Time <= 0)) continue;
                _effectsTime.Remove(_effectsTime[i]);
                i--;
            }


            if(carCrashPopUp.enabled) carCrashPopUp.material.SetFloat(Arc1, 360 * (1 -
_effectsTime.Find(popUp  => popUp.Type == CarPopUpType.Repair).Time / Car.CRASH_TIME));
            if(carDeliveryPopUp.enabled) carDeliveryPopUp.material.SetFloat(Arc1, 360 *
(1 - _effectsTime.Find(popUp  => popUp.Type == CarPopUpType.Delivery).Time /
Car.DELIVERY_TIME));


            keepRoutePopUp.transform.rotation = Quaternion.Euler(Utility.CameraAngle);
        }


        public void SetParticlesActiveState(bool isActive) =>
particles.SetActive(isActive);


        public void TurnOnPopUp(CarPopUpType type, float time)
        {
            SpriteRenderer rend = type switch
            {
                CarPopUpType.Delivery => carDeliveryPopUp,
                CarPopUpType.Repair => carCrashPopUp,
                _ => keepRoutePopUp
            };
            rend.enabled = true;
            rend.transform.rotation = Quaternion.Euler(Utility.CameraAngle);
            _activeEffects.Add(rend);
            if(time != 0)
                _effectsTime.Add(new CarPopUp(type, time));


            SetEffectsPositions();
```

```
        }

        public void TurnOffPopUp(CarPopUpType type)
        {
            SpriteRenderer rend = type switch
            {
                CarPopUpType.Delivery => carDeliveryPopUp,
                CarPopUpType.Repair => carCrashPopUp,
                _ => keepRoutePopUp
            };
            rend.enabled = false;

            if (_activeEffects.Contains(rend))
                _activeEffects.Remove(rend);

            SetEffectsPositions();
        }

        private void SetEffectsPositions()
        {
            if (_activeEffects.Count == 1) _activeEffects[0].transform.localPosition =
new Vector3(0, POP_UP_Y_POS, 0);
            else if (_activeEffects.Count == 2)
            {
                _activeEffects[0].transform.localPosition = new Vector3(-
POP_UP_OFFSET_X, POP_UP_Y_POS, 0);
                _activeEffects[1].transform.localPosition = new
Vector3(POP_UP_OFFSET_X, POP_UP_Y_POS, 0);
            }
            else if (_activeEffects.Count == 3)
            {
                _activeEffects[0].transform.localPosition = new Vector3(-
POP_UP_OFFSET_X * 2, POP_UP_Y_POS, 0);
                _activeEffects[1].transform.localPosition = new Vector3(0,
POP_UP_Y_POS, 0);
                _activeEffects[2].transform.localPosition = new Vector3(POP_UP_OFFSET_X
* 2, POP_UP_Y_POS, 0);
            }
        }
```

```
        public void ApplyCarMaterial(bool isTransparent) => meshRenderer.material =
isTransparent ? transparent[_routeID] : normal[_routeID];
    }

    public struct CarPopUp
    {
        public readonly CarPopUpType Type;
        public readonly float Time;

        public CarPopUp(CarPopUpType type, float time)
        {
            Type = type;
            Time = time;
        }
    }

    public enum CarPopUpType
    {
        Delivery,
        Repair,
        KeepRoute,
    }
}
```

Файл BuildingsGeneration.cs

```
using System.Collections.Generic;
using Unity.Mathematics;
using UnityEngine;
using System;
using Cards.CardsActions;
using DG.Tweening;
using GeneralSystem;
using UtilitySystems;
```

