ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр
на тему: «**Розробка гри «Delivery Service» у жанрі симулятор мовою C# на двигуні Unity**»

Виконав: студент 5 курсу, групи ППЗ-51
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Сендецький Микола Ігорович
(прізвище та ініціали)

Керівник Коба А.Б.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Київ - 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

О.В. Негоденко
“ ” 2022 року

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

Сендецькому Миколі Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи: **«Розробка гри «Delivery Service» у жанрі симулятор мовою C# на двигуні Unity»,**
Керівник роботи Коба Андрій Борисович старший викладач кафедри , затверджені наказом вищого навчального закладу від 18.02.2022 року №
2. Строк подання студентом роботи 03.06.2022 року
3. Вихідні дані до роботи:
   3.1 Інформаційно-аналітичний метод;
   3.2 Метрики програмного забезпечення;
   3.3 Програмне середовище мови програмування C#;
   3.4 Науково-технічна література з питань, пов’язаних програмуванням на мові C#.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
   4.1 Аналіз предметної галузі і виділення переваги розробленого ПЗ;
   4.2 Розробка гри симулятора мовою C# на двигуні Unity;
   4.3 Тестування гри симулятора.
5.Перелік графічного матеріалу (назва слайдів презентації):
   1. Назва роботи;
   2. Мета, об’єкт та предмет дослідження;

3. Аналоги;
4. Таблиця порівняння аналогів;
5. Технічні завдання;
6. Програмні засоби реалізації гри;
7. Алгоритм «Delivery Service»;
8. Діаграма класів «Delivery Service»;
9. Діаграма прецедентів «Delivery Service»;
10. Діаграма компонентів «Delivery Service»;
11. Головне меню та ігрове поле з інтерфейсом користувача;
12. Основні панелі інтерфейсу користувача;
13. Апробація результатів дослідження;
14. Висновки.

6. Дата видачі завдання 11.04.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1.	Підбір науково-технічної літератури	11.04 – 14.04	Викон.
2.	Вивчення та аналіз задачі	15.04 – 17.04	Викон.
3.	Розробка структури додатку	18.04 – 21.04	Викон.
4.	Розробка дизайну та графічних елементів	22.04 – 25.04	Викон.
5.	Програмна реалізація системи	26.04 – 05.05	Викон.
6.	Налагодження програми	05.05 – 07.05	Викон.
7.	Вступ, висновки, реферат	07.05 – 10.05	Викон.
8.	Розробка обов’язкових демонстраційних матеріалів	11.05 – 15.05	Викон.
9.	Попередній захист роботи	16.05 – 01.06	Викон.
10.	Подання роботи в деканат	03.06	Викон.

Студент Сендецький М.І.

(підпис) (прізвище та ініціали)

Керівник роботи Коба А.Б.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 86 сторінок, 38 рисунків, 19 джерел.

Об'єкт дослідження – процес розробки гри симулятора мовою C# на двигуні Unity.

Предмет дослідження – методи та засоби розробки гри симулятора «Delivery Service».

Мета роботи – розробка гри «Delivery Service» у жанрі симулятор мовою C# на двигуні Unity з використанням нових механік та властивостей.

Проведено аналіз розробки гри симулятора. Розглянуто аналоги ігор симуляторів служби доставки. Всі вони абсолютно різні. Пропонують різні активності та їх сукупності.

Проведено аналіз засобів розробки програмного забезпечення. Обрано мову програмування C#, середовище розробки Unity та візуальний редактор коду Microsoft Visual Studio 2022. Визначено об'єкт, предмет та мету бакалаврської роботи.

Розроблено приклад гри симулятора мовою C# на двигуні Unity.

Галузь використання – інформаційні технології.

ДВИГУН UNITY, MICROSOFT VISUAL STUDIO 2022, МОВА C#, ОБ'ЄКТНА ОРІЄНТАЦІЯ ПРОГРАМ, ТИПИ, КЛАСИ, ОБ'ЄКТИ

Зміст

Вступ

Гра — діяльність людини з моделювання іншого виду діяльності з розважальною чи навчальною метою. Людство грається і грає в ігри з доісторичних часів, починаючи з ритуальних ігор (наприклад, обряду ініціації), а з розвитком цивілізації ігри робилися дедалі складнішими й відобразили практично всі сфери життя суспільства: війну, кохання, історію тощо.

Ймовірно, найскладніші нині ігри — MMORPG з режимом мультиплеєра на кшталт World of Warcraft, до якого щохвилини підключені тисячі користувачів у всьому світі, й у якому щомиті відбуваються мільйони різних дій.

Симулятором менеджменту та будування називають тип симуляційних відеоігор в якому гравець займається розбудовою чи розширенням здебільшого вигаданих спільнот, установ, імперій тощо з обмеженою кількістю доступних йому ресурсів, має реагувати на внутрішні проблеми (такі як злочини чи забруднення, як в SimCity), незалежні від дій гравця події (наприклад, стихійні лиха чи монстри, як в SimCity, або рівень необхідності тераформування планет, як в Spaceward Ho!) чи конкурувати з іншими гравцями. В однокористувацьких відеоіграх даного піджанру проходження здебільшого є необмеженим у часі, а кінець може настати тоді, коли гравець сам того захоче, в той час як у багатокористувацьких відеоіграх — проходження, як правило, триває допоки один із гравців не почне домінувати над іншими.

Відеоігри жанру симулятора менеджменту та будування також можна розділити на два підтипи: симулятор бізнесу та симулятор містобудування. В першому випадку гравцеві необхідно буде займатися розвитком якогось конкретного підприємства, або цілої мережі підприємств, в той час як гравцям відеоігор жанру симулятора містобудування необхідно буде розширювати та розбудовувати вже цілу економічну систему, тобто міста чи поселення.

Delivery Service (в перекладі «Служба доставки») – це гра про розвиток нового філіалу в невеликому містечку. Гра «Delivery Service» виконана у жанрі симулятора. Головним героєм гри є звичайний адміністратор, якого призначили керівником філіалу та поставили ціль – заробити більше грошей.

Метою дипломного проекту було створення гри «Delivery Service» на основі класів з використанням концепцій та механізмів об’єктно-орієнтованого програмування. Гра була написана за допомогою мови програмування C#, з використанням ігрового двигуна Unity та візуального редактора коду Microsoft Visual Studio 2022.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ І ВИДІЛЕННЯ ПЕРЕВАГИ РОЗРОБЛЕНОГО ПЗ

1.1 Аналіз предметної галузі

Відеогра́ — це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотну інформацію з відеопристрою. Електронні пристрої, які використовуються для того щоб грати, називаються ігровими платформами. Наприклад, до таких платформ належать персональний комп'ютер та гральна консоль. Пристрій введення, який використовується для керування грою, називається ігровим контролером. Це може бути, наприклад, джойстик, клавіатура та мишка, геймпад або сенсорний екран.

У 2011 році відеоігри були офіційно визнані видом мистецтва урядом США та Національним фондом мистецтв США. Однак, загальносвітове визнання їх мистецтвом лишається дискутивним питанням.

Нерідко відеоігри в повсякденному житті та пресі називають комп'ютерними іграми. Проте розробники та дослідники таких ігор послуговуються терміном «відеогра», виходячи з їх специфіки та місця в ієрархії ігор загалом:

- Гра — діяльність із розважальною і/або навчальною метою за встановленими правилами. Включає в себе такі різновиди як дитячі, спортивні, настільні, азартні, рольові та, окрім інших, електронні ігри.
- Електронна гра — гра, що відбувається за допомогою електронних пристроїв, програмованих чи непрограмованих, які утворюють інтерактивну систему. Такими можуть бути пінбольні автомати, слотові машини, електромеханічні ігри, лотереї, аудіоігри і відеоігри. Оскільки відеоігри на програмованих пристроях складають основну їх частину, нерідко терміни «електронна гра», «комп'ютерна гра» і «відеогра» вживаються як тотожні.

- Комп'ютерна гра — гра, яка забезпечується програмно керованим електронним пристроєм — комп'ютером. Це також може бути слотовий автомат, пінбольна машина, аудіо чи відеогра.
- Відеогра — гра, яка відбувається через керування візуальними образами на моніторі чи іншому дисплеї. Може забезпечуватися як програмованим, так і не програмованим електронним пристроєм: аркадним ігровим автоматом, ігровою приставкою, персональним комп'ютером. Зрідка під відеогрою розуміється гра тільки для ігрової приставки.
- Гра для персонального комп'ютера — відеогра, яка запускається на персональному комп'ютері. Часто називається просто комп'ютерною грою, але не тотожна комп'ютерній грі, описаній вище.

Класифікація відеоігор. Відеоігри, так само як і музичні чи літературні твори, можна розподілити на жанри. Щодо відеоігор жанри виділяються на основі спільних характеристик ігрового процесу або його цілей. Часто у одній відеогрі поєднуються декілька жанрів (наприклад, більшість сучасних рольових ігор мають елементи екшн).

Єдино визначеної класифікації жанрів відеоігор не існує, проте в більшості класифікацій виділяються основні:

Пригодницькі — де дія відбувається в рамках визначеної історії та передбачає детальне дослідження ігрового світу. Пригодницькі ігри менш покладаються на візуальні образи і більшою мірою на переживання сюжету. Нерідко передбачають вирішення головоломок.

Екшн — ігри, де гравцеві слід в більшості покладатися на швидкість реакції. Дія гри, як правило, зосереджена на різного роду боях, в ході якої гравець повинен вчасно виконувати потрібні дії. Небойові завдання можуть включати в себе уникнення пасток, проходження місць за визначений час і т.д.

Гоночні — цей жанр включає в себе всі ігри, в яких участь у перегонах різного роду є основою ігрового процесу. Зазвичай гоночні ігри використовують автомобілі та інші транспортні засоби.

Рольові — рольові відеоігри походять від настільних рольових ігор, звідки взяли ігрову механіку. Гравець «грає роль», певного персонажа, який з часом і в міру виконання поставлених завдань розвивається.

Стратегічні — в широкому сенсі стратегічним відеоіграми є ті, де запорукою перемоги є розв'язання проблем шляхом попереднього обдумування та планування. У більш вузькому, стратегічні ігри відтворюють збройні конфлікти, де гравець керує арміями чи країнами, що вимагає стратегічного мислення.

Симулятори — різною мірою реалістично відтворюють якийсь з аспектів реального життя. Наприклад, існують симулятори побачень чи управління літаком.

Навчальні — служать для навчання гравця в якійсь області. Зазвичай призначені для дітей, проте існують і навчальні ігри для дорослих.

Спортивні — ігри, які відтворюють реальні (футбол, хокей) чи вигадані (квідич) види спорту.

Також відеоігри розрізняються за тематикою: фентезійні, детективні, жахи і т. д. За перспективою: від першої особи, від третьої особи, ізометричні, з видом збоку/згори.

Симуляційна відеогра, або симулятор — це жанр відеоігор, характерною особливістю якого є якомога точніше відтворення фізичних законів реального світу, властивостей реальних предметів, процесів та подій.

Історія симуляторів починається водночас з історією відеоігор. У 1947 році було створено електронну гру на базі електронно-променевої трубки — Cathode Ray Tube Amusement Device. Однак сам апарат не мав цифрового процесора для обробки інформації, а використовував аналогові ланцюги для управління електронно-променевою трубкою і формування зображення на екрані. За виглядом

він нагадував радар часів Другої світової війни, а для прицілювання використовувалися екранні накладання. З часом почалася розробка симуляторів шахів, переважно у 1950-х роках. Спортивні симулятори зростали в кількості. У 1958 році було випущено Tennis for Two.

У 1962 році було випущено космічний симулятор Spacewar!. Його основний ігровий процес має два збройних космічних корабля, які намагаються стріляти один в одного, маневруючи в невагомості між зірок. Кораблі обстрілювали ракетами, які рухалися під дією сили тяжіння. У кожного корабля була обмежена кількість ракет і обмежений запас палива. Функція гіперпростору могла бути використана як останній засіб обходу ворожих ракет, але поява гіперпростору відбувалася у випадковому місці й була ймовірність того, що корабель вибухне при використанні. Гравець може контролювати обертання за годинниковою стрілкою і проти годинникової стрілки, тягу, стрілянину і гіперпростір. Спочатку гра контролювалася за допомогою перемикачів на передній панелі (по чотири для кожного гравця), але з часом стало ясно, що вони дуже швидко зношуються при звичайній грі. Більшість гравців стало використовувати призначені для користувача провідні блоки управління.

У 1972 вийшла відеогра для аркадних ігрових автоматів — Pong, яка була розроблена японською компанією Taito Corporation і випущена компанією Atari. Pong швидко став популярним, і став першим комерційно успішним аркадним ігровим автоматом, який стояв біля витоків ігрової індустрії разом з першою грального консоллю Magnavox Odyssey. Незабаром після його виходу, кілька компаній почали виробляти ігри, які копіювали *Pong*, але в підсумку випустили нові типи ігор. В результаті, Atari закликав своїх співробітників виробляти більше інноваційних ігор. Компанія випустила кілька сиквелів, які були побудовані на геймплеї оригіналу, додаючи нові функції. Під час різдвяного сезону 1975 року, Atari випустила консольну версію *Pong*, реалізуючи її виключно через роздрібні магазини Sears. Це також мало комерційний успіх і призвело до появи численних

копій. Гра була перероблена на численних гральних консолях та портативних платформах. *Pong* пародіювався в декількох телевізійних шоу і відеоіграх, і був частиною культурних виставок.

У 1974 Atari випустила Gran Trak 10 — перший автосимулятор. У 1981 році було випущено President Elect — перший офіційний політичний симулятор. *Bullfrog Productions* випустила гру *Populous*, яка стала одним з перших вдалих симуляторів бога. Вес Черрі створив гру «Косинка» — реалізацію пасьянсу «Косинка», а Роберт Доннер написав *Minesweeper*, які постачаються з кожною версією Microsoft Windows, починаючи з третьої.

У 1994 році було випущено The Need for Speed — симулятор автоперегонів. Оригінал *The Need for Speed* був випущений для 3DO в 1994 році, незабаром після цього були наступні версії для PC-DOS (1995), PlayStation і Sega Saturn (1996). Спеціальне видання *The Need for Speed* було випущено тільки для системи DOS. У наступних версіях на ПК тільки для Windows.

Перший випуск в серії був однією з усього лише двох серйозних спроб, щоб забезпечити реалістичну модель керованості автомобіля і фізики без аркадних елементів (другий Porsche Unleashed). Electronic Arts спільно з автомобільним журналом Road & Track домоглися в грі відповідності з поведінкою реального автомобіля, в тому числі відтворення звуку важеля управління коробкою передач. Гра також містить точні дані про транспортний засіб з усними коментарями, як «стилістика журналу» із зображенням інтер'єру та екстер'єру кожного автомобіля, включаючи навіть короткі відеоролики, покладені на музику.

Більшість автомобілів і трас доступні на початку гри, і мета полягає в тому, щоб розблокувати решту, вигравши всі турніри. Перша версія є погонею з поліцейськими автомобілями, які залишаються популярною темою протягом всієї серії — так звані Hot Pursuit видання (*Need for Speed III: Hot Pursuit, Need for Speed: High Stakes, Need for Speed: Hot Pursuit 2, Need for Speed: Most Wanted, Need for Speed: Carbon, Need for Speed: Undercover, Need for Speed: Hot Pursuit (2010)*) і

продаються краще на ринку, ніж проміжні версії. У початковій версії також фігурує неприємний суперник, який насміхався над гравцем, якщо комп'ютер вигравав гонку або гравець був заарештований (якщо гравець попереджений кілька разів).

Інша версія гри, розроблена в 1995 році, яка отримала назву *The Need for Speed: Special Edition*, була випущена в 1996 році тільки для CD-ROM на ПК з рекомендованою підтримкою DirectX 2 і TCP/IP. Це видання включає в себе: дві нові траси, зміну часу доби для більшості трас (ранок, день і вечір), і різні поліпшення рушія гри.

У 2005 році було випущено Silent Hunter III. Під командування гравцеві дається німецький підводний човен часів Другої Світової війни. У гру включені 5 тренувальних місій, динамічна кампанія і поодинокі (разові) місії. З головного меню доступна тривимірна енциклопедія судів і авіації того періоду для різних країн. Цілі одиночних і тренувальних місій роз'яснюються перед їх запуском. Перед кожною місією гравцеві надається можливість вибрати рівень реалізму геймплея: вразливість судів і кораблів противника, своєї підводного човна, браковані торпеди, стабілізацію перископа тощо. У динамічній кампанії гравцеві пропонується роль командира підводного човна протягом низки бойових походів з широкою самостійністю — для кожного походу призначається лише квадрат патрулювання. Гравець може сам вибирати яким шляхом слідувати, на який субмарині, вибирати цілі для атаки, приймати бій чи йти від нього.

1.2 Аналіз аналогів ігор симуляторів

Більшість аналогів являють собою браузерні проекти. В одних нам необхідно керувати автомобілем(див. рис. 1.1), пересуваючись від точки то точки, намагаючись їхати якомога обережніше оскільки зіткнення зі сторонніми об’єктами знижують

кількість здоров’я автомобіля. За успішну доставку нам дають гроші, які ми можемо витратити на модифікацію автомобіля та кастомізацію.

