МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

ДИПЛОМНА БАКАЛАВРСЬКА РОБОТА

на тему

Розробка програмного забезпечення для логічної гри в мобільному додатку

Виконав: студент групи БІТск-21
спеціальності 122 Комп'ютерні науки
освітньої програми Комп'ютерні науки
Валентин ЛАПА

Науковий керівник: к.т.н., доцент
Тетяна АСТІСТОВА

Рецензент : к.т.н., доцент
Геннадій МЕЛЬНИК

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук та технологій

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
_____ В.Ю. Щербань

«_____» _____ 2023 р.

ЗАВДАННЯ

НА ДИПЛОМНУ БАКАЛАВРСЬКУ РОБОТУ

студенту
Лапі Валентину Сергійовичу

Тема роботи Розробка програмного забезпечення для логічної гри в мобільному додатку

Науковий керівник роботи Астістова Тетяна Іванівна, : к.т.н., доцент

затверджені наказом КНУТД від "08" листопада 2022 року № 224-УЧ

Строк подання студентом дипломної роботи 10.06.2023 р.

2. Вихідні дані до дипломної роботи : Розробки кафедри комп'ютерних наук,

3. Зміст дипломної бакалаврської роботи: 1) Аналіз предметної області, 2) Проектування програмної реалізації, 3) Розробка програмного продукту.

5. Дата видачі завдання 01. 02 .2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної бакалаврської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	03.05.2023	
2	Розділ 1 Аналіз предметної області	06.05.2023	
3	Розділ 2 Проектування програмної реалізації	12.05.2023	
4	Розділ 3 Розробка програмної реалізації та її тестування	25.05.2023	
5	Висновки	26.05.2023	
6	Оформлення дипломної бакалаврської роботи	28.05.2023	
7	Здача дипломної бакалаврської роботи на кафедру для рецензування	30.05.2023	
8	Перевірка дипломної бакалаврської роботи на наявність ознак плагіату	02.06.2023	
9	Подання дипломної бакалаврської роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	07.06.2023	

Студент _____ Валентин ЛАПА

Науковий керівник роботи _____ Тетяна АСТІСТОВА

Рецензент _____ Геннадій МЕЛЬНИК

АНОТАЦІЯ

ЛапаВ.С. Розробка програмного забезпечення для логічної гри в мобільному додатку

Дипломна бакалаврська робота за спеціальністю 122 - «Комп'ютерні науки» – Київський національний університет технологій та дизайну, Київ, 2023 рік.

У дипломній роботі було проведено дослідження та аналіз методів та алгоритмів для розробки програмного забезпечення в мобільному додатку, спрямованого на створення логічної гри під назвою "Parking Logic Game". Гра має на меті вивезти автомобіль швидкої допомоги з паркінгу. Ігрове поле має розмірність 7x7, на якому розташовані різні автомобілі. Кожен автомобіль може рухатися лише по горизонталі або по вертикалі.

Під час дослідження було розглянуто різні методи та класи для створення ігрової сітки. Було вивчено алгоритми, що дозволяють розрахувати найкраще та плавне переміщення автомобілів, враховуючи їх обмежені можливості руху.

У результаті роботи було успішно розроблено програмне забезпечення для мобільного додатку, яке реалізує логічну гру "Parking Logic Game". Гра надає гравцю можливість випробувати свої навички у вивезенні автомобіля швидкої допомоги з переповненого паркінгу. Застосунок розроблено з використанням мови програмування C# та ігрового рушія Unity3D, що забезпечує захоплюючий геймплей, реалістичну графіку та інтуїтивний інтерфейс для користувача.

Ключові слова: C#, Unity3D, алгоритм, програмне забезпечення, Visual Studio, Android.

ANNOTATION

Lapa Valentyn. Topic Development of software for a logic game in a mobile application**.**

Bachelor's thesis degree work specialty 122 "Computer sciences"-

Kyiv National University of Technologies and Design, Kyiv, 2023.

In this thesis, we studied and analysed methods and algorithms for developing software in a mobile application aimed at creating a logic game called "Parking Logic Game". The game aims to get an ambulance out of the car park. The playing field has a dimension of 7x7, on which different cars are located. Each car can only move horizontally or vertically.

During the research, various methods and classes were considered for creating the game grid. Algorithms were studied to calculate the best and smoothest movement of cars, taking into account their limited movement capabilities.

As a result of the work, we have successfully developed software for a mobile application that implements the Parking Logic Game. The game gives the player the opportunity to test their skills in removing an ambulance from a crowded car park. The application was developed using the C# programming language and the Unity3D game engine, which provides exciting gameplay, realistic graphics and an intuitive user interface.

Keywords: C#, Unity3D, algorithm, graphics, , softwar,Visual Studio, Android

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп’ютерних технологій

Кафедра комп’ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної бакалаврської роботи
на тему
Розробка програмного забезпечення для логічної гри в мобільному додатку

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, де мобільні пристрої стали невід’ємною частиною нашого повсякденного життя, розробка програмного забезпечення для мобільних додатків стала важливою складовою технологічного прогресу. Особливо значущими є програмні рішення, спрямовані на навчання та розвиток. У рамках цього контексту виникає потреба у створенні логічних ігор, які не лише надають користувачам розвагу, але й сприяють розвитку їхнього мислення, логічного мислення та проблемного мислення.

Однією з таких логічних ігор є "Parking Logic Game" - мобільна гра, яка ставить перед гравцем завдання впорядкування автомобілів на парковці з обмеженим простором. Гравець повинен знайти оптимальний спосіб розташування автомобілів, дотримуючись певних правил та обмежень, з метою звільнення місця для конкретного автомобіля.

Проблема, що потребує вирішення, полягає у стимулюванні та розвитку логічного мислення гравців, особливо молодшого покоління, за допомогою захоплюючої та веселої гри. На сьогоднішній день, хоча існують різні логічні ігри, недостатньо програмних продуктів, спрямованих саме на розвиток логічного мислення у форматі мобільних додатків. Тому розробка "Parking Logic Game" є актуальною, оскільки вона відповідає сучасним потребам користувачів, а саме - надає можливість комфортного та цікавого способу розвитку логічного мислення.

Мета. Метою даного дослідження є розробка програмного забезпечення для мобільного додатку "Parking Logic Game", яке надасть користувачам можливість вдосконалювати своє логічне мислення та проблемне мислення через захоплюючий геймплей.

Завданням дослідження є аналіз предметної області логічних ігор, огляд існуючих аналогів, вибір оптимальних технологій та середовища розробки, розробка самого додатку та його тестування з метою оцінки його ефективності та користувацького задоволення.

Об'єкт предмет дослідження. Об'єктом дослідження виступають процеси розробки програмного забезпечення для мобільного додатку "Parking Logic Game", а предметом дослідження - механізми та функціональні можливості гри, які сприяють розвитку логічного мислення гравців.

Методи реалізації. Після аналізу мов програмування була обрана мова програмування C#. Вона є найбільш вдалим варіантом для даного проекту, оскільки вона є основною мовою програмування в Unity3D - популярному середовищі розробки ігор. Тому, для створення логічної гри, було вирішено використовувати C# у поєднанні з Unity3D.