```
namespace Generation
{
    public class BuildingsGeneration : MonoBehaviour
    {
        private MapData _mapData;

        private const int WAREHOUSES_MAX_AMOUNT = 50;

        public static event Action<Vector3> OnWarehouseCreated;
        public static event Action<Vector3> OnWarehouseRemoved;

        private static readonly List<Vector3> WarehousesCoords = new List<Vector3>();

        private const float TWEEN_SCALE = 240f;
        private const float TWEEN_DURATION = 0.5f;

        public static bool CanPlaceWarehouse() => WarehousesCoords.Count <
WAREHOUSES_MAX_AMOUNT;
        public static bool CanUseWarehouseExplosion() => WarehousesCoords.Count > 1;

        private void Awake()
        {
            OnWarehouseCreated += AddWarehouseCoords;
            AdditionalWarehouse.OnAdditionalWarehouseUsed += PlaceWarehouse;
            WarehouseExplosion.OnWarehouseExplosionUsed += RemoveWarehouse;
        }

        public void Init(MapData mapData) => _mapData = mapData;

        public void GenerateBuildings()
        {
            WarehousesCoords.Clear();
            for(int i = 0; i < _mapData.cityInfo.warehousesStartingAmount; i++)
                PlaceWarehouse();

            PlaceHouses();
        }
```

```
        private List<int2> _availablePositions = new List<int2>();
        private void PlaceWarehouse()
        {
            if(WarehousesCoords.Count >= WAREHOUSES_MAX_AMOUNT) return;

            _availablePositions.Clear();
            _availablePositions = _mapData.GetPositionsForWarehouse();

            if(_availablePositions.Count == 0) return;

            int2 randomPos = GetWarehousePosition();
            SpawnBuilding(randomPos, TileType.Warehouse, CityObjectType.Warehouse);

            foreach(int2 direction in Utility.Directions)
                if(_mapData.GetTileType(randomPos + direction) == TileType.Road)
_mapData.SetCanStartRoute(randomPos + direction, true);

            OnWarehouseCreated?.Invoke(Utility.GetWorldCoord(randomPos));
        }

        private void RemoveWarehouse()
        {
            if(WarehousesCoords.Count <= 1) return;

            Vector3 warehousePosition = WarehousesCoords.RandomElement();
            int2 gridPos = Utility.GetGridCoord(warehousePosition);
            OnWarehouseRemoved?.Invoke(warehousePosition);

            _mapData.warehouseCoords.Remove(gridPos);
            WarehousesCoords.Remove(warehousePosition);

            foreach (int2 direction in Utility.Directions)
                if(_mapData.Map.ContainsKey(gridPos+direction) &&
_mapData.GetTileType(gridPos + direction) == TileType.Road)
                    _mapData.Map[gridPos + direction].CanStartRoute = false;
```

```
            _mapData.Map[gridPos].SpawnedObject.transform.DOScale(TWEEN_SCALE,
TWEEN_DURATION).SetUpdate(true).OnComplete(() =>
            {
                _mapData.RemoveObject(gridPos);
                PlaceHouses();
            });
        }

        public static void PlaceWarehouseFromSave(int2 position) =>
OnWarehouseCreated?.Invoke(Utility.GetWorldCoord(position));

        private void PlaceHouses()
        {
            _availablePositions.Clear();
            _availablePositions = _mapData.GetPositionsForHouses();

            int posCount = _availablePositions.Count;
            for(int i = 0; i < posCount; i++)
                SpawnBuilding(_availablePositions[i], TileType.House,
CityObjectType.House);
        }

        private void SpawnBuilding(int2 pos, TileType tileType, CityObjectType type)
        {
            _mapData.RemoveObject(pos);

            int yRotation = GetBuildingRotation(pos);
            int representationID = Storage.GetRandomRepresentationID(type);
            GameObject spawnedObject =
GameObjectPool.Instance.GetObject(Storage.GetRepresentationByID(type,
representationID),
                                        Utility.GetWorldCoord(pos), new Vector3(0,
yRotation, 0));
            spawnedObject.transform.DOKill();
            spawnedObject.transform.localScale = new Vector3(200f, 200f, 200f);
            _mapData.Map[pos] = new Tile(tileType, type, representationID,
spawnedObject);

            if(tileType == TileType.House) _mapData.housesCoords.Add(pos);
```

```
            else _mapData.warehouseCoords.Add(pos);
        }

        private int GetBuildingRotation(int2 pos)
        {
            int rotation = 0;
            if (_mapData.GetTileType(pos + new int2(-1,0)) == TileType.Road) return 90;
            else if(_mapData.GetTileType(pos + new int2(1,0)) == TileType.Road) return
-90;
            else if (_mapData.GetTileType(pos + new int2(0,-1)) == TileType.Road)
return 0;
            else if (_mapData.GetTileType(pos + new int2(0,1)) == TileType.Road) return
180;