Рисунок 1.1 – «Служба доставки онлайн»

В інших випробовується реакція гравця. На ігровому полі з’являються замовники разом із замовленням (див. рис. 1.2), яке необхідно перетягнути у відповідну категорію замовлень. Після чого до замовника відправляється автомобіль. Після доставки на місці замовника залишається нагорода. Якщо забаритися з перетягуванням замовлення чи отриманням нагороди, то вони зникають. З кожним рівнем кількість категорій замовлень збільшується, як і загальна кількість замовників.

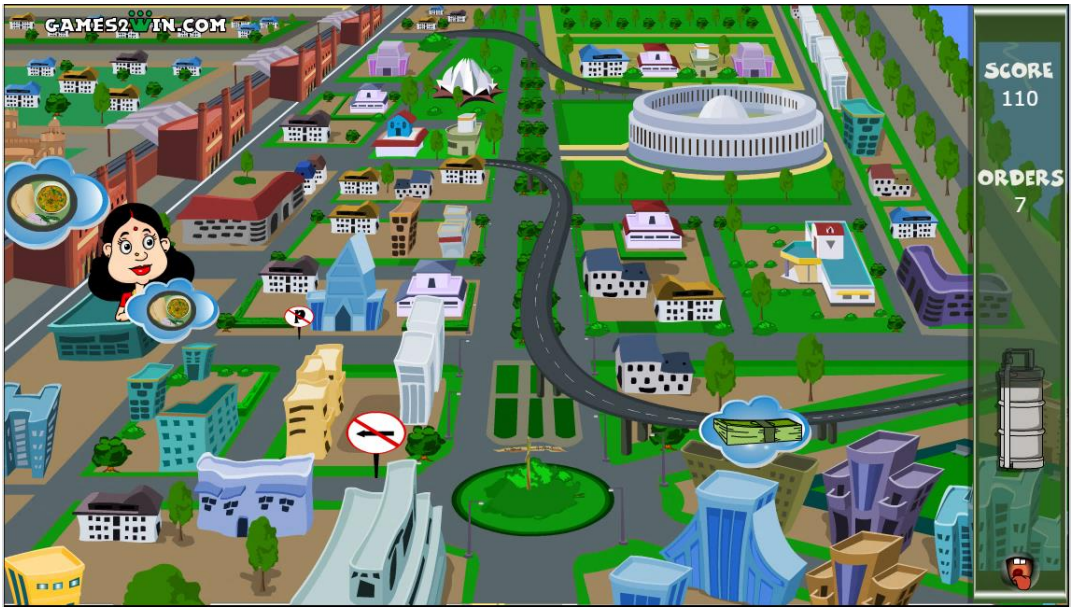


Рисунок 1.2 – «Dabbawalla. International»

Деякі являють собою 3D пісочницю (див. рис. 1.3) з непередбачуваною фізикою і можливістю роботи в команді, що робить з них божевільну суміш. «Пристібайтесь і заводьте машину - посилки самі себе не доставлять! Об'єднуйтесь з друзями в команди до 4-х чоловік і грайте в симулятор з непередбачуваною фізикою для жахливих кур'єрів доставки. Працюйте спільно, використовуйте химерні пристрої, корисні гаджети та чудеса фізики, щоб посилки дісталися своїх одержувачів. Розвозіть посилки поодинці, якщо хочете контролювати кожен крок, або з друзями — перевірте, наскільки ви є гарною командою. Дізнайтеся, що буває, коли непередбачувана фізика поєднується з юрким платформінгом. Бігайте, скачіть, пікіруйте, але варто вам з чимось зіткнутися і вас вирубає, не сумнівайтеся! Влаштуйте собі перерву і трохи відпочиньте! Світ до країв наповнений іграшками, транспортом та пристроями, які можна використовувати як для роботи, так і для розваг».

Рисунок 1.3 – «Totally Reliable Delivery Service»

Крім того є реалізація принципу цієї гри у виді настолки «Wasteland Express. Delivery Service» (див. рис. 1.4). «Ласкаво просимо в службу доставки Пусток! І якщо ви думаєте, що у вас буде шанс стати героєм, то ви жорстоко помиляєтеся. Цьому світу безумців з автоматами цілком вистачає, а ось хороших далекобійників, здатних провезти по закинутим землям все, що завгодно, явно мало. Так що, браток, сідай за кермо, і сподівайся, що твій напарник пристрілявся вже до кулемета. Пустки, вони хоч і пустельні, але точно не порожні. Рейдери та й твої братики далекобійники, вже можуть чекати тебе на дорозі. Ну і сам теж не втрачайся, якщо що. Те, що сталося в пустці, в пустках і залишиться».

Рисунок 1.4 – «Служба доставки. Пустельний Експрес»

1.3 Огляд засобів програмної реалізації гри симулятора

C# - об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (при фірмі Microsoft).

Ключові особливості мови C#:

1) Компонентна орієнтованість.

2) Код зібраний воєдино (декларації і реалізації об'єднані разом).

3) Уніфікована система типів і їх безпечність.

4) Автоматична і мануальна робота з пам'яттю.

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато чого від своїх попередників - мов C++, Delphi і Smalltalk - C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад множинне спадкування класів (на відміну від C++).

C# є дуже близьким родичем мови програмування Java. Мова Java була створена компанією Sun Micro systems, коли глобальний розвиток інтернету поставив задачу роззосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу вказівників без контролю виходу за межі). Для роззосереджених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програм і апаратними інструкціями комп'ютера чи іншого інтелектуального пристрою.

Нововведенням C# стала можливість легшої взаємодії, порівняно з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проектів. Якщо програми на різних мовах виконуються на

платформі .NET, .NET бере на себе клопіт щодо сумісності програм (тобто типів даних, за кінцевим рахунком).

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#.

Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідній конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі.

Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET. CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо.

C# має «препроцесорні директиви» (хоча насправді він не має препроцесора) на основі препроцесора C, це дає програмісту можливість визначити символи, але не макроси. Умовні директиви, такі як #if, #endif, чи #else також можливі. Директиви типу #region дають натяк редактору для згортання фрагментів коду.

Специфікація C# визначає мінімальний набір бібліотек типів і класів, на який має розраховувати компілятор. На практиці, C# найчастіше використовується з якоюсь реалізацією Common Language Infrastructure (CLI), яка стандартизована як ECMA-335 Common Language Infrastructure (CLI).

Середовище розробки. Unity — багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.

Є можливість створювати інтернет-додатки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

Технічні характеристики

Сценарії на C#, JavaScript та Boo;

Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі;

- Робота з ресурсами можлива через звичайний Drag&Drop.
- Система успадкування об'єктів;
- Підтримка імпортування великої кількості форматів файлів;
- Вбудований генератор ландшафтів;
- Вбудована підтримка мережі;
- Існує рішення для спільної розробки — Asset Server. Також можна використовувати зручний для користувача спосіб контролю версій. Наприклад, SVN або Source Gear.

Функціональні можливості. Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C #, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) — окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань.

Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти — тобто ті, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково присутній компонент Transform — він зберігає в собі координати місця

розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна — буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Unity Asset Server

Інструментарій для спільної розробки на базі Unity. Сутність розробки ігор — це робота в команді. Сервер ресурсів Unity це доповнення, яке додає контроль версій у функціонал Unity.

Система контролю версій

Сервер ресурсів Unity це повнофункціональне рішення для контролю версій для всіх ігрових скриптів і ресурсів. Як і все інше у Unity, він простий у використанні.

Оптимізація для великих проектів

Багатогігабайтні проекти з тисячами мегабайтних файлів піддаються легкому керуванню. Налаштування імпорту та інші метадані також зберігаються разом з історією їх версій. Переглядати зміни ресурсів\версій можна одразу всередині Редактора Unity. Якщо файли змінюються, їх статус негайно оновлюється.

Перейменування і переміщення ресурсів не створює будь-яких перешкод для безперервного робочого процесу.

Сервер з відкритим вихідним кодом

Сервер ресурсів Unity управляється базою даних PostgreSQL. PostgreSQL відомий своєю надійністю, цілісністю даних і легкістю адміністрування і відмінно справляється з робочим навантаженням гігантських проектів.

Mac OS X або Linux

Сервер ресурсів доступний як для Mac OS X Installer, так і для Linux RPMs. Підтримка декількох платформ дає вам гнучкість у впровадженні Сервера ресурсів Unity у вашу існуючу IT-інфраструктуру.

Середовище програмування. Visual Studio 2022 - найкраща версія Visual Studio на момент написання роботи. Масштабування для роботи над проектами будь-якого розміру та складності у 64-розрядному інтегрованому середовищі розробки. Код із новим редактором Razor, який може виконувати рефакторинг між файлами. Діагностика проблем із візуалізацією асинхронних операцій та застосуванням автоматичних аналізаторів. Розробка кросплатформових мобільних та класичних додатків за допомогою .NET MAUI. Створення швидких веб-інтерфейсів на C# за допомогою Blazor. Складання, налагодження та тестування додатків .NET та C++ у середовищах Linux. Використання можливостей гарячого перезавантаження у програмах .NET та C++. Зміна сторінок ASP.NET у поданні веб-конструктора. Завершення коду з урахуванням штучного інтелекту. Можливість працювати разом у режимі реального часу за допомогою загальних сеансів написання коду. Клонування репозиторіїв, переміщення по робочих елементах та підготування окремих рядків для фіксації. Автоматичне налаштування робочих процесів CI/CD, які можуть розгортатися в Azure.

Візуа́льне програмува́ння — спосіб створення програм шляхом маніпулювання графічними об'єктами замість написання програмного коду в текстовому вигляді.

Візуальне програмування дозволяє програмувати, використовуючи графічні або символьні елементи, якими можна маніпулювати інтерактивним чином згідно з деякими правилами, причому просторове розташування графічних об’єктів використовувати як елементи синтаксису програми. Значна частина візуальних мов програмування базується на ідеї «фігур і ліній», де фігури (прямокутники, овали та ін.) розглядаються як суб'єкти і з'єднуються лініями (стрілками, дугами тощо), які являють собою відношення. Приклад: UML.

Мови візуального програмування можуть бути додатково класифіковані в залежності від типу і ступеня візуального вираження, на типи:

1 Природно-візуальні мови мають невід'ємне візуальне вираження, для якого немає очевидного текстового еквіваленту (наприклад, графічна мова G в середовищі LabVIEW).

2 Візуально-перетворені мови є невізуальними мовами з накладеним візуальним представленням.

Значна кількість сучасних мов програмування має розвинуті візуальні засоби для розробки графічного інтерфейсу, причому здійснюється програмування розміщених на спеціальних формах об'єктів з настроюванням їх властивостей та поведінки. CodeGear Delphi і C++ Builder, Microsoft Visual Studio та мови, які включає в себе цей засіб (Visual Basic, Visual C#, Visual J# тощо) часто плутають з візуальними мовами програмування. Всі ці мови є текстовими, а не візуальними (графічними). MS Visual Studio та Delphi є візуальними середовищами програмування, але не візуальними мовами програмування.

2 РОЗРОБКА ГРИ СИМУЛЯТОРА МОВОЮ C# НА ДВИГУНІ UNITY

2.1 Створення та налагодження програми

Для написання програми була використана мова програмування C# у середовищі програмування Unity. Для свого дипломного проекту я використовував такі компоненти:

Canvas - це область, всередині якої знаходяться всі елементи UI (користувальницького інтерфейсу). Область Canvas відображається у вигляді прямокутника у вікні Scene View. Це полегшує процес розташування елементів UI без необхідності бачити ігрове вікно (Game View).

TextMesh Pro - це заміна існуючих текстових компонентів Unity, таких як Text Mesh і UI Text. TextMesh Pro використовує поле « Signed Distance Field» (SDF) як основний конвеєр рендеринга тексту, що робить можливим відтворення тексту чисто в будь-якому розмірі і роздільній здатності. Використовуючи набір користувальницьких шейдерів, призначених для використання потужності рендеринга тексту SDF, TextMesh Pro дає можливість динамічно змінювати зовнішній вигляд тексту, просто змінюючи властивості матеріалу, щоб додати візуальні стилі, такі як розширення, контур, м'яка тінь, скошування, текстури, світіння і т.д., а також для збереження та відтворення цих візуальних стилів шляхом створення / використання пресетів матеріалу.

Image - імпортується як спрайт UI, натиснувши на Sprite (2D / UI) в секції Time Texture Type. У спрайтів є додаткові параметри імпорту, порівняно зі старими спрайтами GUI, найбільша різниця - наявність редактора спрайтів. Дає можливість розділити зображення на 9 частин, щоб при зміні розміру спрайту його кути не розтягулися і не спотворилися.

Raw Image – елемент управління відображає неінтерактивне зображення для користувача. Це можна використати для оформлення, значків і т. д. А також

можна змінити зображення з сценарію, щоб відобразити зміни в інших елементах управління. Елемент управління аналогічно елементу керування « Image », але не має того, щоб встановити параметри для анімації зображення і точного заповнення прямокутника елемента управління. Однак необроблене зображення може відображати будь-яку текстуру, а зображення може відображати тільки Sprite текстури.

Button - має функцію OnClick UnityEvent, щоб визначити, що буде робити при натисканні. Керування кнопкою відповідає натисканню користувача і використовується для ініціювання або підтвердження дії. Відомі приклади включають кнопки "Відправити" та "Скасувати", що використовуються у веб-формах. Кнопка призначена для ініціювання дії, коли користувач натискає та відпускає її. Якщо мишу пересунути з кнопки керування, перш ніж натиснути на кнопку, дія не відбудеться.

Scroll Rect (Scroll View) - Scroll Rect можна використовувати, коли вміст, який займає багато місця, потрібно відобразити на невеликій ділянці. Scroll Rect надає функціональні можливості для прокручування цього вмісту. Зазвичай Scroll Rect поєднується з Mask, щоб створити подання прокручування, де видно лише прокручуваний вміст всередині Scroll Rect. Його також можна об’єднати з однією або двома смугами прокрутки, які можна перетягувати для горизонтальної або вертикальної прокрутки.

Scrollbar - смуга прокрутки має десяткове число Value від 0 до 1. Коли користувач перетягує смугу прокрутки, значення відповідно змінюється. Scrollbar часто використовуються разом із Scroll Rect і Mask для створення перегляду прокручування. Scrollbar має значення розміру від 0 до 1, яке визначає розмір ручки як частку від усієї довжини смуги прокрутки. Це часто керується іншим компонентом, щоб вказати, наскільки велика частка вмісту в режимі прокручування видима. Компонент Scroll Rect може робити це автоматично. Scrollbar може бути горизонтальним або вертикальним. Він також має

OnValueChanged UnityEvent, щоб визначити, що він робитиме, коли значення буде змінено.

Dropdown - у спадному меню є список опцій на вибір. Для кожного параметра можна вказати текстовий рядок і, за бажанням, зображення, які можна встановити або в інспекторі, або динамічно з коду. Він має OnValueChanged UnityEvent, щоб визначити, що він робитиме, коли поточний вибраний параметр буде змінено.

Slider - повзунок має десяткове число Value, яке користувач може перетягувати між мінімальним і максимальним значенням. Він може бути як горизонтальним, так і вертикальним. Він також має OnValueChanged UnityEvent, щоб визначити, що він робитиме, коли значення буде змінено.

Animator Controllers - контролер анімацій дозволяє налаштовувати і управляти набором анімацій персонажа чи іншого анімованого Game Object (іграбельного об’єкту).

Контролер має посилання на які використовував анімаційні кліпи, і управляє різними анімаційними станами і переходами між ними, використовуючи, так званий State Machine (кінцевий автомат), який може бути представлений у вигляді блок-схеми, або простої програми, написаної на мові візуального програмування в Unity.

Animation Clips - анімаційні кліпи є одним з основних елементів системи анімації Unity. Unity підтримує імпортування анімації з зовнішніх джерел і надає можливість створювати кліпи анімації з нуля в редакторі за допомогою вікна анімації.

2.2 Опис програми та її алгоритмів

Завданням дипломного проекту було створення гри симулятора "Delivery Service". Для її реалізації мені знадобилися такі класи:

1. MonoBehaviour - основний батьківський клас. Його назва напряму йде від програмного середовища "Mono", на якій і працює програмна частина движка Unity3d. Всі скрипти, створювані в Unity3d, за замовчуванням створюються, як дочірні класи від батьківського класу MonoBehaviour.

2. Main містить початок генерації замовлень та слідкування за часом доставки посилки на склад(також графічно відображає залишки часу) для подальшого виклику генерації наступного замовлення, крім того веде рахунок часу проведеного гравцем у грі для ведення статистики та відображає реальний час.

3. LevelController контролює накопичення досвіду для підвищення рівня компанії.

4. Level - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон, що містить змінні для запису необхідних для підвищення рівня компанії значень. З нього складається список модернізації компанії.

5. NewTask генерує номер, тип, вагу, попередження, відправника, отримувача, пакет страхування, адресу, ціну(кількість грошей та досвіду вираховується в залежності від типу посилки, попередження, пакету страхування, рівню компанії та часу, виділеного на доставку отримувачу), час відведений на доставку замовнику та час відведений на доставку посилки на склад. Додає посилку на склад у відповідності з типом попередження. Містить виклики запуску анімації при оновленні замовлення та закінчення таймера на доставку посилки на склад. Також підраховує кількість отриманих замовлень за типом, попередженням та пакетом страхування для ведення статистики.