Це рішення має кілька переваг. По-перше, використання мови програмування C# дозволяє нам ефективно використовувати можливості Unity3D, такі як графічний двигун, фізика, анімація та багато іншого. Це спрощує розробку гри і забезпечує швидке прототипування і ітерацію.

По-друге, C# є потужною і об'єктно-орієнтованою мовою програмування з багатьма функціями, що полегшують створення складних логічних структур та алгоритмів. Вона має широкий набір бібліотек і фреймворків, що дозволяє нам ефективно працювати зі звуком, графікою, введенням користувача та іншими аспектами гри.

Крім того, спільнота розробників, що працюють з Unity3D та C#, є досить великою і активною. Це означає, що знайти допомогу, поради та ресурси нескладно, що сприяє швидкому розв'язанню проблем та вирішенню складних завдань.

Отже, використання мови програмування C# разом з Unity3D є логічним та ефективним вибором для розробки логічної гри "Parking Logic Game". Це дозволяє нам максимально використати можливості розробки ігор, забезпечуючи швидкість, гнучкість та результативність при створенні цієї захоплюючої гри.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Мобільні ігри та загальні відомості про них

Мобільні ігри - це ігрові додатки для мобільних пристроїв, таких як смартфони та планшети. Вони стали дуже популярними в останні роки завдяки зростанню технологій мобільних пристроїв та їх доступності для більшої частини населення.

Мобільні ігри можна поділити на кілька категорій, зокрема:

- Аркадні ігри: це ігри, де головним завданням є зібрати максимальну кількість очок або вижити якомога довше.
- Рольові ігри: це ігри, де гравець грає роль певного персонажа і розвиває його навички та характеристики.
- Стратегічні ігри: це ігри, де гравець керує армією або цілою нацією та займається розвитком свого королівства чи імперії.
- Симулятори: це ігри, які симулюють реальне життя та різні види діяльності.
- Логічні ігри - це ігри, які вимагають від гравця використання розуму та логіки для вирішення різних завдань та головоломок.

Крім цього, мобільні ігри можуть бути безкоштовними або коштувати. гроші. Безкоштовні ігри часто містять внутрішні покупки, які дозволяють гравцям отримувати додаткові функції та можливості. Платні ігри зазвичай коштують від кількох доларів до кількох десятків доларів та не містять додаткових внутрішніх покупок.

Особливістю мобільних ігор є їх доступність та мобільність. Гравці можуть грати у свої улюблені ігри будь- де і коли їм зручно - в дорозі, в черзі, вільну хвилину вдома чи на роботі. Мобільні ігри також дозволяють гравцям зіграти з друзями з усього світу за допомогою мережі Інтернет, що робить їх ще більш популярними та зручними для використання.

Загалом, мобільні ігри - це важлива складова сучасної індустрії геймінгу, що стала доступною для широкої аудиторії завдяки зростанню технологій та

популярності мобільних пристроїв. Це означає, що мобільні ігри будуть продовжувати розвиватись та створюватись, щоб задовольнити потреби гравців та надати їм більш різноманітні ігрові враження.

Також варто зазначити, що мобільні ігри стали популярними засобом соціалізації та спілкування з іншими гравцями. Багато ігор мають режими онлайн гри, де гравці можуть змагатися один з одним або співпрацювати, що додає іграм елементи соціальної взаємодії та співпраці.

За останні роки мобільні ігри стали однією з найбільш популярних та швидкоростучих форм розваг у світі.

Коротка статистика по мобільним іграм у світі за даними сайту Statista:

- За прогнозами, дохід на ринку мобільних ігор досягне 286,50 млрд доларів США у 2023 році.
- Очікується, що річний темп зростання доходу (CAGR 2023-2027 рр.) становитиме 7,08%, що призведе до прогнозованого обсягу ринку в 376,70 млрд доларів США до 2027 року.
- На ринку мобільних ігор очікується, що до 2027 року кількість користувачів становитиме 2,32 млрд користувачів.
- Проникнення користувачів становитиме 25,1% у 2023 році і, як очікується, досягне 29,2% до 2027 року.
- У глобальному порівнянні, найбільший дохід буде отримано в Китаї (81 900,00 млн доларів США у 2023 році).
- Середній дохід на одного користувача (ARPU) на ринку мобільних ігор, за прогнозами, становитиме 148,80 доларів США у 2023 році.
- За даними дослідницької компанії Newzoo, у 2021 році понад 3 мільярди людей по всьому світу грають у мобільні ігри. Найбільш активними користувачами є жінки та діти.

Крім того, дослідження показують, що найбільша кількість ігор грається на мобільних пристроях, що працюють під операційною системою Android

1.2. Загальні відомості про ігри на логіку

Ігри на логіку - це тип ігор, де гравці мають вирішувати різноманітні завдання та головоломки за допомогою логічного мислення та розуміння принципів гри. Цей жанр ігор має багато різновидів , таких як кросворди, головоломки з рухом та інші.

Однією з основних характеристик ігор на логіку є використання головоломок та задач, які потребують від гравців логічного мислення та розуміння. Ці ігри зазвичай вимагають від гравців уваги до деталей та розвитку логічних навичок, що може покращити їх креативність та аналітичні здібності.

Цей жанр ігор залишається популярним серед гравців усіх вікових груп, які шукають викликів та розваги, які допоможуть їм розвивати логічне мислення та навички розв'язування проблем. Також, ігри на логіку можуть бути корисні для зниження рівня стресу та розвантаження розуму після напруженого робочого дня.

Ігри на логіку можуть мати різний рівень складності, який може варіюватись від простих ігор для дітей до складних головоломок для дорослих. Зазвичай, ігри на логіку мають декілька рівнів складності, які можуть варіюватись від легкого до дуже складного.

Ось деякі загальні характеристики різних рівнів складності:

1. Легкий рівень: ігри на цьому рівні складності зазвичай мають прості завдання та легкі правила гри. Наприклад, ігри на збіг, лабіринти та головоломки, що містять небагато елементів, відносяться до цього рівня.
2. Середній рівень: ігри на цьому рівні складності зазвичай мають більш складні правила гри та завдання. Наприклад, складніші версії судоку, кросвордів та головоломок можуть відноситись до цього рівня.
3. Високий рівень: ігри на цьому рівні складності зазвичай мають дуже складні завдання та правила гри. Наприклад, головоломки з великою кількістю елементів, складні завдання на знаходження шляхів або головоломки з шифруванням можуть відноситись до цього рівня.

4. Експертний рівень: цей рівень складності вимагає від гравця великої уваги, точності та глибокого розуміння правил гри. На цьому рівні можуть бути головоломки з неочікуваними вирішеннями та незвичними правилами.

Крім того, деякі ігри на логіку можуть мати динамічні рівні складності, що залежать від успіху гравця в грі. Наприклад, у деяких іграх на логіку, які включають повторювані завдання, рівень складності може змінюватись в залежності від того, наскільки успішність гравця виконує завдання. Якщо гравець виконує завдання швидко та безпомилково, то рівень складності може збільшуватись.

Також деякі ігри на логіку можуть мати систему підказок або допомогти гравцю впоратись з більш складними завданнями. Наприклад, деякі ігри можуть надавати гравцям можливість переглянути правильні відповіді, використовувати підказки або зменшувати кількість варіантів для вибору правильної відповіді.