            return rotation;
        }

        private int2 GetWarehousePosition()
        {
            for(int i = 0; i < 20; i++)
            {
                int2 randomPos = _availablePositions.RandomElement();
                bool posAvailable = true;

                foreach(Vector3 pos in WarehousesCoords)
                    if(Vector3.Distance(pos, Utility.GetWorldCoord(randomPos)) < 32)
posAvailable = false;

                if(posAvailable) return randomPos;
            }
            return _availablePositions.RandomElement();
        }

        private void AddWarehouseCoords(Vector3 pos) => WarehousesCoords.Add(pos);

        private void OnDestroy()
        {
            OnWarehouseCreated -= AddWarehouseCoords;
```

```
            AdditionalWarehouse.OnAdditionalWarehouseUsed -= PlaceWarehouse;
        }
    }
}
```

Файл CityGeneration.cs

```
using UnityEngine;
using System;
using Audio;
using GeneralSystem;
using Saving;

namespace Generation
{
    public class CityGeneration : MonoBehaviour, ISavable
    {
        [SerializeField] private RoadsGeneration roadsGeneration;
        [SerializeField] private BuildingsGeneration buildingsGeneration;
        [SerializeField] private EnvironmentGeneration environmentGeneration;
        public MapData mapData;

        public static event Action<MapData> OnCityGenerated;

        private CityType _currentCityType = CityType.City8X8;
        private CityType CurrentCityType
        {
            get=> _currentCityType;
            set
            {
                _currentCityType = value;
                CurrentCity = value;
                mapData.cityInfo = Storage.GetCityData(_currentCityType);
            }
```

```
}

public static CityType CurrentCity;

public Type SaveDataType => typeof(MapSave);

private void Awake()
{
    roadsGeneration.Init(mapData);
    buildingsGeneration.Init(mapData);
    environmentGeneration.Init(mapData);
}

private void Start()
{
    CurrentCityType = CityType.City8X8;

    GameManager.OnGameReset += ResetCityData;
    GameManager.OnGameReset += GenerateCity;
    GameManager.OnMovedToNewCity += MoveToNewCity;
    SaveManager.OnGameLoaded += GenerateCityIfSaveEmpty;
}

public void Load(object data)
{
    MapSave save = (MapSave)data;
    mapData.LoadMapData(save);

    CurrentCityType = save.cityType;
    OnCityGenerated?.Invoke(mapData);
}

private void ResetCityData() => CurrentCityType = GameRules.StartingCity;

private void GenerateCityIfSaveEmpty()
```

```
{
    if (mapData.Map.Count == 0)
    {
        CurrentCityType = CityType.City6X6;
        GenerateCity();
    }
}

private void GenerateCity()
{
    mapData.ClearData();
    mapData.FillMapDict(false);

    roadsGeneration.GenerateCityRoads();

    mapData.FillMapDict(true); //*EXPAND MAP

    roadsGeneration.GenerateRoadsToEdge();
    roadsGeneration.PlaceRoads();

    buildingsGeneration.GenerateBuildings();

    environmentGeneration.PlaceEnvironmentalObjects();

    OnCityGenerated?.Invoke(mapData);
    AudioManager.Instance.PlaySound(AudioClipType.GameStart);
}

private void MoveToNewCity()
{
    if(CurrentCityType != CityType.City20X20) CurrentCityType++;
    GenerateCity();
}

public object Save()
```

```
        {
            MapSave save = mapData.GetMapSave();
            save.cityType = CurrentCityType;

            return save;
        }

        private void OnDestroy()
        {
            GameManager.OnGameReset -= ResetCityData;
            GameManager.OnGameReset -= GenerateCity;
            GameManager.OnMovedToNewCity -= MoveToNewCity;
            SaveManager.OnGameLoaded -= GenerateCityIfSaveEmpty;
        }
    }
}
```

Файл EnvironmentGeneration.cs