6. CreateRndmName випадково генерує прізвища та імена відправника та отримувача.

7. CountMoney розраховує нагороду за доставку в залежності від часу доставки. Підраховує зароблені та витрачені гроші та досвід, кількість успішних

та провалених доставок для ведення статистики. Містить виклики анімації отримання або списання грошей.

8. TaskStatistic веде статистику отриманих на склад посилок різних типів, попереджень та пакетів страхувань. Також успішних або провалених(якщо не доставлено вчасно) замовлень, зароблених та витрачених на оплату штрафу(за невчасну доставку) грошей та досвіду. Крім того анімацію відповідного рядка при збільшенні числа.

9. Achievements на початку створює список досягнень з назвою, картинкою, текстом завдання, поточним та необхідними значеннями прогресу(для отримання нагороди) та кількісними значеннями нагороди за виконання кожного рівня(кожне досягнення має 5 рівнів) досягнення. Контролює відображення поточного процесу виконання досягнення та сигналізує про виконання. Оновлює досягнення після отримання нагороди за його виконання.

10. AchievCell - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон досягнення з усіма необхідними змінними для запису інформації. З нього складається список досягнень.

11. Addresses на початку створює список адрес з координатами та назвою, потім створює для кожної адреси маркер, який завантажує з ресурсів, передає йому відповідні коодинати та адресу, тим самим розміщуючи його в потрібному місці. Крім того відповідає за активацію відповідних адресних маркерів у разі появи на складі посилки адресованої за конкретним адресом.

12. LogCellController створює список виконаних замовлень(журнал доставки) в якому зберігається основна інформація про виконане замовлення, залишки часу доставки(або перевищений час доставки) та винагорода в залежності від часу доставки.

13. LogCell - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон запису журналу доставки з усіма необхідними змінними для запису інформації. З нього складається журнал доставки.

14. Address створює список адрес. В кожної адреси він свій. В ньому знаходяться змінні для координат та назви, які потім передаються маркеру.

15. AddressMarker прив’язаний до маркера адреси. В кожного маркера він свій. В ньому знаходиться анімація адреси(при наведені на маркер над нам з’являється панель з адресою, якщо прибрати курсор з маркера то панель зникає) та відкриття панелі відправки при натисканні на маркер(таким чином адреса відправки вже обрана, залишається перетягнути зі складу відповідну посилку у комірку на панелі та обрати якою автівкою буде доставлятись замовлення).

16. StorageController, ThermalStorageController, CRSStorageController, EISStorageController, ERSStorageController, BISStorageController, RISStorageController – на початку створюють списки порожніх комірок для посилок кожен для свого типу складу у відповідності з типами попереджень, списки розблокування та модифікацій складу. Містять функції оновлення вмісту, модифікації складу та відстеження появи на складі нових посилок(посилка вважається новою, якщо гравець не переглядав інформацію про неї, навівши курсор на відповідну комірку на складі) і відображення відповідних поміток в кожному зі складів окремо.

17. StorageUpgrade - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон, що містить змінні для запису необхідних для розблокування та модернізації складу значень. З нього складається список модернізації складу.

18. UnCheckedStorageController відстежує чи є на якому із складів нова посилка та відображає відповідну помітку.

19. Buttons – містить функції для відображення відповідних елементів інтерфейсу(панелі замовлень, статистики, досягнень, журналу доставки, загальної панелі складу та кожного складу окремо, панелі автомобілів, панелі відправки, панелі рівня компанії), паузи та зміни швидкості таймера, що відповідає за час доставки посилки на склад.

20. Package – клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон посилки з усіма необхідними змінними для запису інформації.

21. CellButton прив’язаний до комірки складу. У кожної свій. Містить змінні для збереження власного коду та коду посилки. Містить функцію для відображення панелі з інформацією про посилку, що викликається при наведені на комірку, переміщення посилки зі списку складу в комірку панелі відправлення та для розрахунку й відображення залишку часу для доставки.

22. SendPanel на початку створює список із 4 порожніх комірок для посилок. Залишає доступними в залежності від кількості доступних комірок поточного обраного автомобіля. В ці комірки переміщується посилка зі списку складу. Містить функції повернення посилок на склад, відправлення в автомобіль, вибір автомобілів та для розрахунку й відображення залишку часу для доставки.

23. CarController на початку створює список автомобілів та їх характеристик. Містить функції для оновлення інформації про поточний рівень, статус та характеристики, рівень топлива та можливість його купівлі, поточний та максимальний рівень завантаження, виконання поточного замовлення. Також модифікації автомобіля.

24. Car - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон автомобіля з усіма необхідними змінними для запису інформації. З нього складається список автомобілів.

25. CarCell прив’язаний до комірки автомобіля. У кожного свій. Містить функції для розрахунку залишку часу виділеного на доставку та часу за який автомобіль доставляє замовлення, поточного завантаження та перемикання статусу авто.

26. Menu – викликає анімацію відкриття меню та запускає таймер для відображення реального часу.

27. MenuButtons – містить функції для кнопок меню (почати, продовжити, налаштування та вихід).

28. SpawnLogo – створює на головному меню об’єкт типу Image в випадковому місці.

29. LogoAnimation – здійснює анімацію створеного на головному меню об’єкта.

Алгоритм - набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій; система правил виконання дискретного процесу, яка досягає поставленої мети за скінченний час. Для візуалізації алгоритмів часто використовують блок-схеми.

Для комп'ютерних програм алгоритм є списком деталізованих інструкцій, що реалізують процес обчислення, який, починаючи з початкового стану, відбувається через послідовність логічних станів, яка завершується кінцевим станом. Перехід з попереднього до наступного стану не обов'язково детермінований - деякі алгоритми можуть містити елементи випадковості.

На рисунку 2.1 блок-схема алгоритму програми «Delivery Service». Все починається з завантаження головного меню на якому розташовані чотири кнопки.

При натисканні на кнопку «Start» програма завантажить сцену із ігровим полем та інтерфейсом користувача, після чого гравець може грати. В інтерфейсі користувача є шість кнопок: натиснувши на кнопку «Task», відкривається панель замовлень та чотири кнопки для перемикання між «підпанелями»: замовлень, статистики, досягнень та журналу доставок; натиснувши на кнопку «Storage», відкривається панель складів та сім кнопок перемикання між складами; натиснувши на кнопку «Car», відкривається панель автомобілів; натиснувши на кнопку з подвійною стрілкою, що дивиться вниз, відкриється панель рівня компанії;натиснувши на кнопку з трьома горизонтальними рисками гра стане на

паузу і відкриється меню паузи, з якого можна повернутись до гри або вийти в головне меню; натиснувши на кнопку «x1», змінюється швидкість таймеру.

Кнопка «Quit» головного меню завершує програму.

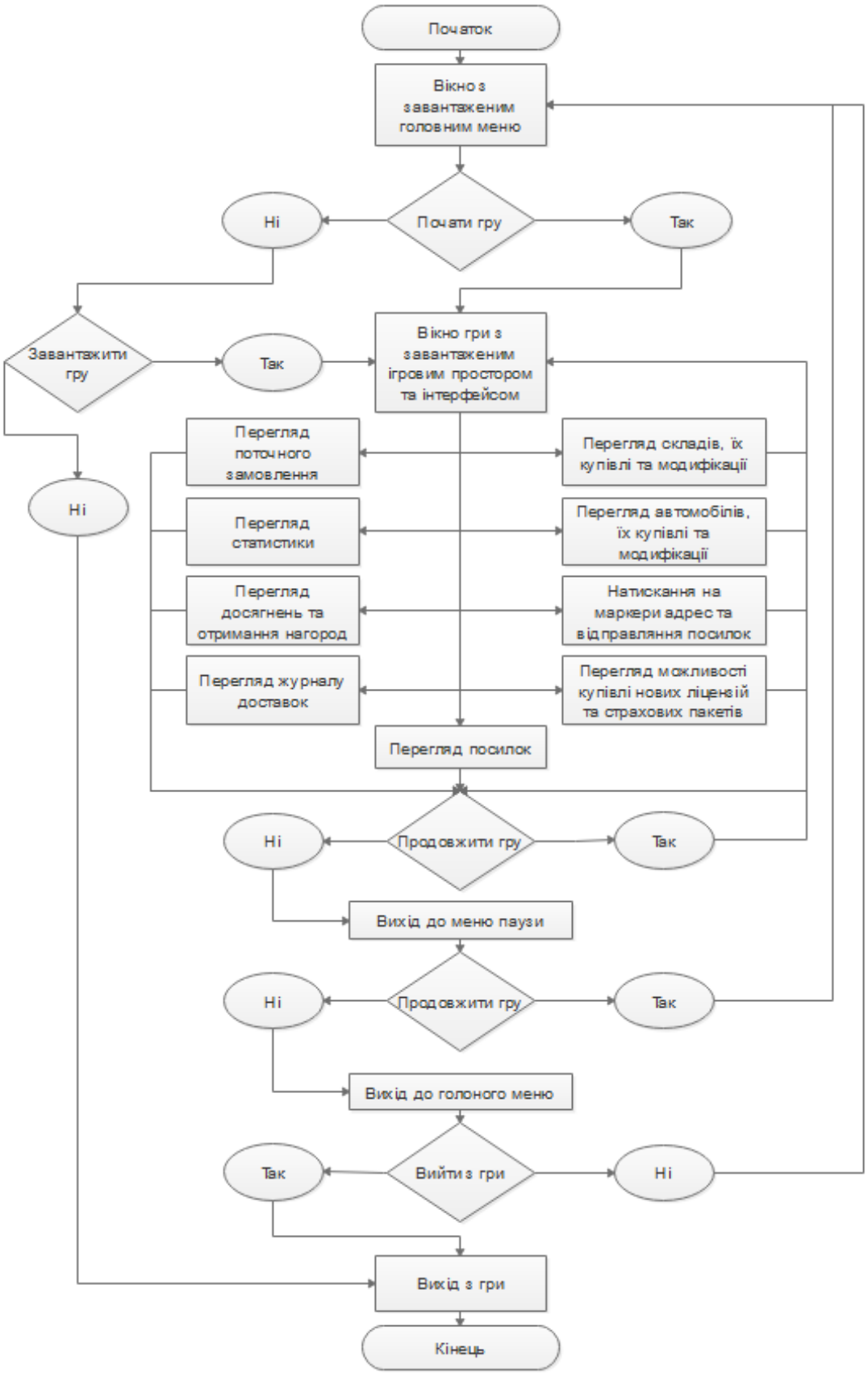


Рисунок 2.1 – Алгоритм програми

Діаграма класів - це набір статичних, декларативних елементів моделі. Діаграми класів можуть застосовуватися і при прямому проектуванні, тобто в процесі розробки нової системи, і при зворотному проектуванні - описі існуючих і використовуваних систем. Діаграма класів є ключовим елементом в об'єктно-орієнтованому моделюванні. Графічно клас зображується у вигляді прямокутника, розділеного на 3 блоки горизонтальними лініями, що містять:

– ім'я класу;
– атрибути (властивості) класу;
– операції (методи) класу.

Для атрибутів і операцій може бути вказаний один з трьох типів видимості:

– private (приватний)
– protected (захищений)
– public (загальний)

У верхній частині написано ім'я класу. Посередині розташовуються атрибути класу. Нижня частина містить операції класу. Для побудови діаграми класів я вибрав основні класі, дивіться рисунок 2.2.

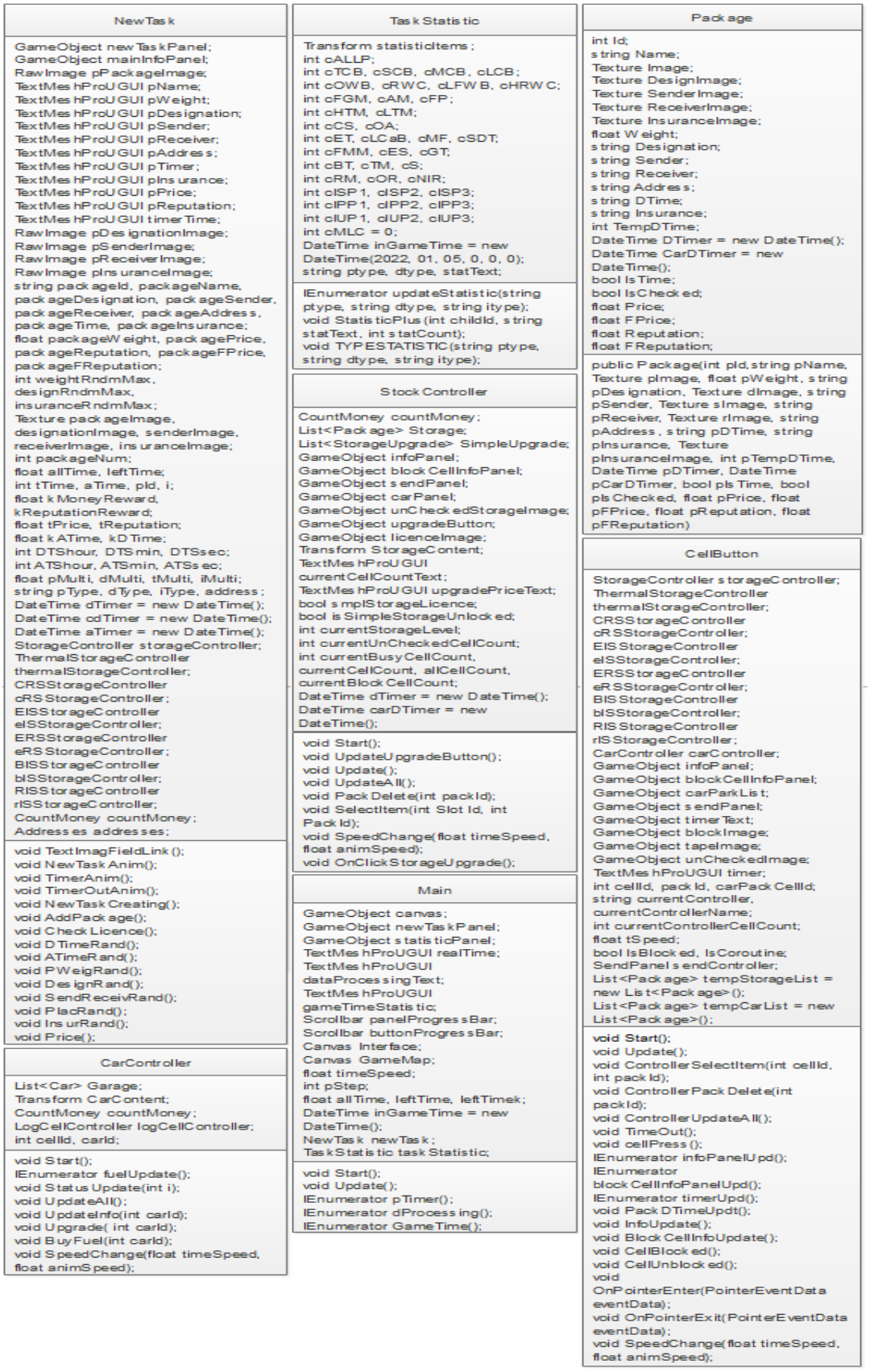


Рисунок 2.2 – Діаграма класів

Розглянемо атрибути і методи основного класа NewTask.