Крім того, деякі ігри на логіку можуть мати режим гри проти інших гравців, що додає складності до гри. Наприклад, гра проти комп'ютера або інших гравців може вимагати від гравця більш високого рівня складності, оскільки вони мають змагатись з більш досвідченими гравцями.

1.3. Огляд існуючих аналогів

Перед розробкою програмної реалізації мобільних логічних ігор необхідно виконати огляд існуючих ігор цього жанру для уникнення помилок, що можуть бути вже зроблені в інших іграх, та зрозуміти загальну концепцію їх використання. Під час пошуку мобільних логічних ігор було виявлено широкий вибір ігор, які різняться рівнем складності, графікою, тематикою та іншими параметрами. Наприклад, деякі ігри націлені на розвиток певних навичок, таких як логічне мислення або просторова уява, тоді як інші ігри зосереджені на розвагах та розважальних задачах.

Незалежно від того, на якій мобільній логічній грі я зупиню свій вибір, основними характеристиками гри будуть її рівень складності, графіка, геймплей

та звуковий супровід. Рівень складності може варіюватися від простих ігор з декількома рівнями до складних ігор з сотнями рівнів та різноманітними завданнями. Графіка може бути як 2D, так і 3D, а геймплей може включати різні елементи, такі як головоломки, збирання об'єктів або стратегічне планування дій. Звуковий супровід також може відрізнятися від простих звукових ефектів до повноцінної музики та голосових коментарів.

1.3.1. Гра 2048

2048 - це популярна гра на логіку, яка була розроблена Габріелем Кузніцою в 2014 році. Гра відбувається на дошці 4x4 клітинок, де гравець повинен об'єднувати числа від 2 до 2048, рухаючи цифри вгору, вниз, вліво або вправо.

Гравець починає з двох клітинок з числами 2 або 4. Кожен хід гравець може здійснити один рух: перемістити всі числа вгору, вниз, вліво або вправо. Якщо дві клітинки з однаковими числами зіткнуться, то вони об'єднуються в одну, а їх значення додається разом. Наприклад, дві клітинки з числами 2 і 2 об'єднуються в одну клітинку з числом 4.

Гра закінчується, коли на дошці більше немає вільних клітинок або коли гравець досягне числа 2048. Якщо на дошці немає вільних клітинок і гравець не може зробити жодного ходу, то гра також закінчується.

2048 стала дуже популярною через свою просту геймплейну механіку та викликаючи в той же час складні стратегічні виклики, які вимагають від гравців добре продуманих рішень. Гра доступна на різних платформах, включаючи веб-версії, мобільні додатки та інші.

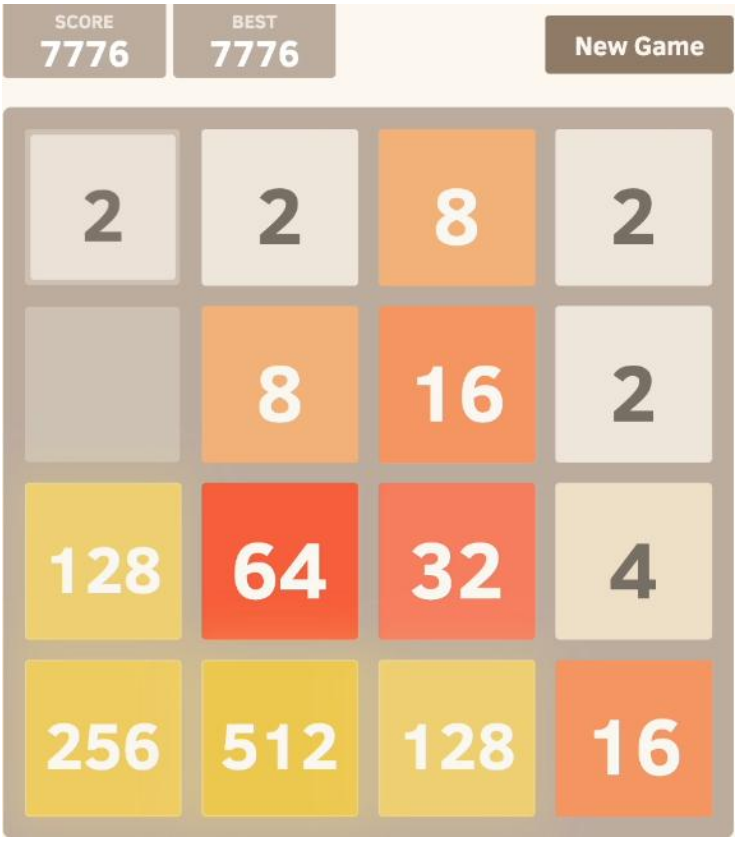


Рис. 1.1 - Інтерфейс гри 2048

1.3.2. Гра тетріс

Тетріс - це класична логічна гра, яка була розроблена компанією Tetris Company в 1984 році. Гра складається з ряду фігурок, які складаються з чотирьох квадратів, які можуть бути розташовані в різних комбінаціях. Мета гри полягає у тому, щоб розмістити ці фігурки в такий спосіб, щоб заповнити горизонтальні рядки без прогалин. Кожен заповнений рядок зникає, звільняючи місце для нових фігурок. Чим більше рядків ви видаляєте, тим більше очок ви отримуєте.

У грі є кілька рівнів складності, які можна налаштувати відповідно до ваших навичок гри. Ви можете вибрати рівень легкої, середньої або важкої складності. Також є можливість вибрати швидкість, з якою фігурки падають, відповідно до ваших особистих переваг.

Один з головних елементів гри - стратегія. Ви повинні добре обміркувати, куди помістити фігурку, щоб не залишити прогалин і якнайбільше заповнити

рядки. Також важливо знати, як правильно використовувати фігурки, які найбільше підходять для поточної ситуації.

У грі є можливість зберігати свій рекорд, щоб порівнювати його з рекордами інших гравців. Також є можливість змінити оформлення гри за допомогою налаштувань відображення.

Тетріс - це чудова гра для тих, хто любить використовувати свій розум і стратегічне мислення. Вона є класикою жанру і досі залишається однією з найпопулярніших логічних ігор у світі.

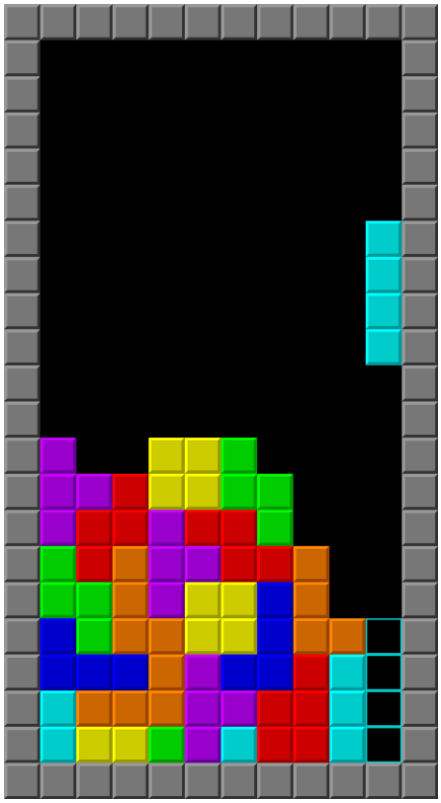


Рис. 1.2 -Інтерфейс гри тетріс

1.3.3. Гра I Love Hue

I Love Hue - це мобільна гра на логіку, в якій гравець повинен відновлювати порушене кольорове спектральне коло. У грі відображаються різні кольорові блоки, які розбиваються на фрагменти, і ваше завдання - правильно їх відновити, щоб отримати спектральне коло.