```
using System.Collections.Generic;
using GeneralSystem;
using Unity.Mathematics;
using UnityEngine;
using UtilitySystems;

namespace Generation
{
    public class EnvironmentGeneration : MonoBehaviour
    {
        private MapData _mapData;
        private int _edgeForestLength;

        public void Init(MapData mapData) => _mapData = mapData;

        public void PlaceEnvironmentalObjects()
        {
```

```
            _edgeForestLength = _mapData.cityInfo.edgeForestLength;
            int campfiresAmount = _mapData.cityInfo.campfiresAmount;
            int treeTilesAmount = _mapData.cityInfo.treeTilesAmount;
            int envInsideCity = _mapData.cityInfo.envInsideCityAmount;

            PlaceForestOnEdge();

            PlaceDecorationObjects(campfiresAmount, CityObjectType.EnvOutsideCity,
EnvObjSpawnType.OutsideCity);
            PlaceDecorationObjects(envInsideCity, CityObjectType.EnvInsideCity,
EnvObjSpawnType.InsideCity);
            PlaceDecorationObjects(treeTilesAmount, CityObjectType.Forest,
EnvObjSpawnType.AllMap);
        }

        private void PlaceForestOnEdge()
        {
            int2 startCoord = new int2(-_mapData.cityInfo.tilesToEdge, -
_mapData.cityInfo.tilesToEdge);
            int2 endCoord = _mapData.cityInfo.cityBorders + new
int2(_mapData.cityInfo.tilesToEdge, _mapData.cityInfo.tilesToEdge);

            for(int j = startCoord.y; j < endCoord.y; j++)
            for(int i = startCoord.x; i < endCoord.x; i++)
            {
                int2 currentPos = new int2(i, j);
                if(i < startCoord.x + _edgeForestLength) PlaceForestTile(currentPos);
                else if(i >= endCoord.x - _edgeForestLength)
PlaceForestTile(currentPos);
                else if(j < startCoord.y + _edgeForestLength)
PlaceForestTile(currentPos);
                else if(j >= endCoord.y - _edgeForestLength)
PlaceForestTile(currentPos);
            }
        }

        private void PlaceForestTile(int2 position)
        {
            if(_mapData.IsTilePopulated(position)) return;
```

```
            CityObjectType type = CityObjectType.Forest;
            int representationID = Storage.GetRandomRepresentationID(type);
            GameObject spawnedObject =
GameObjectPool.Instance.GetObject(Storage.GetRepresentationByID(type,
representationID),
                                              Utility.GetWorldCoord(position), new Vector3(0,
Utility.GetRandomRotation(), 0));

            _mapData.Map[position] = new Tile(TileType.EnvironmentalObject, type,
representationID, spawnedObject);
        }

        private List<int2> _availablePositions = new List<int2>();
        private void PlaceDecorationObjects(int maxAmount, CityObjectType type,
EnvObjSpawnType spawnType)
        {
            _availablePositions.Clear();
            _availablePositions = _mapData.GetPositionsForEnvObjects(spawnType);

            for (int i = 0; i < maxAmount; i++)
            {
                if(_availablePositions.Count == 0) return;
                int2 randomPos = _availablePositions.RandomElement();

                int representationID = Storage.GetRandomRepresentationID(type);
                GameObject spawnedObject =
GameObjectPool.Instance.GetObject(Storage.GetRepresentationByID(type,
representationID),
                                                Utility.GetWorldCoord(randomPos), new
Vector3(0, Utility.GetRandomRotation(), 0));

                _mapData.Map[randomPos] = new Tile(TileType.EnvironmentalObject, type,
representationID, spawnedObject);

                _availablePositions.Remove(randomPos);
            }
        }
    }
    public enum EnvObjSpawnType
```

```
    {
        InsideCity,
        OutsideCity,
        AllMap
    }
}
```

Файл RoadsGeneration.cs

```
using Random = UnityEngine.Random;
using System.Collections.Generic;
using GeneralSystem;
using Unity.Mathematics;
using UnityEngine;
using UtilitySystems;

namespace Generation
{
    public class RoadsGeneration : MonoBehaviour
    {
        private MapData _mapData;

        private readonly List<int> _segmentSizes = new List<int> {3,4,6};
        private readonly List<int2> _roads = new List<int2>();

        public void Init(MapData mapData) => _mapData = mapData;

        public void GenerateCityRoads()
        {
            ClearData();