Атрибути:

public GameObject newTaskPanel – містить посилання на панель поточного замовлення.

public GameObject mainInfoPanel - містить посилання на панель, яка містить основну інформацію про замовлення.

private RawImage pPackageImage – містить посилання на об’єкт, який відображає зображення посилки нового замовлення.

public TextMeshProUGUI pName - містить посилання на об’єкт, який відображає назву посилки.

public TextMeshProUGUI pWeight - містить посилання на об’єкт, який відображає вагу посилки.

private TextMeshProUGUI pDesignation - містить посилання на об’єкт, який відображає тип попередження.

public TextMeshProUGUI pSender - містить посилання на об’єкт, який відображає відправника посилки.

public TextMeshProUGUI pReceiver - містить посилання на об’єкт, який відображає отримувача посилки.

public TextMeshProUGUI pAddress - містить посилання на об’єкт, який відображає адресу доставки посилки.

public TextMeshProUGUI pTimer - містить посилання на об’єкт, який відображає час доставки посилки.

private TextMeshProUGUI pInsurance - містить посилання на об’єкт, який відображає пакет страхування.

public TextMeshProUGUI pPrice - містить посилання на об’єкт, який відображає гроші за доставку посилки.

public TextMeshProUGUI pReputation - містить посилання на об’єкт, який відображає досвід за доставку посилки.

public TextMeshProUGUI timerTime містить посилання на об’єкт, який відображає текстом скільки часу залишилося до доставки посилки на склад.

private RawImage pDesignationImage – містить посилання на об’єкт, який відображає зображення типу попередження.

private RawImage pSenderImage – містить посилання на об’єкт, який відображає зображення відправника посилки.

private RawImage pReceiverImage – містить посилання на об’єкт, який відображає зображення отримувача посилки.

private RawImage pInsuranceImage – містить посилання на об’єкт, який відображає зображення пакета страхування.

public string packageId, packageName, packageDesignation, packageSender, packageReceiver, packageAddress, packageTime, packageInsurance – у ці змінні записуються згенеровані номер, назва, попередження, відправник, отримувач, адреса, час доставки та пакет страхування.

private float packageWeight, packagePrice, packageReputation, packageFPrice, packageFReputation - у ці змінні записуються згенеровані вага, гроші та досвід за доставку.

public int weightRndmMax, designRndmMax, insuranceRndmMax – ці змінні містять максимальні значення для Random генерування ваги, типу попередження та пакету страхування. Вони збільшуються при розблокуванні нових складів та пакетів страхування.

private Texture packageImage, designationImage, senderImage, receiverImage, insuranceImage - містять шляхи до файлів відповідних картинок згенерованих посилки, попередження, відправника, отримувача та пакету страхування в залежності від згенерованого типу.

private int packageNum – містить код згенерованої посилки.

private float allTime, leftTime – містять значення початково виділеного та залишки часу на доставку посилки на склад відповідно. Необхідні для графічного відображення залишку часу на доставку посилки на склад.

private int tTime – містить початково виділену на доставку кількість часу.

private int aTime – змінна необхідна для розрахунку згенерованого на доставку посилки на склад часу.

private int pId – змінна необхідна для запису інформації про нову посилку у список відповідного складу.

private int i – у змінній підраховується загальна кількість згенерованих посилок.

public float kMoneyReward, kReputationReward – змінні містять коефіцієнти грошей та досвіду для розрахунку нагороди при генерації посилки. Збільшуються при підвищенні рівня компанії.

public float tPrice, tReputation – змінні містять згенеровані значення нагороди(гроші та досвід відповідно) за доставку посилки.

public float kATime, kDTime – змінні містять коефіцієнти часу відправки на склад та доставки посилки відповідно. Використовуються у генерації цих значень. При підвищенні рівня компанії коефіцієнт часу відправки на склад зменшується, а коефіцієнт часу доставки посилки збільшується.

public int DTShour, DTSmin, DTSsec – містять згенерований час доставки посилки в годинах, хвилинах та секундах відповідно.

public int ATShour, ATSmin, ATSsec містять згенерований час доставки посилки на склад в годинах, хвилинах та секундах відповідно.

public float pMulti, dMulti, tMulti, iMulti – змінні містять коефіцієнти для розрахунку нагороди(однакові як для грошей так і для досвіду) в залежності від типу посилки, попередження, виділеного на доставку часу та пакету страхування.

public string pType, dType, iType – у змінні записуються коди згенерованих типів посилки, попередження та пакету страхування для подальшого підрахунку статистики.

public DateTime dTimer, aTimer, cdTimer – у змінні записується згенерований на доставку, відправку на склад та безпосередню доставку автомобілем посилки. Працюють у вигляді таймера відраховуючи час.

private StorageController storageController, ThermalStorageController thermalStorageController, CRSStorageController cRSStorageController, EISStorageController eISStorageController, ERSStorageController eRSStorageController, BISStorageController bISStorageController, RISStorageController rISStorageController – містять посилання на класи конкретних складів для подальшого «перевезення» до них нових згенерованих посилок.

public CountMoney countMoney – містить посилання на клас, в якому міститься коефіцієнт(використовується у розрахунку нагороди при генерації нової посилки), що був створений для пришвидшення тестування.

public Addresses addresses – містить посилання на клас адрес, з якого викликає функцію активації маркера адреси у відповідності зі згенерованою посилкою.

Операції:

void TextImagFieldLink() – записує у змінні pName, pWeight, pDesignation, pSender, pReceiver, pAddress, pTimer, pInsurance, pPrice, pReputation, pCustomerImage, pPackageImage, pDesignationImage, pSenderImage, pReceiverImage, pInsuranceImage посилання на відповідні об’єкти для відображення.

void NewTaskAnim() – викликає анімацію зміни поточного замовлення.

void TimerAnim() - викликає анімацію таймера коли час закінчився.

void TimerOutAnim() – викликає анімацію таймера коли до кінця часу залишається десять секунд.

void NewTaskCreating() – виклик функції перевірки ліцензії та всіх функцій генерації посилки.

void AddPackage() – «відправляє» посилку на відповідний склад після закінчення часу виділеного на це.

void CheckLicence() – перевіряє розблоковані склади, збільшує значення designRndmMax(збільшення дозволяє генерувати посилки з новими типами попередження), якщо гравець розблокував новий склад.

void DTimeRand() – генерація випадкового часу доставки посилки.

void ATimeRand() - генерація випадкового часу доставки посилки на склад.

void PWeigRand() – генерація випадкового типу посилки, від якого залежать її вага(випадково генерується в діапазоні залежному від типу) та картинка.

void DesignRand() – генерація випадкового типу попередження.

void SendReceivRand() – генерація випадкового відправника та отримувача посилки та їх картинки.

void PlacRand() – генерація випадкової адреси.

void InsurRand() – генерація випадкового пакету страхування.

void Price() – підрахунок ціни доставки(грошей та досвіду) на основі типу посилки, попередження, пакету страхування, рівня компанії та часу виділеного на її доставку.

Діаграма прецедентів в UML - діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

На діаграмі прецедентів (див. рис. 2.3 і 2.4) можна побачити такі елементи:

- Актор - чоловічок, що позначає набір ролей користувача (розуміється в широкому сенсі: людина, зовнішня сутність, клас, інша система), що взаємодіє з

деякою сутністю (системою, підсистемою, класом). Актори не можуть бути пов'язані один з одним (за винятком відносин узагальнення / успадкування).

- Прецедент - еліпс з написом, що позначає виконувані системою дії (можуть включати можливі варіанти), що призводять до спостережуваних акторами результатами. Ім'я прецеденту пов'язано з неперервним сценарієм - конкретної послідовністю дій, що ілюструє поведінку.

На діаграмі прецедентів ми можемо бачити, як користувач може користуватися системою.

Рисунок 2.3 – Діаграма прецедентів (User)

Рисунок 2.4 – Діаграма прецедентів (Core)

Згідно діаграми-прецедентів програма має два актори, а саме "Ядро програми" та "Користувача".

За допомогою запитів до ядра операційної системи, ядро програми виконує такі функції:

– генерує ігровий простір;
– генерує замовлення посилок;
– переміщує посилок на склади;
– слідкує за часом;
– слідкує за виконанням досягнень;
– введе статистику;
– введе журнал доставки;
– слідкує за статусом авто;
– слідкує за паливом авто;

– слідкує за накопиченням досвіду;
– підвищує рівень компанії;
– розраховує нагороди за замовлення в залежності від часу доставки;
– запускає анімацію;
– генерує відмови.

В свою чергу користувач має наступний набір функцій:

– переглядає нові замовлення;
– переглядає список посилок на складах;
– переглядає статистику;
– переглядає досягнення;
– отримує нагороди за досягнення;
– переглядає виконані замовлення;
– переглядає список авто;
– купує нові склади;
– модифікує склади;
– купує додаткові авто;
– модифікує авто;
– поповнює паливо авто;
– розблоковує нові склади та пакети страхування;
– вибирає адресу;
– вибирає замовлення;
– вибирає авто.

Діаграма компонентів - діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та

компоненти, що можуть виконуватись. Модуль програмного забезпечення може бути представлено як компонент. Деякі компоненти існують під час компіляції, деякі - під час компонування, а деякі під час роботи програми. Діаграма компонент відображає лише структурні характеристики.

Компонент призначений для зображення фізичної організації асоційованих з ним елементів моделі. Додатково компонент може мати текстовий стереотип і помічені значення, а деякі компоненти - власне графічне зображення.

На рисунку 2.5 діаграма компонентів, яка складається з таких компонентів:

MonoBehaviour - основний батьківський клас. Його назва напряму йде від програмного середовища "Mono", на якій і працює програмна частина движка Unity3d. Всі скрипти, створювані в Unity3d, за замовчуванням створюються, як дочірні класи від батьківського класу MonoBehaviour. На цьому класі тримаються всі інші компоненти.

Main – компонент, який містить початок генерації замовлень та слідкування за часом доставки посилки на склад(також графічно відображає залишки часу) для подальшого виклику генерації наступного замовлення, крім того веде рахунок часу проведеного гравцем у грі для ведення статистики та відображає реальний час.

LevelController – компонент, який контролює накопичення досвіду для підвищення рівня компанії.

Level - компонент, який є основою списку підвищення рівня компанії. Являє собою порожній шаблон, що містить змінні для запису необхідних для підвищення рівня компанії значень.

NewTask – компонент, який генерує номер, тип, вагу, попередження, відправника, отримувача, пакет страхування, адресу, ціну(кількість грошей та досвіду вираховується в залежності від типу посилки, попередження, пакету страхування, рівню компанії та часу, виділеного на доставку отримувачу), час відведений на доставку замовнику та час відведений на доставку посилки на

склад. Додає посилку на склад у відповідності з типом попередження. Містить виклики запуску анімації при оновленні замовлення та закінчення таймера на доставку посилки на склад. Також підраховує кількість отриманих замовлень за типом, попередженням та пакетом страхування для ведення статистики.

CreateRndmName – компонент, який випадково генерує прізвища та імена відправника та отримувача.

CountMoney – компонент, який розраховує нагороду за доставку в залежності від часу доставки. Підраховує зароблені та витрачені гроші та досвід, кількість успішних та провалених доставок для ведення статистики. Містить виклики анімації отримання або списання грошей.

TaskStatistic – компонент, який веде статистику отриманих на склад посилок різних типів, попереджень та пакетів страхувань. Також успішних або провалених(якщо не доставлено вчасно) замовлень, зароблених та витрачених на оплату штрафу(за невчасну доставку) грошей та досвіду. Крім того анімацію відповідного рядка при збільшенні числа.

Achievements – компонент, який на початку створює список досягнень з назвою, картинкою, текстом завдання, поточним та необхідними значеннями прогресу(для отримання нагороди) та кількісними значеннями нагороди за виконання кожного рівня(кожне досягнення має 5 рівнів) досягнення. Контролює відображення поточного процесу виконання досягнення та сигналізує про виконання. Оновлює досягнення після отримання нагороди за його виконання.

AchievCell – компонент, який є основою списку досягнень. Являє собою порожній шаблон досягнення з усіма необхідними змінними для запису інформації.

Addresses – компонент, який на початку створює список адрес з координатами та назвою, потім створює для кожної адреси маркер, який завантажує з ресурсів, передає йому відповідні коодинати та адресу, тим самим розміщуючи його в потрібному місці. Крім того відповідає за активацію

відповідних адресних маркерів у разі появи на складі посилки адресованої за конкретним адресом.

LogCellController – компонент, який створює список виконаних замовлень(журнал доставки) в якому зберігається основна інформація про виконане замовлення, залишки часу доставки(або перевищений час доставки) та винагорода в залежності від часу доставки.

LogCell – компонент, який є основою журналу доставки. Являє собою порожній шаблон запису журналу доставки з усіма необхідними змінними для запису інформації.

Address – компонент, який створює список адрес. В кожної адреси він свій. В ньому знаходяться змінні для координат та назви, які потім передаються маркеру.

AddressMarker – компонент, який прив’язаний до маркера адреси. В кожного маркера він свій. В ньому знаходиться анімація адреси(при наведені на маркер над нам з’являється панель з адресою, якщо прибрати курсор з маркера то панель зникає) та відкриття панелі відправки при натисканні на маркер(таким чином адреса відправки вже обрана, залишається перетягнути зі складу відповідну посилку у комірку на панелі та обрати якою автівкою буде доставлятись замовлення).

StorageController, ThermalStorageController, CRSStorageController, EISStorageController, ERSStorageController, BISStorageController, RISStorageController – компоненти, які на початку створюють списки порожніх комірок для посилок кожен для свого типу складу у відповідності з типами попереджень, списки розблокування та модифікацій складу. Містять функції оновлення вмісту, модифікації складу та відстеження появи на складі нових посилок(посилка вважається новою, якщо гравець не переглядав інформацію про неї, навівши курсор на відповідну комірку на складі) і відображення відповідних поміток в кожному зі складів окремо.

StorageUpgrade - клас, що не наслідується від MonoBehaviour і є Serializable. Являє собою порожній шаблон, що містить змінні для запису необхідних для розблокування та модернізації складу значень. З нього складається список модернізації складу.

UnCheckedStorageController – компонент, який відстежує чи є на якому із складів нова посилка та відображає відповідну помітку.

Buttons – компонент, який містить функції для відображення відповідних елементів інтерфейсу(панелі замовлень, статистики, досягнень, журналу доставки, загальної панелі складу та кожного складу окремо, панелі автомобілів, панелі відправки, панелі рівня компанії), паузи та зміни швидкості таймера, що відповідає за час доставки посилки на склад.

Package – компонент, який є основою списку посилок, що знаходяться на складах. Являє собою порожній шаблон посилки з усіма необхідними змінними для запису інформації.

CellButton – компонент, який прив’язаний до комірки складу. У кожної свій. Містить змінні для збереження власного коду та коду посилки. Містить функцію для відображення панелі з інформацією про посилку, що викликається при наведені на комірку, переміщення посилки зі списку складу в комірку панелі відправлення та для розрахунку й відображення залишку часу для доставки.

SendPanel – компонент, який на початку створює список із 4 порожніх комірок для посилок. Залишає доступними в залежності від кількості доступних комірок поточного обраного автомобіля. В ці комірки переміщується посилка зі списку складу. Містить функції повернення посилок на склад, відправлення в автомобіль, вибір автомобілів та для розрахунку й відображення залишку часу для доставки.

CarController – компонент, який на початку створює список автомобілів та їх характеристик. Містить функції для оновлення інформації про поточний рівень, статус та характеристики, рівень топлива та можливість його купівлі, поточний та

максимальний рівень завантаження, виконання поточного замовлення. Також модифікації автомобіля.

Car – компонент, який є основою списку автомобілів. Являє собою порожній шаблон автомобіля з усіма необхідними змінними для запису інформації.

CarCell – компонент, який прив’язаний до комірки автомобіля. У кожного свій. Містить функції для розрахунку залишку часу виділеного на доставку та часу за який автомобіль доставляє замовлення, поточного завантаження та перемикання статусу авто.

Menu – компонент, який викликає анімацію відкриття меню та запускає таймер для відображення реального часу.

MenuButtons – компонент, який містить функції для кнопок меню (почати, продовжити, налаштування та вихід).

SpawnLogo – компонент, який створює на головному меню об’єкт типу Image в випадковому місці.

LogoAnimation – компонент, який здійснює анімацію створеного на головному меню об’єкта.

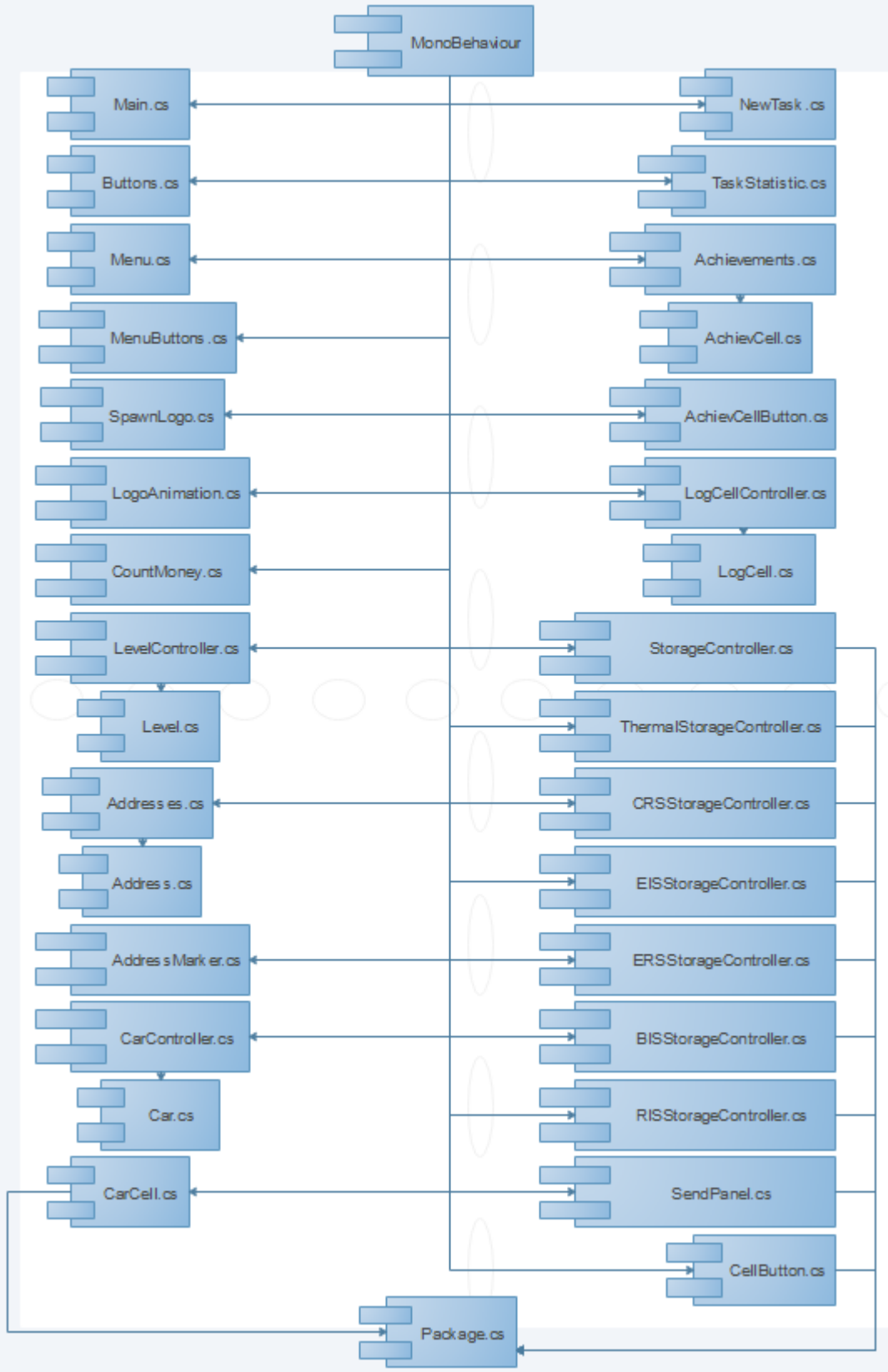


Рисунок 2.5 – Діаграма компонентів

3 ТЕСТУВАННЯ ГРИ СИМУЛЯТОРА

Для можливості зіграти потрібен комп'ютер з операційною системою Windows 7 та вище, оперативною пам’яттю - 190 мб, процесором: Intel Pentium x86 і вище, частотою від 300 mHz, миша, клавіатура.