Гра складається з 900 рівнів, і кожен наступний рівень є складнішим за попередній. На початку кожного рівня ви побачите розбите коло, і всі фрагменти будуть відображені нижче. Щоб відновити коло, ви повинні перетягнути фрагменти на правильне місце в колі, дотримуючись кольорової послідовності.

Що далі? Якщо ви успішно пройдете всі 900 рівнів, то гра пропонує вам створити власний колірний пазл. Ви можете вибрати кількість фрагментів, колір, форму та інші параметри, щоб створити свій власний пазл.

I Love Hue - гра з прекрасною графікою та музикою, яка допоможе вам розслабитися та сконцентруватися під час вирішення складних кольорових головоломок. Це чудова гра для тих, хто любить грати в логічні ігри.

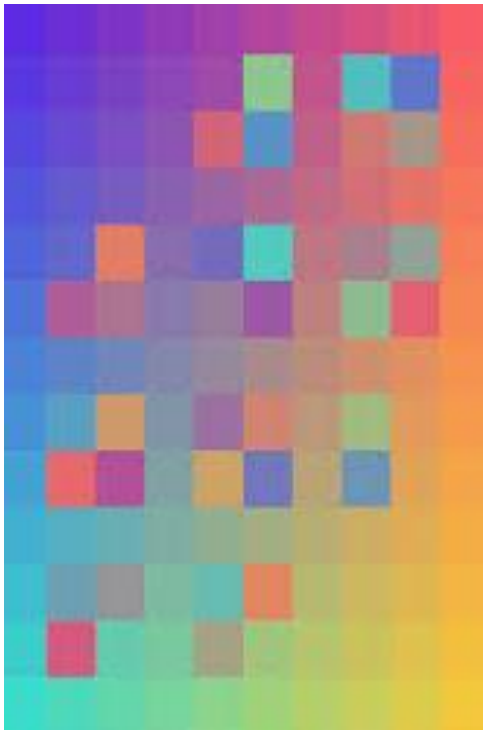


Рис. 1.3 - Інтерфейс гри I Love Hue

1.3.4. Гра Block Puzzle

Block Puzzle - це логічна гра, де гравці повинні розміщувати фігурки з різними формами, щоб заповнити головоломні сітки. Головна мета гри полягає в тому, щоб заповнити всі клітинки на полі, використовуючи задану кількість фігурок.

У грі Block Puzzle є кілька різних режимів гри, які відрізняються за складністю та кількістю фігурок. У деяких режимах гравці можуть обертати фігурки, щоб їх розмістити на більш складних головоломних сітках.

Гра пропонує простий, але ефективний інтерфейс, що дозволяє легко переміщувати фігурки та повертати їх. Гравці можуть обирати різні теми оформлення та кольори фігурок, щоб налаштувати гру під свій смак.

Гра Block Puzzle є чудовою грою для розвитку креативності та логічного мислення. Вона допоможе гравцям розвивати свої навички у вирішенні головоломок та забезпечить час забави та релаксу.

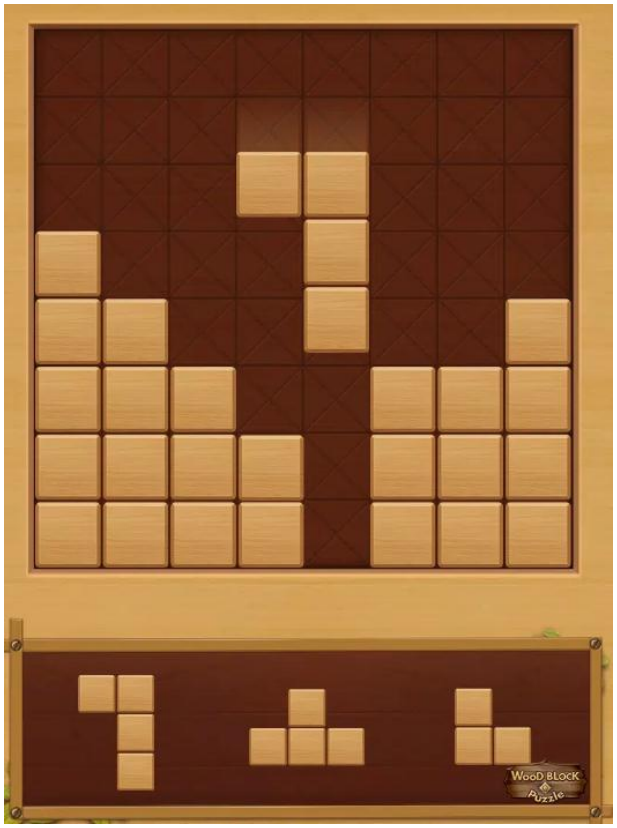


Рис. 1.4- Інтерфейс гри Block Puzzle

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

2.1. Призначення логічної гри Parking Logic Game

Призначення логічної гри Parking Logic Game полягає у наданні користувачам можливості розвитку своєї логіки та підвищенні рівня концентрації у веселій та захоплюючій формі гри. Гра дозволяє користувачам відчути себе водієм та розв'язувати завдання на керування автомобілем, що розташовує його на певній території. Головною метою гри є правильно розташувати автомобіль у паркінгу , так щоб вони не перекривали один одного та відкрили проїзд для автомобіля швидкої допомоги.

Паркінг має різні рівні складності, які відрізняються кількістю автомобілів та складністю їх розташування. Гравці повинні пройти кожен рівень, розв'язуючи логічні задачі та шукаючи оптимальне положення для кожного автомобіляу паркінгу. Гра розвиває логічне мислення, концентрацію, а також вміння швидко приймати рішення та вирішувати проблеми. Іноді необхідно буде правильно рухати один і той же автомобіль для відкриття шляху для інших.

Parking Logic Game підходить для гравців будь-якого віку та рівня підготовки, і може бути використана як розважальна гра для відпочинку, так і як інструмент для розвитку логіки та мислення. Крім того, гра дозволяє гравцям відчути себе водієм та збільшити рівень самодисципліни та терпіння.

2.2. Опис вимог до логічної гри Parking Logic Game

Перш за все, гра повинна працювати на всіх мобільних пристроях з ОС Android, які її підтримують. Гра повинна повністю відповідати поставленим вимогам, тобто мати всі необхідні функції. Гра повинна мати зручний інтерфейс, який не перевантажує користувача непотрібним текстом, а надає необхідну інформацію.

Розроблена гра повинна забезпечувати надійність, безпеку та цілісність даних, які вводить користувач.

Логіка гри повинна бути точною та відповідною. Якщо гра допустить помилку або будь-яку небажану поведінку, користувач повинен мати можливість зв’язатися з адміністратором гри. Будь-який мобільний пристрій з ОС Android, який може встановити гру та має доступ до Інтернету, є єдиною вимогою, яку потрібно виконати для користування грою.