            GenerateRectangle(int2.zero, _mapData.cityInfo.cityBorders); //GENERATING
BORDERS
            FillWithRandomRectangles(_mapData.cityInfo.cityBorders);
        }
```

```
        private void ClearData() => _roads.Clear();

        private void GenerateRectangle(int2 startCoord, int2 endCoord)
        {
            for(int j = startCoord.y; j < endCoord.y; j++)
            for(int i = startCoord.x; i < endCoord.x; i++)
            {
                int2 currentPos = new int2(i, j);
                if(i == startCoord.x) PlaceRoadIfNoRoad(currentPos);
                else if(i == endCoord.x - 1) PlaceRoadIfNoRoad(currentPos);
                else if(j == startCoord.y) PlaceRoadIfNoRoad(currentPos);
                else if(j == endCoord.y - 1) PlaceRoadIfNoRoad(currentPos);
            }
        }

        private void FillWithRandomRectangles(int2 cityBorders)
        {
            int2 currentCoord = int2.zero;

            for(int i = 0; i < 1000; i++)
            {
                if(currentCoord.x >= cityBorders.x - _segmentSizes[0]) break;
                currentCoord.y = 0;

                for(int j = 0; j < 1000; j++)
                {
                    if(currentCoord.y >= cityBorders.y - _segmentSizes[0]) break;

                    int2 rectSize = new int2(_segmentSizes.RandomElement(),
_segmentSizes.RandomElement());
                    if(currentCoord.y + rectSize.y == cityBorders.y - 1) rectSize.y +=
1;

                    int offsetX = 0;
                    if(i > 0 && rectSize.x > _segmentSizes[0] + 1) offsetX = 2 *
Random.Range(0, 2);
```

```
                    if(currentCoord.x + rectSize.x == cityBorders.x - 1) rectSize.x +=
1;

                    if(currentCoord.x + offsetX == cityBorders.x - 2) break;


                    GenerateRectangle(currentCoord + new int2(offsetX, 0), new
int2(currentCoord.x + rectSize.x, currentCoord.y + rectSize.y));


                    currentCoord.y = currentCoord.y + rectSize.y - 1;

                }

                currentCoord.x += _segmentSizes[_segmentSizes.Count - 1] - 1;

            }

        }


        public void GenerateRoadsToEdge()

        {

            int2 cityBorders = _mapData.cityInfo.cityBorders;

            int tilesToEdge = _mapData.cityInfo.tilesToEdge;


            int4 randomCoord = new int4(Random.Range(1, cityBorders.y - 1),
Random.Range(1, cityBorders.y - 1), Random.Range(1, cityBorders.y - 1), Random.Range(1,
cityBorders.y - 1));

            for(int i = 0; i < tilesToEdge; i++)

            {

                PlaceRoadIfNoRoad(new int2(cityBorders.x + i, randomCoord[0])); //*
RIGHT BORDER

                PlaceRoadIfNoRoad(new int2(-tilesToEdge + i, randomCoord[1])); //* LEFT
BORDER

                PlaceRoadIfNoRoad(new int2(randomCoord[2], cityBorders.y + i)); //* TOP
BORDER

                PlaceRoadIfNoRoad(new int2(randomCoord[3], -tilesToEdge + i)); //*
BOTTOM BORDER

            }

        }


        public void PlaceRoads()

        {

            foreach(int2 pos in _roads)

                PickRoadVariant(pos);

        }
```

```
        private void PlaceRoadIfNoRoad(int2 pos)
        {
            if(_mapData.IsTilePopulated(pos)) return;

            _mapData.Map[pos] = new Tile (TileType.Road, CityObjectType.None);
            _roads.Add(pos);
        }

    #region RoadRules
        private readonly Dictionary<bool4, (CityObjectType type, Vector3 rotation)>
_roadVariants = new Dictionary<bool4, (CityObjectType, Vector3)>()
        {
            //* RIGHT || LEFT || UP || DOWN
            {new bool4(false, false, true, true), (CityObjectType.RoadStraight,
Vector3.zero)}, //* VERTICAL ROAD
            {new bool4(true, true, false, false), (CityObjectType.RoadStraight, new
Vector3(0f, 90f, 0f))}, //* HORIZONTAL ROAD\