Для запуску гри потрібно перейти до папки з її пакетом натиснувши ЛКМ на файлі Delivery Service.exe.

Після цього відкривається вікно гри з завантаженим головним меню, де гравцю дають вибір - одразу грати, змінити налаштування, вийти. Меню показано на рисунку 3.1.

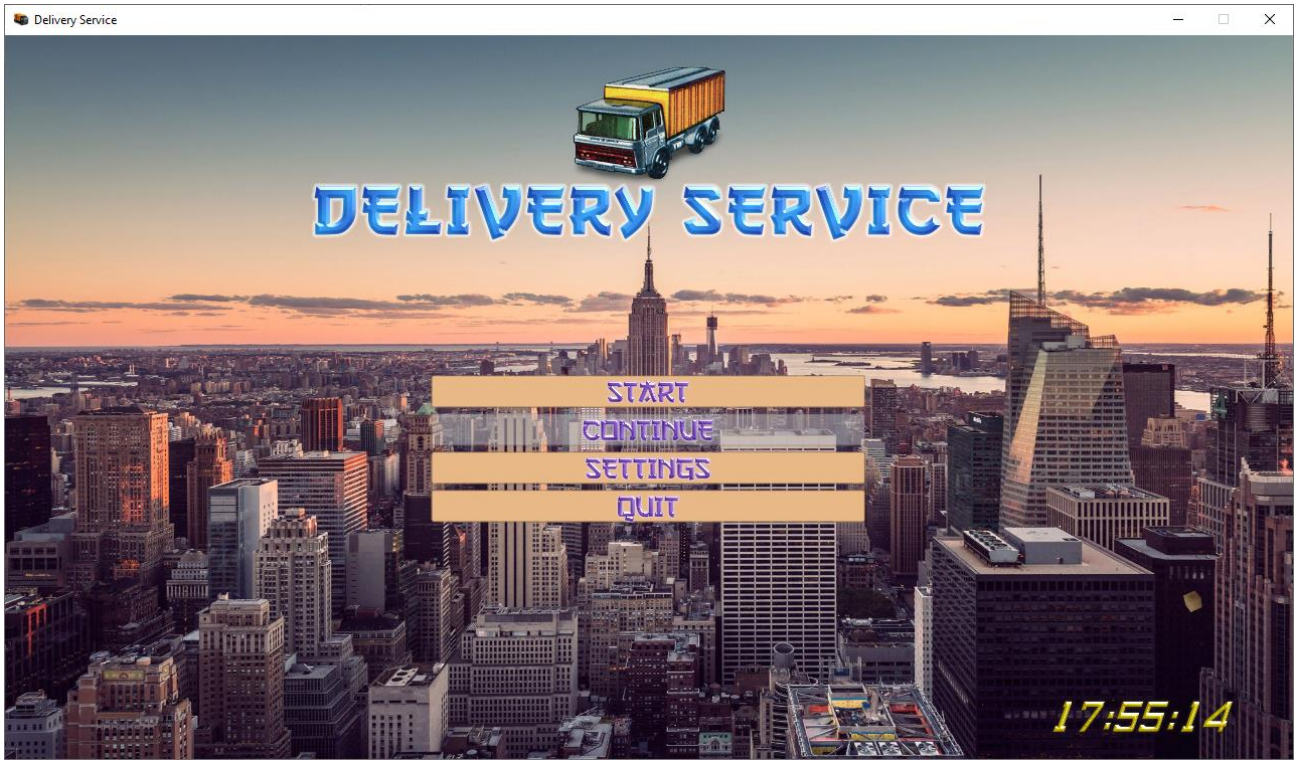


Рисунок 3.1 – Головне меню

Натиснувши кнопку Settings відкривається невелика панель налаштувань, яка зображена на рисунку 3.2. В ній можливо змінити фонову картинку, обрати повноекранний чи віконний режим та змінити роздільну здатність графіки гри.

Налаштування музики не працюють оскільки у грі відсутня музика. Рядок зроблений на майбутнє.

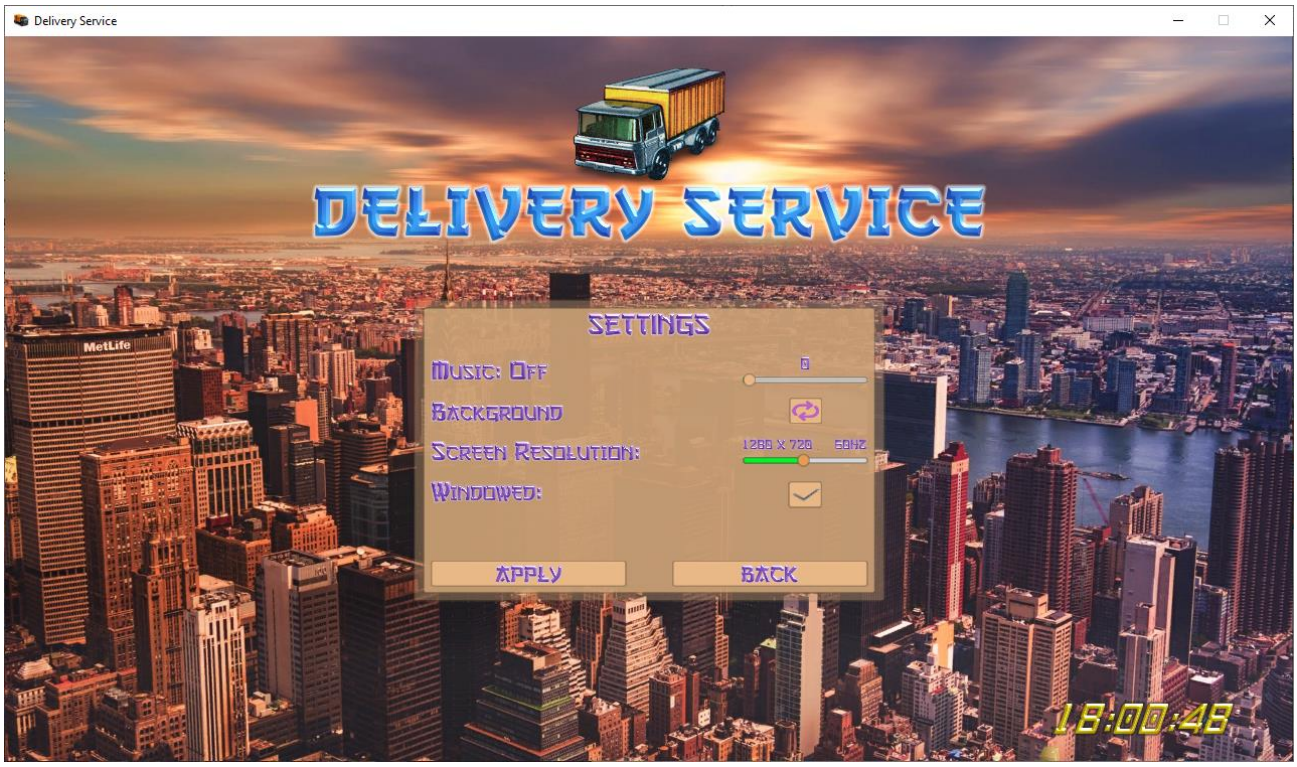


Рисунок 3.2 – Налаштування гри

Натиснувши кнопку Start програма завантажує ігрове поле з інтерфейсом, яке зображене на рисунку 3.3. В інтерфейс користувача входять: кнопка паузи, кнопка швидкості, кнопка замовлень, кнопка складу, кнопка авто, кнопка рівня, панель рівня компанії, панель з кількістю грошей та реальний час.

Рисунок 3.3 – Ігрове поле з Інтерфейсом користувача.

Рисунок 3.4 – Панель нових замовлень.

Натиснувши кнопку Task з’являється панель на якій можна побачити поточне нове замовлення, як показано на рисунку 3.4. Поточне замовлення відображається у вигляді переліку інформації про нього. Для перемикання між панелями нового замовлення, статистики, списку досягнень та журналу доставки необхідно перемикатись кнопками, що знаходяться на панелі збоку, як показано на рисунку 3.5.

Рисунок 3.5 – Кнопки перемикання панелей.

Натиснувши на кнопку Order statistic відкривається панель статистики. Список на панелі статистики розділений на категорії, які показано на рисунку 3.6.

Рисунок 3.6 – Панель статистики.

Перша категорія – кількість отриманих посилок взагалі та кожного типу. Друга категорія – кількість отриманих посилок за типами попередження. Третя категорія - кількість отриманих посилок за типами пакетів страхування. Четверта

категорія – загальна фінансова статистика(успішні і провалені доставки, зароблені і витрачені на штрафи гроші і досвід). В п’ятій категорії лише один рядок – проведений гравцем час у грі.

Натиснувши на кнопку Achievements відкривається панель досягнень, як показано на рисунках 3.7, 3.8, 3.9. У списку досягнень на даний момент знаходиться 46 досягнень, кожне має 5 рівнів виконання(відображається у вигляді зірок).

Рисунок 3.7 – Панель досягнень.

Рисунок 3.8 – Панель досягнень.

Рисунок 3.9 – Панель досягнень.

Якщо досягнення виконано з’являються помітки для звертання уваги гравця у верхньому правому куті кнопки отримання нагороди досягнення, кнопці перемикання на панель досягнень Achievements та кнопки відкриття загальної панелі Task, як показано на рисунку 3.10.

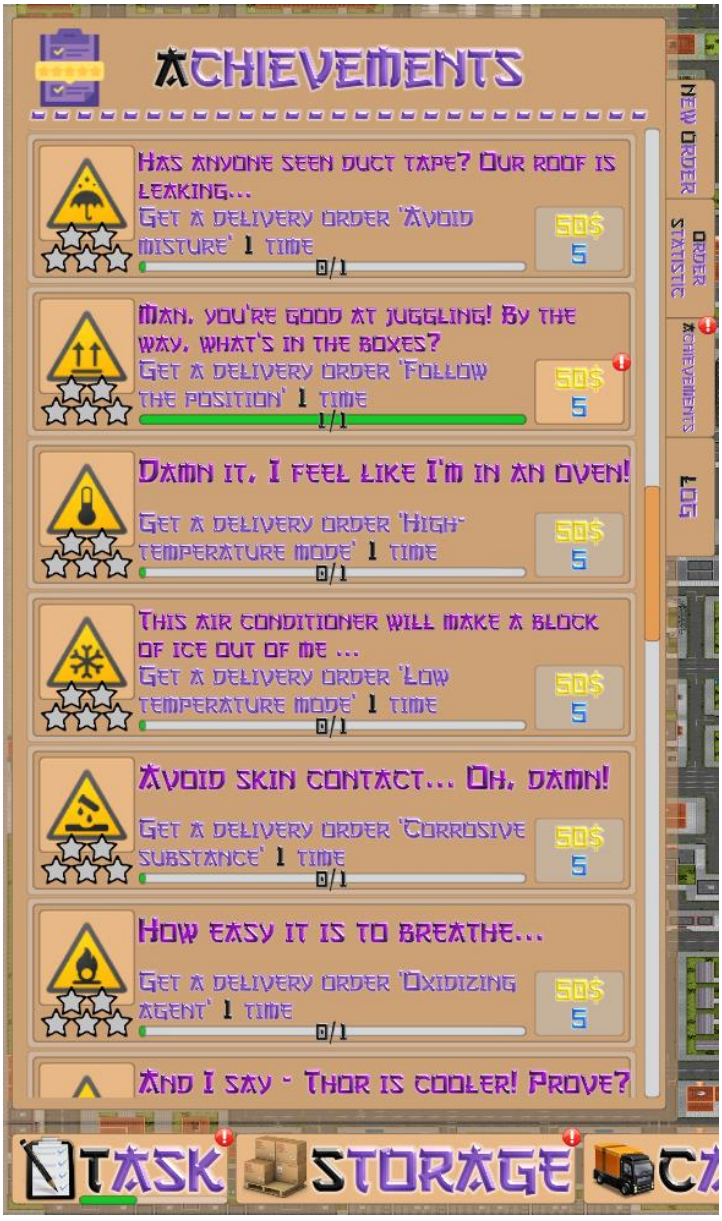


Рисунок 3.10 – Мітки виконаного досягнення на панелі досягнень.

Після того як гравець забере нагороду за досягнення, воно перейде на наступний рівень, зникне мітка на кнопці нагороди, як показано на рисунку 3.11.

Мітки на кнопках Achievements та Task зникнуть якщо не залишиться ні одного виконаного на той момент досягнення.

Рисунок 3.11 – Виконане досягнення на панелі досягнень.

Натиснувши на кнопку Log відкривається панель журналу доставки, як показано на рисунку 3.12. На початку, коли не завершено ще ні одного замовлення, там є текс, який пояснює для чого ця панель. Пізніше, після завершення замовлень, будуть відображатись результати виконаних доставок.

Рисунок 3.12 – Панель журналу доставки.

Натиснувши на кнопку Storage відкривається панель складів, як показано на рисунку 3.13. На ній знаходиться сім кнопок перемикання між різними складами, як показано на рисунку 3.14. Візуально вони однакові, але кожний призначений для окремої категорії посилок. На початку доступний звичайний склад, на якому можуть знаходитись посилки з попередженями: «Ніякого», «Крихкий матеріал», «Уникайте вологи» та «Дотримуйтесь положення». На Thermal Storage можуть знаходитись посилки з попередженями: «Високотемпературний режим» та «Низькотемпературний режим». На Corrosion-resistant Storage можуть знаходитись посилки з попередженями: «Корозійна речовина» та «Окислювач». На Electrically insulated Storage можуть знаходитись посилки з попередженями: «Загроза

електроудару», «Літієві елементи/батареї», «Магнітні поля» та «Загроза статичного розряду». На Explosion-resistant Storage можуть знаходитись посилки з попередженями: «Займистий матеріал», «Вибухова речовина» та «Газова загроза». На Biologically isolated Storage можуть знаходитись посилки з попередженями: «Біологічна загроза», «Токсичний матеріал» та «Подразник». На Radiation isolated Storage можуть знаходитись посилки з попередженями: «Радіоактивний матеріал», «Оптичне випромінювання» та «Неіонізуюче випромінювання». Розблокування та модифікація складу можлива лише при наявності ліцензії для кожного складу окремо, як показано на рисунку 3.15.

Рисунок 3.13 – Панель складів.

Рисунок 3.14 – Панель кнопок перемикання складів.

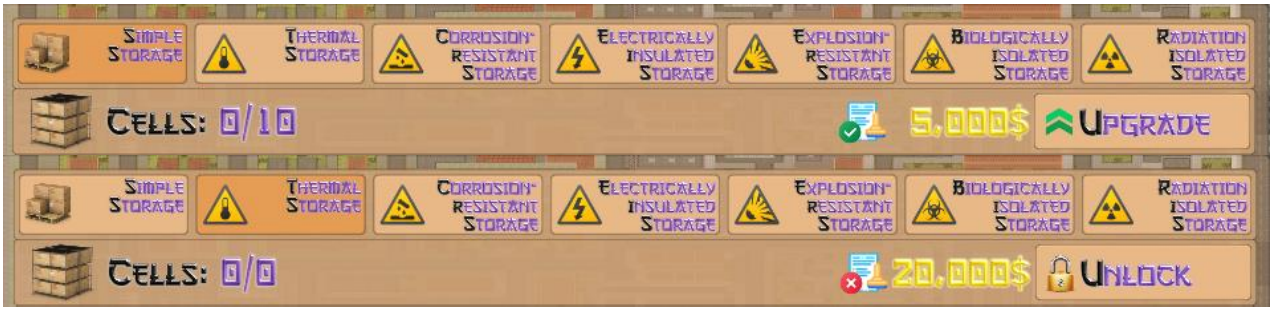


Рисунок 3.15 – Розблокування та модифікація складу.

Якщо на склад надходить нова посилку інформацію про яку гравець не переглядав, то комірка з цією посилкою, а також кнопка перемикання складу на якому вона знаходиться і загальна кнопка панелі складу помічається попереджувальною міткою, як показано на рисунку 3.16.

Рисунок 3.16 – Мітка попередження про нову посилку на складі.

На складі інформацію про наявні посилки можливо переглянути після наведення курсору на відповідну комірку, після чого з’являється панель з інформацією, як показано на рисунку 3.17.