2.3. Проектування функціональних можливостей гри

Мобільна гра Parking Logic Game, має наступні функціональні можливості:

- інтуїтивно зрозумілий та приємний інтерфейс, який дозволяє користувачам з легкістю орієнтуватися в грі;
- можливість переміщення автомобілів по заданій осі, не дозволяючи їм перетинатися;
- завдання полягає у вивезенні автомобіля швидкої допомоги, що додає грі додаткової складності;
- гра працює на всіх мобільних пристроях, що підтримують операційну систему Android або iOS;
- можливість збереження результатів гри та порівняння їх з результатами інших користувачів;
- швидкий відгук на дії гравця та надання відповідей на його запити.

Кожна сцена гри має різний функціонал, розглянемо їх:

- Головне меню

Рис. 2.1 Інтерфейс головного меню

Функціонал головного меню:

- Вибір елементів меню: на цій сторінці користувач може вибрати потрібну опцію, таку як "Грати", "Рейтинг", "Налаштування", "Про нас", "Вихід".
- Рух автомобілів на фоні для створення динамічності меню: під час вибору опцій користувачів зображення автомобілів можуть рухатися по головному меню, щоб створити більш динамічний вигляд
- Вибір необхідного рівня: користувач може вибрати потрібний рівень гри для початку гри, наприклад "Легкий", "Середній" або "Складний".
- Розблокування пройдених рівнів: користувач може переглянути, які рівні він вже пройшов, та розблокувати наступні рівні, які ще не доступні.
- Регулювання гучності музики та звуків гри: користувач може регулювати гучність музики та звуків гри окремо, використовуючи відповідні налаштування.
- Відкриття рейтингової сторінки: користувач може переглянути свій рейтинг або рейтинг інших гравців, які грають в гру.

• Ігрова сцена

Рис. 2.2 Інтерфейс ігрової сцени

Функціонал ігрової сцени:

- Переміщення автомобілів: користувач може переміщати автомобілі по ігровому полі шляхом натискання кнопок на екрані свого мобільного пристрою.
- Перевірка перетину автомобілів: гра перевірятиме, чи зіткнулися автомобілі в ігровому полі, і поверне повідомлення про це, якщо так.
- Тригер фінішу: як тільки автомобіль швидкої допомоги досягає певної позиції на полі, настає фініш гри. Гравець отримує повідомлення про успішне завершення рівня та переходить на екран з результатами.
- Ігрове поле: поле складається з сітки клітинок, які відображають місце розташування автомобілів. Гравець може переміщати автомобілі тільки в ті клітинки, які вільні. Клітинки, зайняті іншими автомобілями, блокуються.

До інших функціональних можливостей гри можна віднести наявність різноманітних рівнів складності з різними кількостями автомобілів на полі, можливість зміни кольору автомобілів та фону гри, наявність музики та звуків в грі, реалістична анімація переміщення автомобілів, можливість збереження прогресу та перезапуску гри.

2.4. Розробка блок - схем основних алгоритмів функцій гри

1. Функція MoveCar використовується для переміщення автомобіля по ігровому полю. Для переміщення спочатку отримується місце пересічення променю з ігровим полем, потім його необхідно округлити до сітки (Функція SnapToGrid). Для уникнення виходу автомобіля з ігрового поля та проходження його скрізь інші автомобілі необхідно спочатку перевірити чи є автомобілі на місця пересічення і тільки якщо поле не зайняте здійснити переміщення автомобіля.

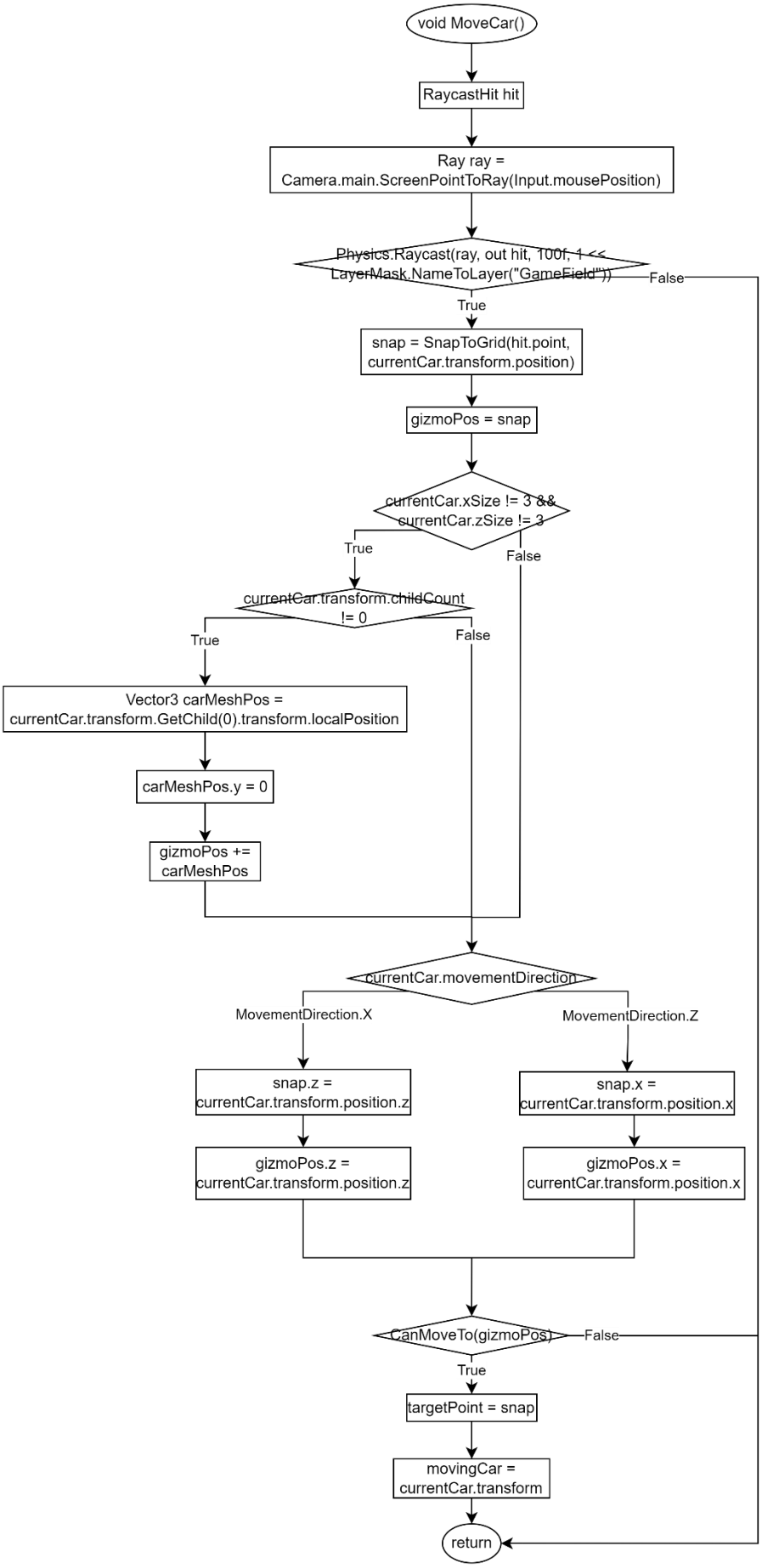


Рис. 2.3 Функція MoveCar

2. Функція *CanMoveTo* використовується для перевірки можливості руху до нової позиції автомобіля на ігровому полі. Вона перевіряє наявність колізій з іншими об'єктами, що мають шар "Collidable", у вказаній позиції.