            {new bool4(true, true, true, true), (CityObjectType.RoadCross,
Vector3.zero)}, //* CROSS ROAD

            {new bool4(false, true, true, false), (CityObjectType.RoadTurn,
Vector3.zero)}, //* TURN UP+LEFT
            {new bool4(true, false, true, false), (CityObjectType.RoadTurn, new
Vector3(0f, 90f, 0f))}, //* TURN UP+RIGHT
            {new bool4(false, true, false, true), (CityObjectType.RoadTurn, new
Vector3(0f, -90f, 0f))}, //* TURN DOWN+RIGHT
            {new bool4(true, false, false, true), (CityObjectType.RoadTurn, new
Vector3(0f, 180f, 0f))}, //* TURN DOWN+LEFT

            {new bool4(false, true, true, true), (CityObjectType.RoadT, Vector3.zero)},
//* T UP+DOWN+LEFT
            {new bool4(true, false, true, true), (CityObjectType.RoadT, new Vector3(0f,
180f, 0f))}, //* T UP+DOWN+RIGHT
            {new bool4(true, true, false, true), (CityObjectType.RoadT, new Vector3(0f,
-90, 0f))}, //* T DOWN+RIGHT+LEFT
            {new bool4(true, true, true, false), (CityObjectType.RoadT, new Vector3(0f,
90, 0f))}, //* T UP+DOWN+RIGHT
```

```
            {new bool4(true, false, false, false), (CityObjectType.RoadStraight, new
Vector3(0f, 90f, 0f))}, //* LEFT ROAD EDGE

            {new bool4(false, true, false, false), (CityObjectType.RoadStraight, new
Vector3(0f, 90f, 0f))}, //* RIGHT ROAD EDGE

            {new bool4(false, false, true, false), (CityObjectType.RoadStraight,
Vector3.zero)}, //* BOTTOM ROAD EDGE

            {new bool4(false, false, false, true), (CityObjectType.RoadStraight,
Vector3.zero)}, //* TOP ROAD EDGE

        };


        private void PickRoadVariant(int2 position)

        {

            bool4 connections = new bool4(false);

            for(int i = 0; i < 4; i++)

                connections[i] = _roads.Contains(position + Utility.Directions[i]);


            CityObjectType type = _roadVariants[connections].type;

            Vector3 rotation = _roadVariants[connections].rotation;


            int representationID = Storage.GetRandomRepresentationID(type);

            GameObject spawnedRoad =
GameObjectPool.Instance.GetObject(Storage.GetRepresentationByID(type,
representationID), Utility.GetWorldCoord(position), rotation);

            _mapData.Map[position] = new Tile(TileType.Road, type, representationID,
spawnedRoad);

        }

    #endregion

    }

}
```

Файл RoutesManager.cs

```
using Unity.Mathematics;

using UnityEngine;

using System;

using Cards;

using Cards.CardsActions;

using GeneralSystem;

using Orders;
```

```
using Saving;
using UI.InGame;
using UI.MainMenu;
using UtilitySystems;

namespace Routes
{
    public class RoutesManager : MonoBehaviour, ISavable
    {
        public static RoutesManager Instance;

        public static event Action<EditingActionType> OnEditingActionChanged;
        public static event Action<int> OnRouteSelected;

        [SerializeField] private RoutesBuilder routesBuilder;
        private int _currentRouteID;

        private EditingActionType _editingAction = EditingActionType.Drawing;
        public EditingActionType EditingAction
        {
            get => _editingAction;
            set
            {
                _editingAction = value;
                OnEditingActionChanged?.Invoke(_editingAction);
            }
        }

        public bool editingAllowed;

        private void Awake()
        {
            Instance = this;

            GameManager.OnGameReset += ResetRoutesManager;
            GameManager.OnMovedToNewCity += ResetRoutesManager;
```