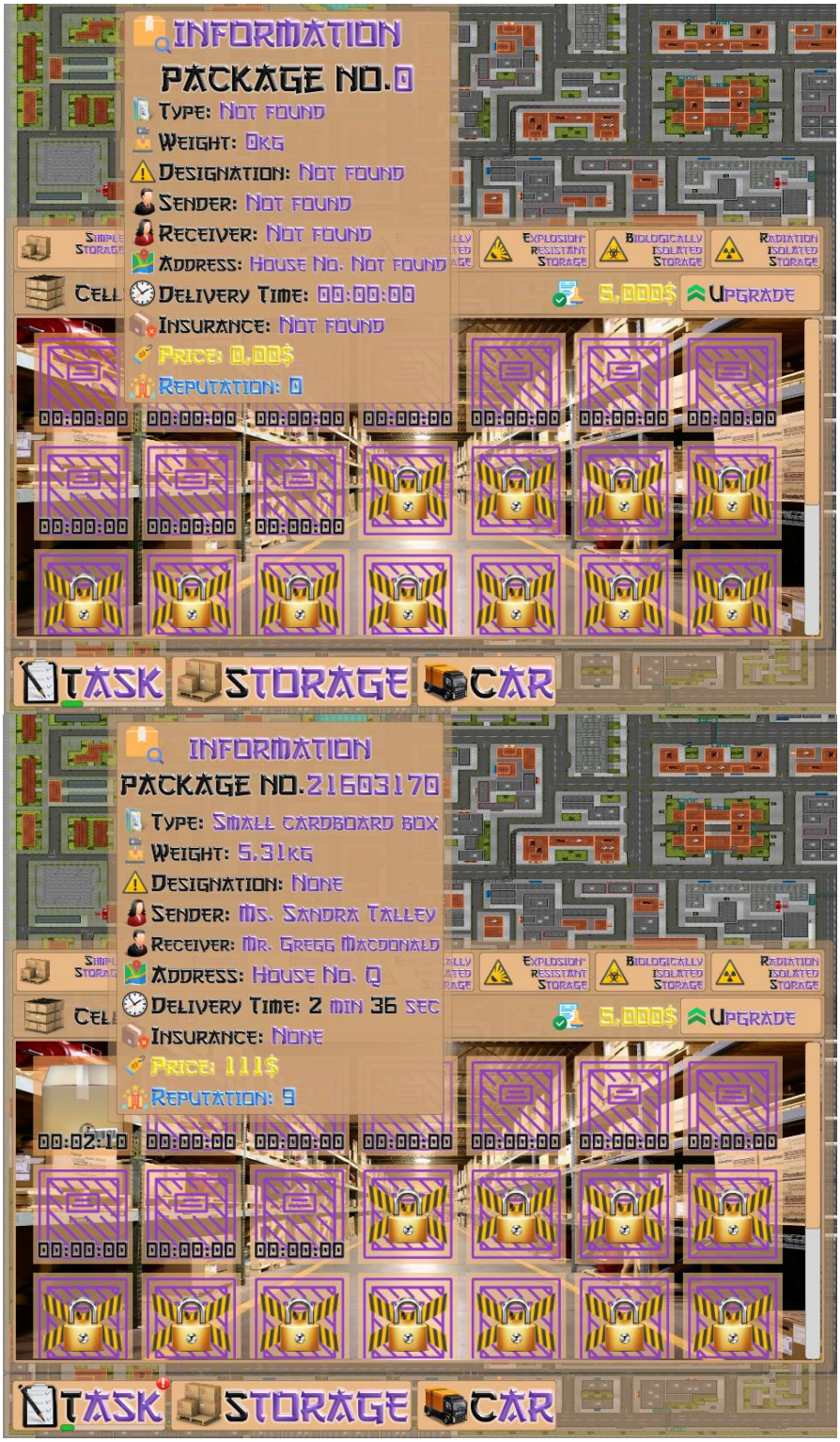


Рисунок 3.17 – Панель інформації про посилку на складі.

Якщо час доставки посилки вже збіг, то на панелі інформації відображається штраф в залежності від запізнення, як показано на рисунку 3.18.

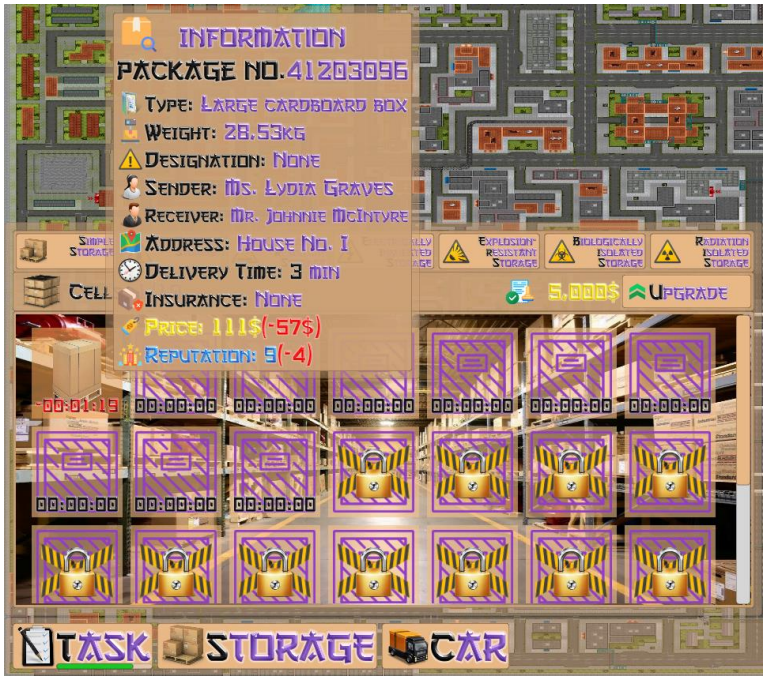


Рисунок 3.18 – Відображення штрафу за запізнення.

Натиснувши на кнопку з подвійною стрілкою на панелі рівня компанії відкривається панель рівня, як показано на рисунку 3.19.

Рисунок 3.19 – Панель рівня компанії.

На панелі рівня компанії відображаються коефіцієнти нагороди та часу, які змінюються від підвищення рівня компанії. Підвищення рівня компанії також розблоковує можливості купівлі ліцензій для розблокування нових складів(після чого їх можливо розблоковувати та модифікувати) та пакетів страхування(починають з’являтися замовлення зі страхуванням), як показано на рисунку 3.20.

Рисунок 3.20 – Панель рівня компанії після підняття декількох рівнів.

Для відправлення посилки необхідно натиснути на один із маркерів, що з’являються на ігровому полі після появи посилки на складі, як показано на рисунку 3.21.

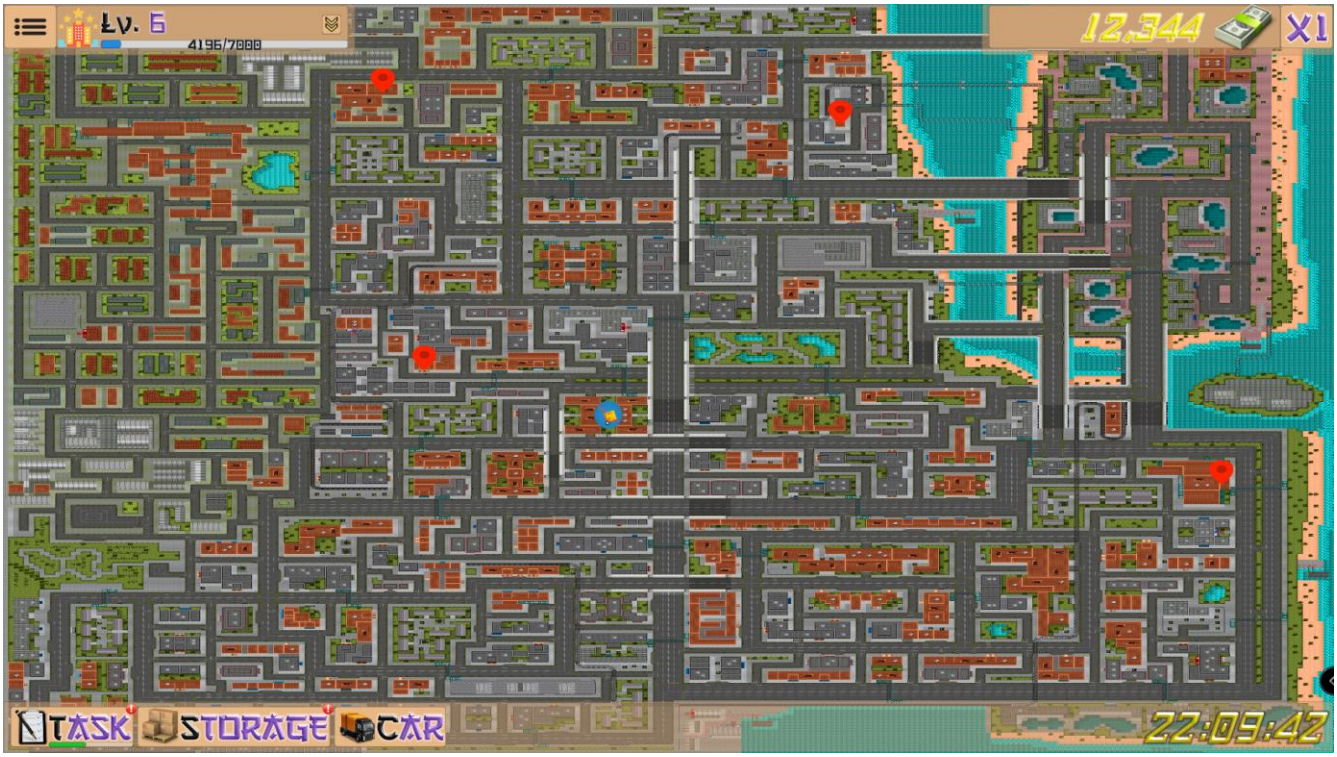


Рисунок 3.21 – Маркери адрес ще не виконаних замовлень на ігровому полі.

Кожен маркер підписаний, як показано на рисунку 3.22. Підпис з’являється при наведені курсору на маркер.

Рисунок 3.22 – Маркери адрес підписані.

Після натискання на маркер з’являється панель відправки, як показано на рисунку 3.23. Кількість комірок для посилок на панелі відправки залежить від кількості місць вибраного авто, як показано на рисунку 3.24.

Рисунок 3.23 – Панель відправки.

Рисунок 3.24 – Кількість комірок панелі відправки.

Якщо під час відправки час доставки посилки вже збіг, то на панелі відображається штраф в залежності від запізнення, як показано на рисунку 3.25.

Рисунок 3.25 – Відображення штрафу в залежності від запізнення.

Якщо адреса не співпадає з натиснутим маркером, обране авто в дорозі, заблоковане або без топлива то на панелі після натискання кнопки Send з’являтимутся відповідні попередження, як показано на рисунку 3.26.

Рисунок 3.26 – Відображення проблем з відправленням.

Натиснувши на кнопку Car відкривається панель автомобілів, як показано на рисунку 3.27.

Рисунок 3.27 – Панель автомобілів.

У автомобілів є декілька статусів: «Заблокований», «Вільний», «В дорозі» та «Нема палива», як показано на рисунку 3.28.

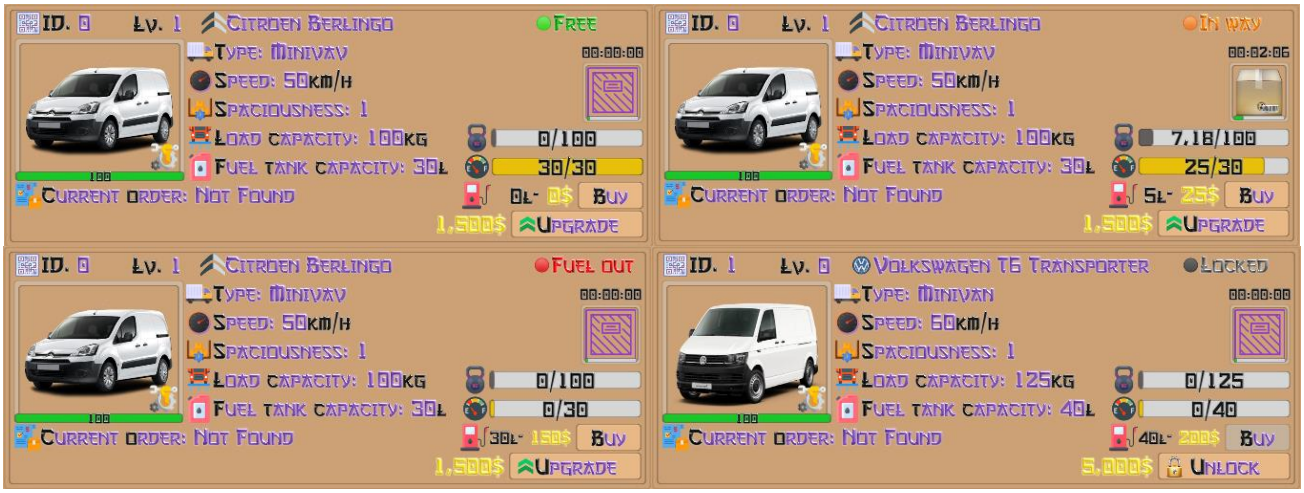


Рисунок 3.28 – Статуси автомобілів.

Під час модифікації авто збільшується його швидкість, вмісткість, вантажність та максимальний запас палива, як показано на рисунку 3.29.

Рисунок 3.29 – Повністю модифіковане авто.

Ще є декілька дрібниць, але незначних щоб їх описувати. Загалом гра потенційно доволі довга, якщо завдання розблокувати та модифікувати все на максимум. Але місця для нових механік, взаємодій та графічних перероблень ще достатньо.