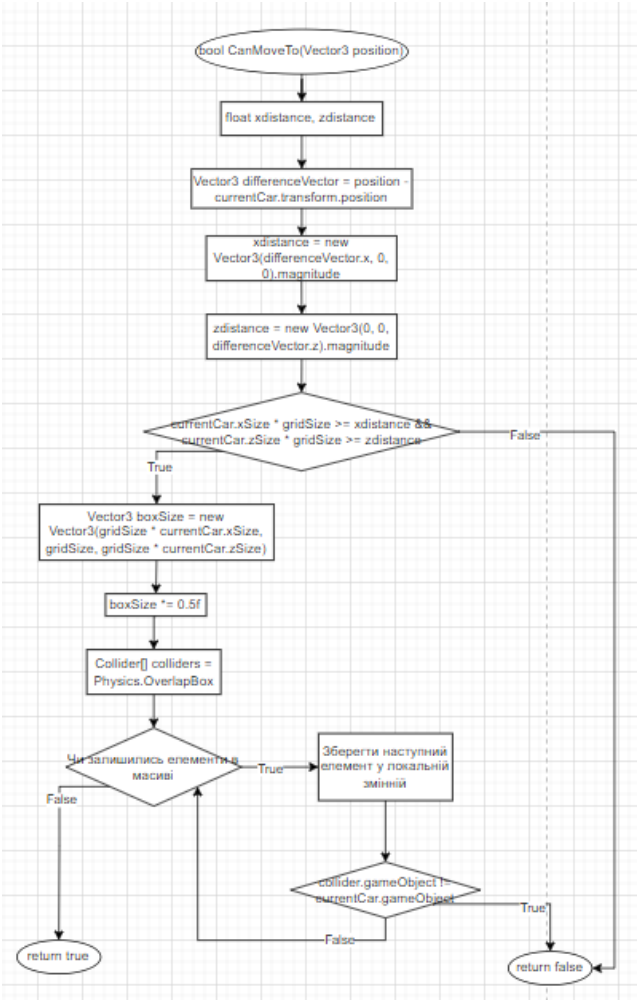


Рис. 2.4 Функція CanMoveTo

3. Функція **SmoothMovement** відповідає за плавний рух автомобіля. Основна ідея полягає в поступовому переміщенні автомобіля з його поточного положення до цільової точки. У цій функції спочатку перевіряється, чи є задана цільова точка (targetPoint) і чи є вказівник на рухомий автомобіль (movingCar). Якщо хоча б одна з цих умов не виконується, функція просто повертається.

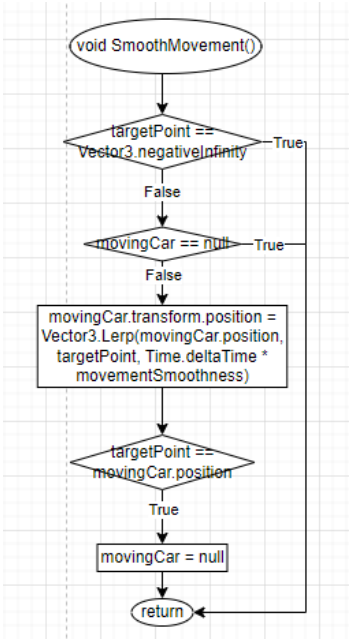


Рис. 2.3 Функція **SmoothMovement**

4. Функція SnapToGrid використовується для отримання точки прив'язки на сітці гри. Основна ідея полягає у тому, щоб округлити позицію об'єкту до найближчого значення на сітці заздалегідь визначеного розміру. Функція отримує позицію автомобіля в якості аргументу, потім за допомогою ‘Mathf.Round()’ округлює координати до найближчої позиції на сітці. Позиція за координатою Y залишається незмінною, щоб залишити автомобіль на тому самому рівні по висоті.

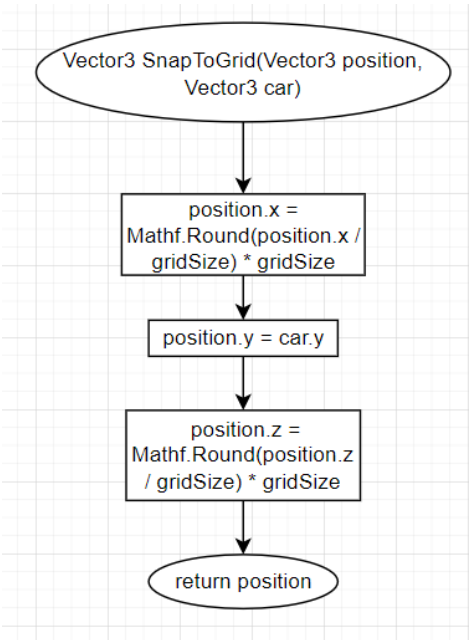


Рис.2.4 Функція SnapToGrid

2.5. Вибір засобів розробки

Під час створення дипломного проекту одним з ключових завдань є правильний вибір інструментів розробки. Після аналізу мов програмування, які зазвичай використовуються для створення логічних ігор, було визначено, що мова програмування C# є найбільш вдалим варіантом для даного проекту, оскільки вона є основною мовою програмування в Unity3D - популярному середовищі розробки ігор. Тому, для створення логічної гри, було вирішено використовувати C# у поєднанні з Unity3D.

C# (- це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Вона була створена в 2000 році як частина платформи .NET Framework і є однією з основних мов програмування для розробки додатків для Windows, мобільних пристроїв, ігор та веб-програмування.

C# поєднує в собі переваги мов C++ та Java, заснованих на синтаксисі C. Мова пропонує високу продуктивність, безпеку та надійність, а також має багато корисних функцій, таких як автоматичне управління пам'яттю, обробка винятків, делегати, лямбда-вирази та інші.

Основні переваги C#:

- Легкість вивчення та використання
- Переносимість програмного забезпечення на різні платформи
- Велика кількість готових бібліотек та фреймворків
- Підтримка візуального програмування за допомогою інтегрованого середовища розробки (IDE) Visual Studio
- Багата документація та активна спільнота розробників

Основні особливості мови програмування C# включають наступне:

- Синтаксис, схожий на синтаксис Java та C++. Це дозволяє розробникам, які мають досвід роботи з цими мовами, швидко вивчити C#.
- Підтримка об'єктно-орієнтованого програмування. C# дозволяє використовувати класи, об'єкти, успадкування, інтерфейси та інші конструкції ООП.

- Підтримка динамічної типізації. C# має вбудовану підтримку динамічних типів даних, що дозволяє зменшити кількість коду та покращити читабельність програм.
- Підтримка делегатів та лямбда-виразів. Це дозволяє зручно реалізувати функціональне програмування в C#.
- Підтримка LINQ (Language Integrated Query). LINQ є інтерфейсом для маніпулювання даними, що знаходяться в різних джерелах (наприклад, в БД, XML-файлах тощо), та дозволяє виконувати запити до цих даних у зручний спосіб.
- Підтримка асинхронного програмування. C# дозволяє розробникам писати асинхронний код з використанням ключових слів async та await, що покращує продуктивність та реактивність програм.
- Підтримка безпеки пам'яті. C# має вбудовані механізми для запобігання помилкам, пов'язаним з пам'яттю, таким як витоки пам'яті та помилки, пов'язані з багатопоточністю.