```
            RoutesBuilder.OnRouteCreated += PreventEditing;
            CompleteRoute.OnCompleteRouteUsed += CompleteOrdersOnRoute;
            CardsView.OnCardsViewShown += PreventClick;
            MainMenuUI.OnMenuShown += PreventClick;
            GgScreen.OnGgScreenShown += PreventClick;
        }

        public Type SaveDataType => typeof(RoutesSave);

        private void ResetRoutesManager()
        {
            SelectRoute(0);
            EditingAction = EditingActionType.Drawing;
        }

        public void Load(object data)
        {
            RoutesSave save = (RoutesSave)data;
            routesBuilder.LoadData(save);
            SelectRoute(0);
        }

        private bool _clickBlocked;
        private void PreventClick()
        {
            _clickBlocked = true;
            editingAllowed = false;
        }

        private void Update()
        {
            if(CardsView.ViewShown || GgScreen.GgScreenShown ||
MainMenuUI.MainMenuShown) return;

            if (Input.GetMouseButtonDown(0) && !Input.GetMouseButton(1) &&
_clickBlocked == false)
            {
```

```
            EditingAction = EditingActionType.Drawing;
            editingAllowed = true;
        }
        if (Input.GetMouseButtonDown(1) && !Input.GetMouseButton(0))
        {
            editingAllowed = true;
            EditingAction = EditingActionType.Erasing;
        }
        if(Input.GetMouseButtonUp(0))
        {
            if (_clickBlocked)
            {
                _clickBlocked = false;
                return;
            }

            if (EditingAction != EditingActionType.Erasing)
            {
                TimeManager.ResumeFromRouteBuilt();
                routesBuilder.RemoveAllUnfinishedRoutes();
            }
            editingAllowed = false;
            SelectRoute(routesBuilder.lastModifiedRoute);
        }
        if(Input.GetMouseButtonUp(1)) editingAllowed = false;
    }

    public void ModifyRoute(int2 position)
    {
        if(editingAllowed == false) return;

        if(EditingAction == EditingActionType.Drawing)
routesBuilder.BuildRoute(position, _currentRouteID);
        else if(EditingAction == EditingActionType.Erasing)
routesBuilder.EraseRoute(position, _currentRouteID);
    }
```

```
        public void SelectRoute(int routeIndex, bool carSelect = false)
        {
            if (carSelect) routesBuilder.lastModifiedRoute = routeIndex;
            _currentRouteID = routeIndex;
            routesBuilder.SetCurrentRoute(routeIndex);

            OnRouteSelected?.Invoke(_currentRouteID);
        }

        public bool CanAddRoute() => routesBuilder.CanAddRoute;
        public bool CanCompleteRoute() => routesBuilder.CanCompleteRoute;

        private void CompleteOrdersOnRoute()
        {
            foreach (Vector3 pos in routesBuilder.CurrentRouteCoords)
                OrdersManager.Instance.TryCompleteOrder(Utility.GetGridCoord(pos),
true);
        }

        public void RemoveRoute(int routeIndex) =>
routesBuilder.RemoveRoute(routeIndex);

        private void PreventEditing(Route route) => editingAllowed = false;

        public object Save() => routesBuilder.GetRoutesSave();

        private void OnDestroy()
        {
            GameManager.OnGameReset -= ResetRoutesManager;
            GameManager.OnMovedToNewCity -= ResetRoutesManager;
            RoutesBuilder.OnRouteCreated -= PreventEditing;
            CompleteRoute.OnCompleteRouteUsed -= CompleteOrdersOnRoute;
            CardsView.OnCardsViewShown -= PreventClick;
            MainMenuUI.OnMenuShown -= PreventClick;
            GgScreen.OnGgScreenShown -= PreventClick;
        }
    }
```

```
    public enum EditingActionType
    {
        Drawing,
        Erasing,
    }
}
```

Додаток Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

Додаток В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Дробний.О.Ю.docx	Пояснювальна записка до дипломного проекту. Документ Word.
Диплом_Дробний.О.Ю.docx	Пояснювальна записка до дипломного проекту в форматі PDF
Програма	
Game.rar	Архів. Містить файли гри
Додаткові файли	
Scripts.rar	Архів. Містить коди гри
Презентація	
Презентація_Дробний.О.Ю.ppt	Презентація дипломного проекту