Нижче наведено код одного з основних класів NewTask, який генерує нові посилки, на мові C#.

```
using System;
using System.Threading;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System.Globalization;
public class NewTask : MonoBehaviour
{    public GameObject newTaskPanel;
   public GameObject mainInfoPanel;
   private RawImage pCustomerImage;
   private RawImage pPackageImage;
   private TextMeshProUGUI pName;
   private TextMeshProUGUI pWeight;
   private TextMeshProUGUI pDesignation;
   private TextMeshProUGUI pSender;
   private TextMeshProUGUI pReceiver;
   private TextMeshProUGUI pAddress;
   private TextMeshProUGUI pTimer;
   private TextMeshProUGUI pInsurance;
   private TextMeshProUGUI pPrice;
   private TextMeshProUGUI pReputation;
   public TextMeshProUGUI timerTime;
   private RawImage pDesignationImage;
   private RawImage pSenderImage;
   private RawImage pReceiverImage;
   private RawImage pInsuranceImage;
   private string packageId = "0", packageName = "Not found",
       packageDesignation = "Not found",
      packageSender = "Not found", packageReceiver = "Not found",
      packageAddress = "Not found", packageTime = "00:00:00",
   packageInsurance = "Not found";
   private float packageWeight = 0f;
   private float packagePrice = 0f, packageReputation = 0f;
   private float packageFPrice = 0f, packageFReputation = 0f;
   public int weightRndmMax = 8;
   public int designRndmMax = 300;
   public int insuranceRndmMax = 51;
   private Texture packageImage;
   private Texture designationImage;
   private Texture senderImage;
   private Texture receiverImage;
   private Texture insuranceImage;
   private int packageNum = 0;
   private float allTime = 0f, leftTime = 0f;
   private int tTime = 0, aTime = 0, pId = 0, i = 0;
   public float kMoneyReward = 0.6f, kReputationReward = 0.6f;
   public float tPrice = 0f, tReputation = 0f;
```

```
    public float kATime = 2.5f, kDTime = 0.6f;
    private int DTShour = 0, DTSmin = 0, DTSsec = 0;
    public int ATShour = 0, ATSmin = 0, ATSsec = 0;
    public float pMulti = 0f, dMulti = 1f, tMulti = 0f, iMulti = 1f;
    public string pType = "0", dType = "0", iType = "0", address = "Not found";
    public DateTime dTimer = new DateTime();
    public DateTime cdTimer = new DateTime();
    public DateTime aTimer = new DateTime();
    public StorageController storageController;
    public ThermalStorageController thermalStorageController;
    public CRSStorageController cRSStorageController;
    public EISStorageController eISStorageController;
    public ERSStorageController eRSStorageController;
    public BISStorageController bISStorageController;
    public RISStorageController rISStorageController;
    public CountMoney countMoney;
    public Addresses addresses;
    public void TextImagFieldLink()
    {       pName = mainInfoPanel.transform.GetChild(0).GetComponent<TextMeshProUGUI>();
        pWeight = mainInfoPanel.transform.GetChild(1).GetComponent<TextMeshProUGUI>();
        pDesignation = mainInfoPanel.transform.GetChild(2).GetComponent<TextMeshProUGUI>();
        pSender = mainInfoPanel.transform.GetChild(3).GetComponent<TextMeshProUGUI>();
        pReceiver = mainInfoPanel.transform.GetChild(4).GetComponent<TextMeshProUGUI>();
        pAddress = mainInfoPanel.transform.GetChild(5).GetComponent<TextMeshProUGUI>();
        pTimer = mainInfoPanel.transform.GetChild(6).GetComponent<TextMeshProUGUI>();
        pInsurance = mainInfoPanel.transform.GetChild(7).GetComponent<TextMeshProUGUI>();
        pPrice = mainInfoPanel.transform.GetChild(8).GetComponent<TextMeshProUGUI>();
        pReputation = mainInfoPanel.transform.GetChild(9).GetComponent<TextMeshProUGUI>();
        pCustomerImage =
newTaskPanel.transform.GetChild(1).GetChild(0).GetChild(0).GetChild(0).GetComponent<RawImage>();
        pPackageImage =
newTaskPanel.transform.GetChild(1).GetChild(1).GetChild(0).GetChild(0).GetComponent<RawImage>();
        pDesignationImage = mainInfoPanel.transform.GetChild(2).GetChild(0).GetComponent<RawImage>();
        pSenderImage = mainInfoPanel.transform.GetChild(3).GetChild(0).GetComponent<RawImage>();
        pReceiverImage = mainInfoPanel.transform.GetChild(4).GetChild(0).GetComponent<RawImage>();
        pInsuranceImage = mainInfoPanel.transform.GetChild(7).GetChild(0).GetComponent<RawImage>();    }
    #region Animation
    public void NewTaskAnim()
    {       Animator animator = newTaskPanel.GetComponent<Animator>();
        if (animator != null)
        {           bool newTask = animator.GetBool("newTask");
            animator.SetBool("newTask", !newTask);       }    }
    public void TimerAnim()
    {       Animator animator = timerTime.GetComponent<Animator>();
        if (animator != null)
        {           bool timerOver = animator.GetBool("timerOver");
            animator.SetBool("timerOver", !timerOver);       }    }
    public void TimerOutAnim()
    {       Animator animator = timerTime.GetComponent<Animator>();
        if (aTimer.Hour == 0 && aTimer.Minute == 0 && aTimer.Second < 10)
        {           if (animator != null)
```

```
        {               bool timerOver = animator.GetBool("timerOver");
            animator.SetBool("timerOver", !timerOver);          }       }
        else
        {           animator.SetBool("timerOver", false);       }   }
    #endregion
    public void NewTaskCreating()
    {       CheckLicence();
        DTimeRand();
        ATimeRand();
        PWeigRand();
        DesignRand();
        SendReceivRand();
        PlacRand();
        InsurRand();
        Price();    }
    public void AddPackage()
    {       if (packageDesignation == "None" || packageDesignation == "Fragile material" || packageDesignation == "Avoid
moisture" || packageDesignation == "Follow the position")
        {           for (int pId = 0; pId < storageController.currentCellCount; pId++)
            {               if (storageController.Storage[pId].Id == 0)
                {                   storageController.Storage[pId] = new Package(Convert.ToInt32(packageId + i), packageName,
packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                    packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
                    addresses.AddressActiv(address);
                    storageController.UpdateAll();
                    i++;
                    break;              }           }       }
        if (packageDesignation == "High-temperature mode" || packageDesignation == "Low temperature mode")
        {           for (int pId = 0; pId < thermalStorageController.currentThermalCellCount; pId++)
            {               if (thermalStorageController.ThermalStorage[pId].Id == 0)
                {                   thermalStorageController.ThermalStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                    packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
                    addresses.AddressActiv(address);
                    thermalStorageController.UpdateAll();
                    i++;
                    break;              }           }       }
        if (packageDesignation == "Corrosive substance" || packageDesignation == "Oxidizing agent")
        {           for (int pId = 0; pId < cRSStorageController.currentCRSCellCount; pId++)
            {               if (cRSStorageController.CRSStorage[pId].Id == 0)
                {                   cRSStorageController.CRSStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                    packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
                    addresses.AddressActiv(address);
                    cRSStorageController.UpdateAll();
                    i++;
                    break;              }           }       }
```

```
      if (packageDesignation == "Electrical threat" || packageDesignation == "Lithium cells/Batteries" || packageDesignation
== "Magnetic fields" || packageDesignation == "Static discharge threat")
      {           for (int pId = 0; pId < eISStorageController.currentEISCellCount; pId++)
         {              if (eISStorageController.EISStorage[pId].Id == 0)
            {                  eISStorageController.EISStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                  packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
               addresses.AddressActiv(address);
               eISStorageController.UpdateAll();
               i++;
               break;              }           }       }
      if (packageDesignation == "Flammable material" || packageDesignation == "Explosive substance" ||
packageDesignation == "Gas threat")
      {           for (int pId = 0; pId < eRSStorageController.currentERSCellCount; pId++)
         {              if (eRSStorageController.ERSStorage[pId].Id == 0)
            {                  eRSStorageController.ERSStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                  packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
               addresses.AddressActiv(address);
               eRSStorageController.UpdateAll();
               i++;
               break;              }           }       }
      if (packageDesignation == "Biological threat" || packageDesignation == "Toxic material" || packageDesignation ==
"Stimulus")
      {           for (int pId = 0; pId < bISStorageController.currentBISCellCount; pId++)
         {              if (bISStorageController.BISStorage[pId].Id == 0)
            {                  bISStorageController.BISStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                  packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
               addresses.AddressActiv(address);
               bISStorageController.UpdateAll();
               i++;
               break;              }           }       }
      if (packageDesignation == "Radioactive material" || packageDesignation == "Optical radiation" || packageDesignation
== "Non-ionizing radiation")
      {           for (int pId = 0; pId < rISStorageController.currentRISCellCount; pId++)
         {              if (rISStorageController.RISStorage[pId].Id == 0)
            {                  rISStorageController.RISStorage[pId] = new Package(Convert.ToInt32(packageId + i),
packageName, packageImage, packageWeight, packageDesignation, designationImage, packageSender, senderImage,
                  packageReceiver, receiverImage, packageAddress, packageTime, packageInsurance, insuranceImage, tTime,
dTimer, cdTimer, true, false, packagePrice, packageFPrice, packageReputation, packageFReputation);
               addresses.AddressActiv(address);
               rISStorageController.UpdateAll();
               i++;
               break;              }           }       }
      packageNum++;    }
   public void CheckLicence()
   {       if (storageController.isSimpleStorageUnlocked == true)
```

```
            designRndmMax = 300;
        if (thermalStorageController.isThermalStorageUnlocked == true)
            designRndmMax = 375;
        if (cRSStorageController.isCRSStorageUnlocked == true)
            designRndmMax = 400;
        if (eISStorageController.isEISStorageUnlocked == true)
            designRndmMax = 425;
        if (eRSStorageController.isERSStorageUnlocked == true)
            designRndmMax = 450;
        if (bISStorageController.isBISStorageUnlocked == true)
            designRndmMax = 475;
        if (rISStorageController.isRISStorageUnlocked == true)
            designRndmMax = 500;    }
    #region Randomizer
    public void DTimeRand()
    {       System.Random randDTime = new System.Random((int)DateTime.Now.Ticks);
        int rdt = randDTime.Next(8, 21);
        pTimer.text = "<color=black>Time of delivery:</color=black> ";
        DTShour = 0;
        DTSmin = rdt / 4;
        DTSsec = rdt % 4;
        DTSsec = DTSsec * 15;
        tTime = (int)((DTShour * 360 + DTSmin * 60 + DTSsec) * kDTime);
        DTSmin = tTime / 60;
        DTSsec = tTime % 60;
        if (DTSmin == 0 && DTSsec != 0)
            packageTime = "<color=black>" + DTSsec + "</color=black> sec";
        if (DTSmin != 0 && DTSsec == 0)
            packageTime = "<color=black>" + DTSmin + "</color=black> min";
        if (DTSmin != 0 && DTSsec != 0)
            packageTime = "<color=black>" + DTSmin + "</color=black> min " + "<color=black>" + DTSsec +
"</color=black> sec";
        pTimer.text += packageTime;
        dTimer = new DateTime(2022, 01, 05, DTShour, DTSmin, DTSsec);    }
    public void ATimeRand()
    {       if (packageNum == 0)
        {           ATSmin = 1;
            ATSsec = 0;       }
        else
        {           System.Random randATime = new System.Random((int)DateTime.Now.Ticks);
            int rat = randATime.Next(2, 13);
            ATShour = 0;
            ATSmin = rat / 4;
            ATSsec = rat % 4;
            ATSsec = ATSsec * 15;
            aTime = (int)((ATShour * 360 + ATSmin * 60 + ATSsec) * kATime);
            ATSmin = aTime / 60;
            ATSsec = aTime % 60;       }
        aTimer = new DateTime(2022, 01, 05, ATShour, ATSmin, ATSsec);    }
    public void PWeigRand()
    {       System.Random randParcels = new System.Random((int)DateTime.Now.Ticks);
```

```
System.Random randWeight = new System.Random((int)DateTime.Now.Ticks);
int rp = randParcels.Next(1, weightRndmMax);
double rw = randWeight.NextDouble() * (5 - 1) + 100;
string srp = Convert.ToString(rp);
pName.text = "<color=black>Type:</color=black> ";
pWeight.text = "<color=black>Weight:</color=black> ";
switch (srp)
{           case "1":
       packageName = "Tiny cardboard box";
       rw = randWeight.NextDouble() * (5 - 1) + 1;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box1");
       break;
   case "2":
       packageName = "Small cardboard box";
       rw = randWeight.NextDouble() * (10 - 5) + 5;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box2");
       break;
   case "3":
       packageName = "Medium carton box";
       rw = randWeight.NextDouble() * (20 - 10) + 10;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box3");
       break;
   case "4":
       packageName = "Large cardboard box";
       rw = randWeight.NextDouble() * (30 - 20) + 20;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box4");
       break;
   case "5":
       packageName = "Ordinary wooden box";
       rw = randWeight.NextDouble() * (45 - 30) + 30;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box5");
       break;
   case "6":
       packageName = "Reinforced wooden case";
       rw = randWeight.NextDouble() * (70 - 45) + 45;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box6");
       break;
   case "7":
       packageName = "Large fortified wooden box";
       rw = randWeight.NextDouble() * (100 - 70) + 70;
       packageWeight = (float)Math.Round(rw, 2);
       packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box7");
       break;
   case "8":
       packageName = "Huge reinforced wooden crate";
```

```
            rw = randWeight.NextDouble() * (150 - 100) + 100;
            packageWeight = (float)Math.Round(rw, 2);
            packageImage = (Texture)Resources.Load("Sprites/MainSprites/Boxes/box8");
            break;      }
        pType = srp;
        packageId = srp;
        pName.text += packageName;
        pWeight.text += packageWeight + "kg";
        pPackageImage.texture = packageImage;    }
    public void DesignRand()
    {       System.Random randDesignation = new System.Random((int)DateTime.Now.Ticks);
        int rd = randDesignation.Next(0, designRndmMax);
        string srd = Convert.ToString(rd);
        pDesignation.text = "<color=black>Designation:</color=black> ";
        if (rd >= 0 && rd < 200)
        {           packageDesignation = "None";
            dMulti = 1f;
            dType = "1";
            designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/warning_sign");       }
        if (rd >= 200 && rd < 300)
        {           int rsd = randDesignation.Next(0, 91);
            if (rsd >= 0 && rsd < 31)
            {               packageDesignation = "Fragile material";
                dMulti = 1.15f;
                dType = "2";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/fragile");          }
            if (rsd >= 31 && rsd < 61)
            {               packageDesignation = "Avoid moisture";
                dMulti = 1.15f;
                dType = "3";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/avoid_moisture");          }
            if (rsd >= 61 && rsd < 91)
            {               packageDesignation = "Follow the position";
                dMulti = 1.15f;
                dType = "4";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/follow_the_position");          }
}
        if (rd >= 300 && rd < 375)
        {           int rtm = randDesignation.Next(0, 2);
            dMulti = 1.3f;
            if (rtm == 0)
            {               packageDesignation = "High-temperature mode";
                dType = "5";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/high-temperature_mode");
}
            if (rtm == 1)
            {               packageDesignation = "Low temperature mode";
                dType = "6";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/low-temperature_mode");
}       }
        if (rd >= 375 && rd < 400)
```

```
        {           int rct = randDesignation.Next(0, 2);
            if (rct == 0)
            {               packageDesignation = "Corrosive substance";
                dMulti = 1.55f;
                dType = "7";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/corrosive_substance");           }
            if (rct == 1)
            {               packageDesignation = "Oxidizing agent";
                dMulti = 1.65f;
                dType = "8";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/oxidizing_agent");           }
}
        if (rd >= 400 && rd <= 425)
        {           int ret = randDesignation.Next(0, 4);
            if (ret == 0)
            {               packageDesignation = "Electrical threat";
                dMulti = 1.75f;
                dType = "9";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/electrical_threat");           }
            if (ret == 1)
            {               packageDesignation = "Lithium cells/Batteries";
                dMulti = 1.45f;
                dType = "10";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/batteries");           }
            if (ret == 2)
            {               packageDesignation = "Magnetic fields";
                dMulti = 1.4f;
                dType = "11";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/magnetic_fields");           }
            if (ret == 3)
            {               packageDesignation = "Static discharge threat";
                dMulti = 1.25f;
                dType = "12";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/static_discharge");           }
}
        if (rd >= 425 && rd < 450)
        {           int rft = randDesignation.Next(0, 3);
            if (rft == 0)
            {               packageDesignation = "Flammable material";
                dMulti = 1.5f;
                dType = "13";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/flammable_material");           }
            if (rft == 1)
            {               packageDesignation = "Explosive substance";
                dMulti = 1.75f;
                dType = "14";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/explosive_substance");           }
            if (rft == 2)
            {               packageDesignation = "Gas threat";
                dMulti = 1.9f;
                dType = "15";
```

```
            designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/gas_threat");          }       }
        if (rd >= 450 && rd < 475)
        {           int rbt = randDesignation.Next(0, 3);
            if (rbt == 0)
            {               packageDesignation = "Biological threat";
                dMulti = 2.5f;
                dType = "16";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/biological_threat");          }
            if (rbt == 1)
            {               packageDesignation = "Toxic material";
                dMulti = 2.1f;
                dType = "17";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/toxic_material");          }
            if (rbt == 2)
            {               packageDesignation = "Stimulus";
                dMulti = 1.4f;
                dType = "18";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/stimulus");          }       }
        if (rd >= 475 && rd < 500)
        {           int rrt = randDesignation.Next(0, 3);
            if (rrt == 0)
            {               packageDesignation = "Radioactive material";
                dMulti = 2.3f;
                dType = "19";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/radioactive_material");          }
            if (rrt == 1)
            {               packageDesignation = "Optical radiation";
                dMulti = 2f;
                dType = "20";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/optical_radiation");          }
            if (rrt == 2)
            {               packageDesignation = "Non-ionizing radiation";
                dMulti = 1.85f;
                dType = "21";
                designationImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Danger/non-ionizing_radiation");
}       }
        pDesignation.text += packageDesignation;
        pDesignationImage.texture = designationImage;    }
    public void SendReceivRand()
    {
        System.Random randCustomers = new System.Random((int)DateTime.Now.Ticks);
        int rs = randCustomers.Next(1, 4);
        pSender.text = "<color=black>Sender:</color=black> ";
        if (rs == 1 || rs == 2)
        {           packageSender = "Ms. ";
            int rsfa = randCustomers.Next(0, 2);
            if (rs == 1)
            {               if (rsfa == 0)
                    senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/woman-1");
                if (rsfa == 1)
                    senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/woman-2");          }
```

```
    if (rs == 2)
    {               if (rsfa == 0)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-girl-1");
        if (rsfa == 1)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-girl-2");          }
}
if (rs == 3 || rs == 4)
{           packageSender = "Mr. ";
    int rsma = randCustomers.Next(0, 2);
    if (rs == 3)
    {               if (rsma == 0)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/men-1");
        if (rsma == 1)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/men-2");          }
    if (rs == 4)
    {               if (rsma == 0)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-guy-1");
        if (rsma == 1)
            senderImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-guy-2");          }       }
packageSender += CreateRndmName.CreateName(rs);
pSender.text += packageSender;
pSenderImage.texture = senderImage;
rs = (randCustomers.Next(1, 9) * randCustomers.Next(1, 9)) / rs;
if (rs < 10)
    packageId += "0";
packageId += rs;
int rr = randCustomers.Next(1, 4);
pReceiver.text = "<color=black>Receiver:</color=black> ";
if (rr == 1 || rr == 2)
{           packageReceiver = "Ms. ";
    int rrfa = randCustomers.Next(0, 2);
    if (rr == 1)
    {               if (rrfa == 0)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/woman-1");
        if (rrfa == 1)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/woman-2");          }
    if (rr == 2)
    {               if (rrfa == 0)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-girl-1");
        if (rrfa == 1)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-girl-2");          }       }
if (rr == 3 || rr == 4)
{           packageReceiver = "Mr. ";
    int rrma = randCustomers.Next(0, 2);
    if (rr == 3)
    {               if (rrma == 0)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/men-1");
        if (rrma == 1)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/men-2");          }
    if (rr == 4)
    {               if (rrma == 0)
```

```
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-guy-1");
        if (rrma == 1)
            receiverImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/People/young-guy-2");        }      }
    packageReceiver += CreateRndmName.CreateName(rr);
    pReceiver.text += packageReceiver;
    pReceiverImage.texture = receiverImage;
    rs = (randCustomers.Next(1, 9) * randCustomers.Next(1, 9)) / rr;
    if (rr < 10)
      packageId += "0";
    packageId += rr;   }
  public void PlacRand()
  {      System.Random randPlaces = new System.Random((int)DateTime.Now.Ticks);
    int rp = randPlaces.Next(65, 85);
    char srp = (char)rp;
    packageAddress = address = Convert.ToString(srp);
    pAddress.text = "<color=black>Address:</color=black> House No. " + packageAddress;
    rp = rp - 64;
    if (rp < 10)
      packageId += "0";
    packageId += rp;   }
  public void InsurRand()
  {      System.Random randInsurance = new System.Random((int)DateTime.Now.Ticks);
    int ri = randInsurance.Next(1, insuranceRndmMax);
    string sri = Convert.ToString(ri);
    pInsurance.text = "<color=black>Insurance:</color=black> ";
    if (ri <= 50)
    {          iMulti = 1f;
      packageInsurance = "None";
      iType = "0";
      insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-no-64");
}
    if (ri > 50 && ri <= 75)
    {          iMulti = 1.05f;
      packageInsurance = "Standart Pack-I 5";
      iType = "1";
      insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-s1-yes-
64");     }
    if (ri > 75 && ri <= 100)
    {          iMulti = 1.1f;
      packageInsurance = "Standart Pack-II 10";
      iType = "2";
      insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-s2-yes-
64");      }
    if (ri > 100 && ri <= 125)
    {          iMulti = 1.15f;
      packageInsurance = "Standart Pack-III 15";
      iType = "3";
      insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-s3-yes-
64");      }
    if (ri > 125 && ri <= 150)
    {          iMulti = 1.2f;
```

```
        packageInsurance = "Premium Pack-I 20";
        iType = "4";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-p1-yes-
64");      }
    if (ri > 150 && ri <= 175)
    {          iMulti = 1.3f;
        packageInsurance = "Premium Pack-II 30";
        iType = "5";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-p2-yes-
64");      }
    if (ri > 175 && ri <= 200)
    {          iMulti = 1.4f;
        packageInsurance = "Premium Pack-III 40";
        iType = "6";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurancep3-yes-64");
}
    if (ri > 200 && ri <= 225)
    {          iMulti = 1.5f;
        packageInsurance = "Ultimate Pack-I 50";
        iType = "7";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-u1-yes-
64");      }
    if (ri > 225 && ri <= 250)
    {          iMulti = 1.75f;
        packageInsurance = "Ultimate Pack-II 75";
        iType = "8";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-u2-yes-
64");      }
    if (ri > 250 && ri <= 275)
    {          iMulti = 2f;
        packageInsurance = "Ultimate Pack-III 100";
        iType = "9";
        insuranceImage = (Texture)Resources.Load("Sprites/MainSprites/Icons/Insurance/icons8-box-insurance-u3-yes-
64");      }
    pInsurance.text += packageInsurance;
    pInsuranceImage.texture = insuranceImage;    }
  public void Price()
  {       float price = 150;
    float reputation = 10;
    if (Convert.ToInt32(pType) >= 1 && Convert.ToInt32(pType) <= 4)
    {          System.Random randCBPrice = new System.Random((int)DateTime.Now.Ticks);
        int rcbp = randCBPrice.Next(1, 5);
        string srcbp = Convert.ToString(rcbp);
        switch (srcbp)
        {              case "1":
                pMulti = 0.5f;
                break;
            case "2":
                pMulti = 0.7f;
                break;
            case "3":
```

```
                pMulti = 0.9f;
                break;
            case "4":
                pMulti = 1.1f;
                break;          }       }
        if (Convert.ToInt32(pType) >= 5 && Convert.ToInt32(pType) <= 8)
        {           System.Random randWBPrice = new System.Random((int)DateTime.Now.Ticks);
            int rwbp = randWBPrice.Next(1, 5);
            string srwbp = Convert.ToString(rwbp);
            switch (srwbp)
            {               case "1":
                pMulti = 1.3f;
                break;
            case "2":
                pMulti = 1.5f;
                break;
            case "3":
                pMulti = 1.7f;
                break;
            case "4":
                pMulti = 1.9f;
                break;          }       }
        if (tTime <= 120)
        {           tMulti = 100;       }
        if (tTime > 120 && tTime <= 180)
        {           if (tTime <= 150)
            tMulti = 90;
            if (tTime >= 150 && tTime <= 180)
            tMulti = 80;        }
        if (tTime > 180 && tTime <= 240)
        {           if (tTime <= 210)
            tMulti = 70;
            if (tTime >= 210 && tTime <= 240)
            tMulti = 60;        }
        if (tTime > 240 && tTime <= 300)
        {           if (tTime <= 270)
            tMulti = 50;
            if (tTime >= 270 && tTime <= 300)
            tMulti = 40;        }
        if (tTime > 300)
        {           tMulti = 30;        }
        tPrice = (int)(((((price * pMulti) + tMulti) * dMulti * iMulti) * kMoneyReward) * countMoney.ik);
        tReputation = (int)(((((reputation * pMulti) + (tMulti / 10)) * dMulti * iMulti) * kReputationReward) *
countMoney.ik);
        packagePrice = tPrice;
        packageReputation = tReputation;
        pPrice.text = "Price: " + packagePrice.ToString("0,0", CultureInfo.InvariantCulture) + "$";
        pReputation.text = "Reputation: " + packageReputation;    }
#endregion}
```