2.6. Вибір середовища розробки

2.6.1. Unity3D

Unity3D - це інтегроване середовище розробки ігор, що дозволяє розробникам створювати графічні додатки на різних платформах, таких як Windows, Mac, Android, iOS, Xbox, PlayStation і т.д. Основна мова програмування для Unity3D - це C#, що дозволяє створювати багатофункціональні ігри з багатим геймплеєм.

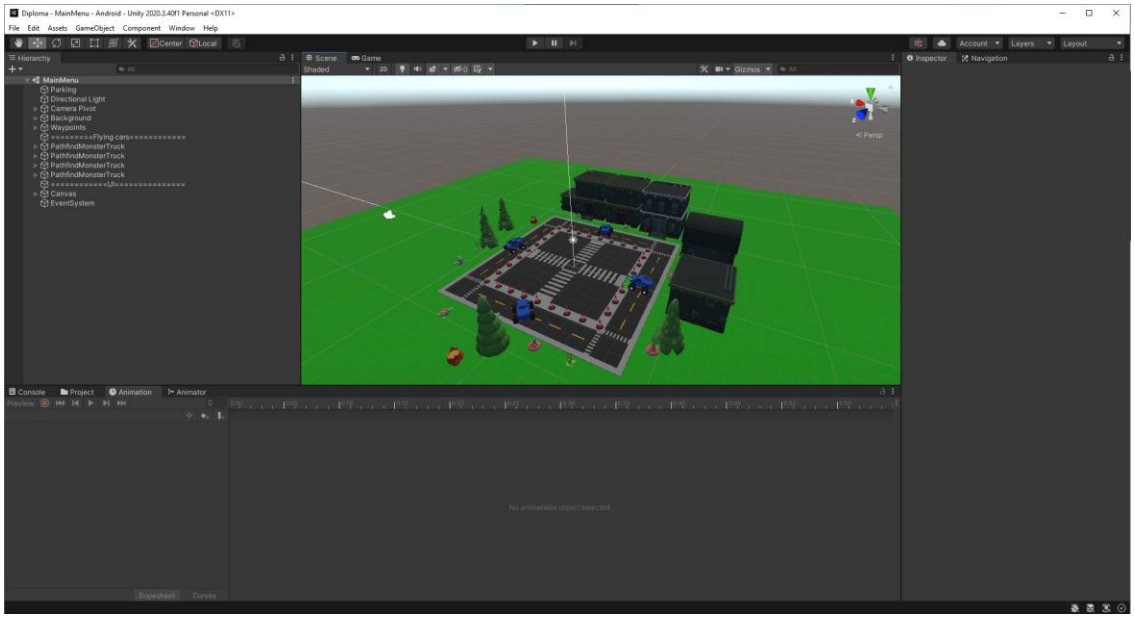


Рис. 2.1 - Інтерфейс Unity3D

Основні переваги Unity3D:

- Кроссплатформність: Unity3D підтримує розробку на більшості платформ, що дозволяє легко переносити розроблені проекти на інші платформи.
- Легка доступність: Unity3D є безкоштовним для особистого використання та має велику спільноту розробників, що дозволяє знайти відповіді на більшість питань та проблем під час розробки.
- Наявність великої кількості готових компонентів: Unity3D має багато вбудованих інструментів та готових компонентів, що дозволяє значно скоротити час розробки.
- Можливість розробки як 2D, так і 3D ігор: Unity3D підтримує розробку ігор у 2D та 3D форматах.

Основні недоліки Unity3D:

- Низька продуктивність: Unity3D не є найкращим вибором для розробки ігор з високою деталізацією та великою кількістю об'єктів на екрані.
- Обмежена можливість редагування компонентів: Іноді розробникам не вистачає можливостей редагувати вбудовані компоненти, що може

призводити до обмеження можливостей створення унікальних ефектів та механік. Однак, завдяки величезній кількості ресурсів та спільнот, що розвивається навколо Unity3D, допомогти у вирішенні складних проблем можна знайти в Інтернеті. Крім того, Unity3D пропонує платформонезалежний розробка, що означає, що гру можна розробляти один раз і запускати на різних платформах, таких як Windows, MacOS, Android, iOS та інші.

- Висока складність деяких аспектів: Деякі аспекти розробки в Unity3D можуть бути складними для розуміння та використання для новачків. Для створення повноцінної гри необхідно мати знання не тільки Unity3D, але й різних мов програмування, таких як C# або JavaScript. Крім того, великий обсяг документації може бути непростим для розуміння.

Також, Unity3D має вбудовану систему фізики, яка дозволяє створювати реалістичні ігрові середовища та поведінку об'єктів в них. Крім того, Unity3D підтримує віртуальну реальність та допомагає створювати VR-ігри.

Узагалі, Unity3D є потужним інструментом для створення ігор та віртуальних середовищ з багатим набором функцій та можливостей. Він підходить як для новачків, так і для досвідчених розробників, але потребує певного часу та зусиль для оволодіння всіма можливостями цієї платформи.

2.6.2. Microsoft Visual Studio

Microsoft Visual Studio є інтегроване середовище розробки (IDE), призначене для створення різноманітних програмних продуктів, включаючи десктопні додатки, веб-додатки, мобільні додатки та інші. Це один з найпопулярніших та найвикористовуваніших інструментів розробки програмного забезпечення в світі.