ВИСНОВКИ

В межах дипломного проекту було реалізовано програму "Delivery Service", яка розрахована для людей які хочуть пограти в симулятор і добре провести час, при цьому не навантажуючись занадто реалістичними умовностями. В ході написання дипломної роботи була розроблена гра "Delivery Service" під управлінням операційної системи Windows у середовищі розробки Unity мовою програмування C#.

Також під час виконання дипломної роботи дослідив базові поняття об’єктно-орієнтованого програмування, сформував технічне завдання та план його виконання.

Під час виконання дипломної роботи проаналізував існуючі методи відтворення анімації та застосував об’єктно-орієнтований підхід, створивши систему класів для реалізації програми "Delivery Service".

В результаті виконання дипломної роботи було розроблено робочий варіант програми "Delivery Service", яка працює локально та може бути розширена іншими розробниками для покращення функціоналу на будь-якому комп'ютері на ос Windows.

ПЕРЕЛІК ПОСИЛАНЬ

1. Симуляційна відеогра [Електронний ресурс] - Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Симуляційна_відеогра.
2. Мова програмування C# [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/C_Sharp.
3. Unity [Електронний ресурс] - Режим доступу до ресурсу: https://unity.com/
4. Unity спрайти [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Sprites.html.
5. Unity 2d [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Unity2D.html.
6. Unity графіка [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Graphics.html.
7. Unity фізика [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Physics.html.
8. Unity скриптинг [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Scripting.html.
9. Unity анімація [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/Manual/Animation.html.
10. Користувацький інтерфейс [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/UISystem.html.
11. Компоненти Unity [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/UsingComponents.html
12. Unity об'єкти [Електронний ресурс] - Режим доступу до ресурсу: https://docs.unity3d.com/ru/current/class-GameObject.html
13. Visual Studio 2022 [Електронний ресурс] - Режим доступу до ресурсу: https://visualstudio.microsoft.com/vs/
14. Джефрі Р.П. Програмування на платформі Microsoft.NET Framework 4.5 на мові C# / Джефрі Р.П., 2016. - 896 с. - (Пітер).
15. Эндрю Т.Л. Мова програмування C# і платформа .NET 4.6 / Эндрю Т.Л, Філіп Д.Л., 2016. - 1440 с. - (Вільямс).
16. Тюкачев Н.А. Алгоритми і структури даних C# / Тюкачев Н.А, Хлебостроев В.Г., 2017. - 232 с. - (Лань).
17. Констатайн Л. Розробка програмного забезпечення / Констатайн Л, Локвуд Л., 2004. - 470 с. - (Пітер).
18. Мандел Т. Розробка користувацького інтерфейсу / Мандел Т, 2001. - 416 с. - (ДМК Пресс).
19. Періс Б.А. Unity для розробника. Мобільні мультиплатформенні ігри / Періс Б.А., 2016. - 352 с. - (Пітер).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО- НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка гри «Delivery Service»у жанрі симулятор мовою C# на двигуні Unity

Виконав студент 5 курсу
групи ППЗ-51
СендецькийМикола Ігорович
Керівникроботи
Коба Андрій Борисович

Київ – 2022

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** розробка гри «Delivery Service» у жанрі симулятор мовою C# на двигуні Unity з використанням нових механік та властивостей
- **Об’єкт дослідження:** процес розробки гри симулятора мовою C# на двигуні Unity
- **Предмет дослідження:** методи та засоби розробки гри симулятора «Delivery Service»

АНАЛОГИ

«Служба доставки онлайн»

«Totally ReliableDelivery Service»

«DabbawallaInternational»

«WastelandExpress.Delivery Service»

3

Таблиця порівняння аналогів

	«Служба доставки онлайн»	«Dabbawalla International»	«Totally Reliable Delivery Service»	«Wasteland Express. Delivery Service»	«Delivery Service»
Можливість керувати кур'єром	Ні	Ні	Так	Ні	Ні
Можливість керувати автомобілем доставки	Так	Ні	Так	Так	Ні
Можливість керувати компанією	Ні	Так	Ні	Ні	Так
Відкритий ігровий світ	Так	Ні	Так	Ні	Ні
Наявність мультиплеєру	Ні	Ні	Так	Так	Ні
Можливість ігрових модифікацій	Так	Так	Ні	Так	Так

4

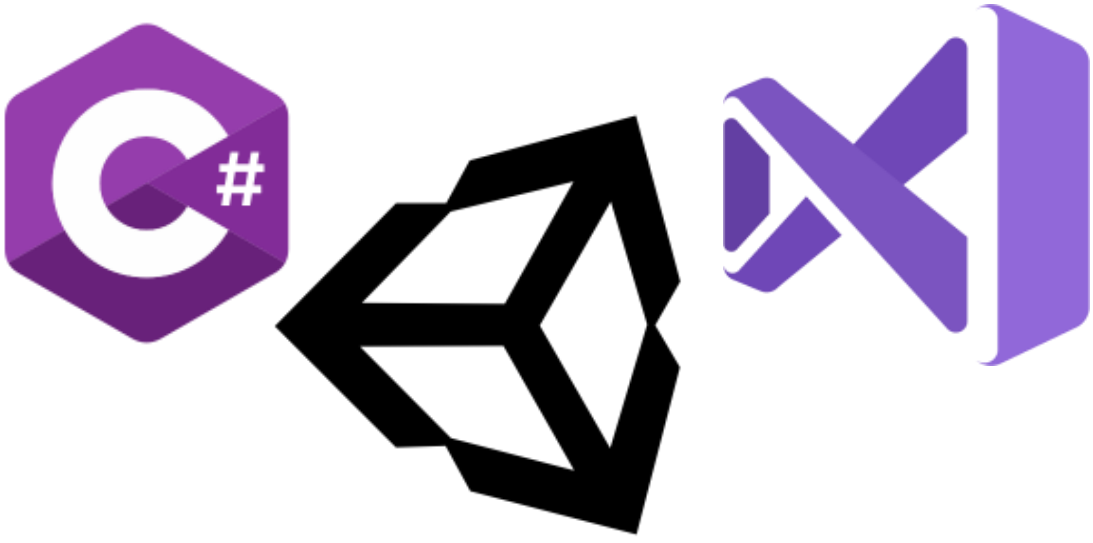
ТЕХНІЧНІ ЗАВДАННЯ

- Аналіз предметної галузі і виділення переваги розробленого ПЗ
- Розробка гри симулятора мовою C# на двигуні Unity
- Реалізація генерації випадкових замовлень
- Реалізація складу для зберігання посилок
- Реалізація авто для доставки замовлень
- Реалізація мапи міста з адресами
- Реалізація можливості розблокування та модифікації складів, авто та компанії
- Тестування гри симулятора

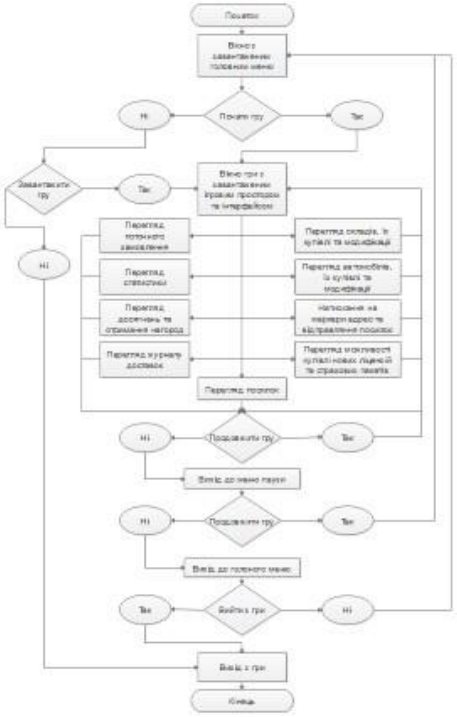
5

Програмні засоби реалізації гри

- Гра була написана за допомогою мови програмування C#, з використанням ігрового двигуна Unity та візуального редактора коду Microsoft Visual Studio 2017.

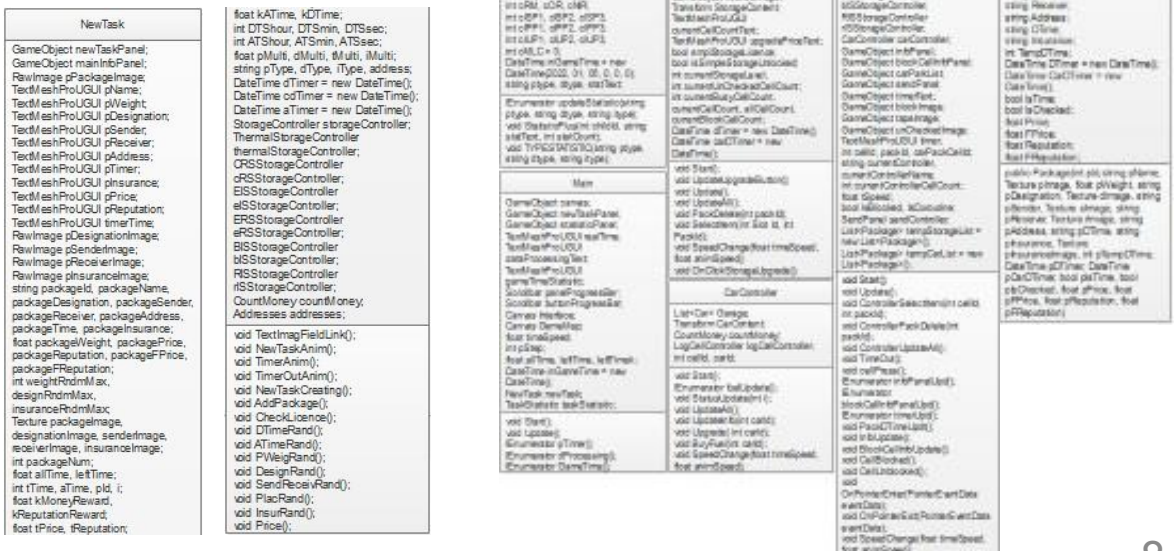


Алгоритм «Delivery Service»



7

Діаграма класів «Delivery Service»

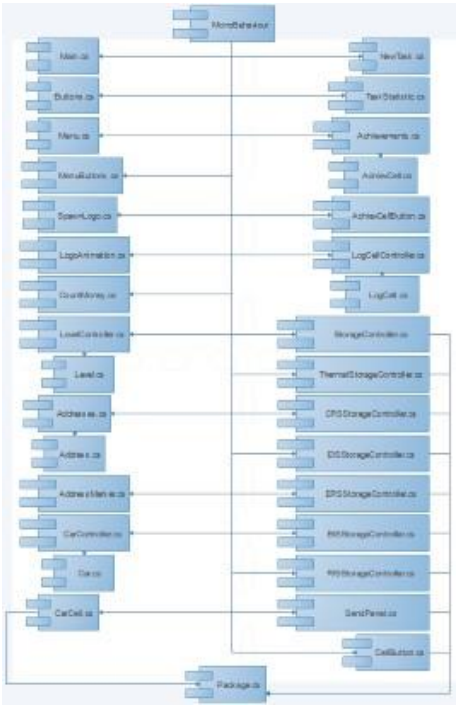


8

Діаграма прецедентів «Delivery Service»

9

Діаграма компонентів «Delivery Service»



10

Головне меню та ігрове поле з інтерфейсом користувача

11

Основні панелі інтерфейсу користувача

12

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

- СендецькийМ.І. Розробка гри «Delivery Service» у жанрі симулятор мовою C# на двигуні Unity. Збірниктез. 20.04.2022, ДУТ, м. Київ – К.: ДУТ, 2022

13

ВИСНОВКИ

- Реалізованопрограму"Delivery Service", яка розрахованадля людей які хочуть пограти в симулятор і добре провести час, при цьому не навантажуючись занадто реалістичними умовностями В ході написаннядипломної роботи була розробленагра "Delivery Service" під управліннямопераційної системи Windows у середовищі розробкиUnity мовоюпрограмування C#.
- Дослідив базові поняття об’єктно-орієнтованого програмування, сформував технічне завданнята план його виконання
- Проаналізувавіснуючі методи відтворення анімації та застосував об’єктно-орієнтований підхід, створивши систему класівдля реалізації програми"Delivery Service".

14

ДЯКУЮ ЗА УВАГУ!