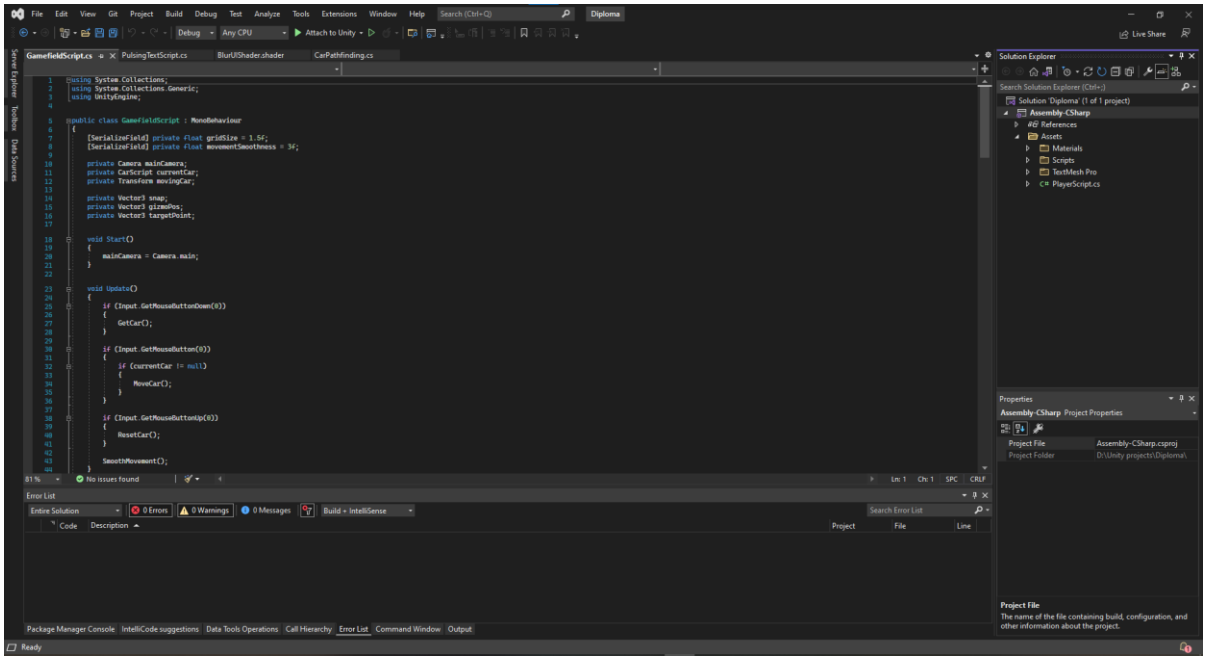


Рис. 2.2. - Інтерфейс Microsoft Visual Studio

Особливості Microsoft Visual Studio:

- Вбудований пакет для роботи з Git: Visual Studio має вбудований пакет для роботи з Git, що дозволяє розробникам зручно працювати з версійним контролем.
- Можливість створення інсталяторів: Visual Studio надає можливість створення інсталяторів для додатків, що дозволяє зручно встановлювати та розповсюджувати програмне забезпечення.
- Розширення: Visual Studio має велику кількість розширень, які дозволяють розширити функціональність середовища розробки та додати нові інструменти.
- Підтримка Azure: Visual Studio підтримує розробку для хмарних платформ Azure, що дозволяє розробникам створювати та розгортати додатки у хмарі.
- Live Share: Visual Studio має функцію Live Share, що дозволяє розробникам спільно працювати над кодом у реальному часі, незалежно від місцезнаходження.

- CodeLens: Visual Studio має функцію CodeLens, яка відображає додаткову інформацію про код, таку як авторство, історію змін та інші деталі, що допомагає розробникам зрозуміти структуру та історію проекту.

РОЗДІЛ 3 . РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Створення та налаштування Unity

Спочатку необхідно створити новий проект в Unity3D. Для цього необхідно в UnityHub натиснути на “New project”. Після чого необхідно ввести назву нового проекту в моєму випадку це “Diploma”.

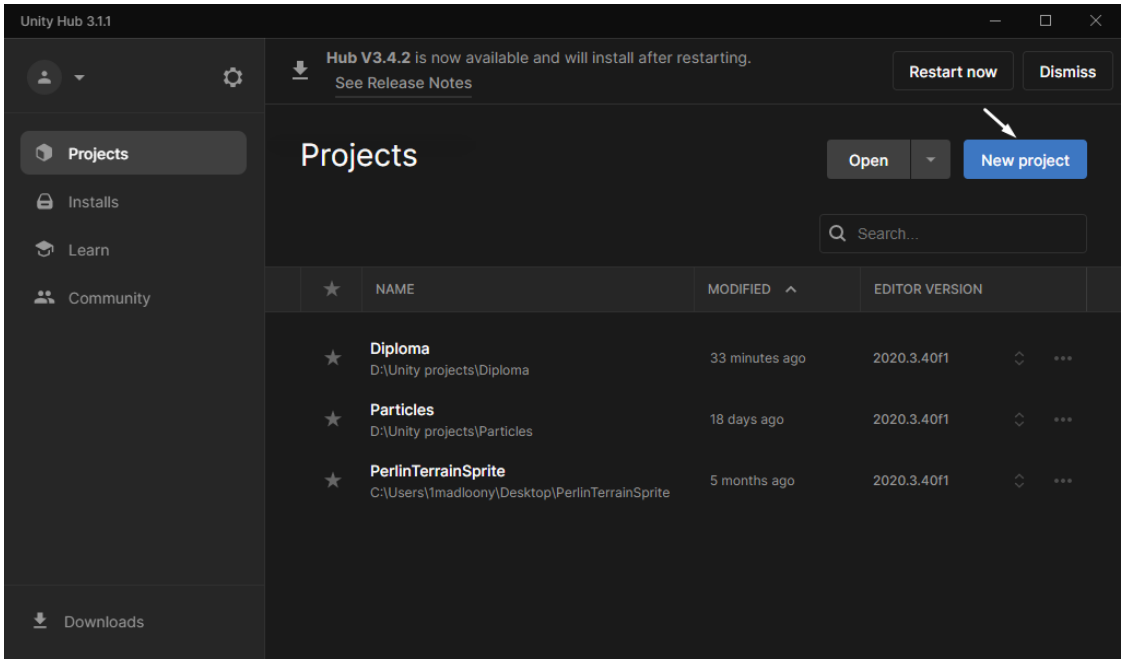


Рис. 3.1 - Інтерфейс UnityHub

Після створення нового проекту відкриється вікно Unity3D з порожньою сценою на якій буде лише камера та освітлення. Першим кроком в створенні проекту буде створення основних папок у вікні Project Manager. Це допоможе організувати ваш проект та легше знайти потрібні файли пізніше.

Натисніть правою кнопкою миші на панелі Project та виберіть опцію "Create Folder". Зазвичай створюють такі основні папки, як:

- Animations: для анімацій, які можуть бути пов'язані з персонажами, об'єктами та іншими елементами гри.
- Audio: для зберігання звукових ефектів, музики та іншого аудіоконтенту.
- Editor: для зберігання скриптів, які можуть використовуватися в редакторі Unity для автоматизації деяких задач.

- Materials: для зберігання матеріалів, які можуть бути застосовані до об'єктів, щоб надати їм текстуру та інші властивості.
- Models: для зберігання 3D-моделей, які можуть бути використані в грі.
- Prefabs: для зберігання префабів, які є готовими елементами, які можуть бути використані в різних частинах гри.
- Scenes: для зберігання сцен, які є окремими рівнями або ділянками гри.
- Scripts: для зберігання скриптів, які використовуються для програмування логіки гри.
- Textures: для зберігання текстур, які можуть бути застосовані до об'єктів та інших елементів гри.

Тепер можна додати готові 3D моделі автомобілів, природи (кущі, дерева та інше), будинків, доріг та самого паркінгу. Також додаємо файли звуків, музику та спрайти для UI. Розміщуємо всі елементи на сцені та створюємо Canvas для створення інтерфейсу користувача. На Canvas розміщуємо кнопку для відкриття внутрішньоігрового меню та текст для таймеру.

Після розстановки всіх об’єктів на сцені, створюємо файли скриптів та приєднуємо їх до ігрових об’єктів. На паркінг додаємо скрипт «GameField.c», а на Canvas - скрипт, який буде відповідати за вивід часу таймеру та обробку натискання на кнопку. Коли ми розставили та налаштували всі ігрові об’єкти зберігаємо сцену як SceneTemplate.

SceneTemplate - це вбудований механізм в Unity, що дозволяє зберігати об'єкти та налаштування сцени в шаблон, який можна використовувати для створення нових сцен.

За допомогою SceneTemplate ви можете створювати стандартні шаблони для різних типів сцен, що дозволяє швидко створювати нові сцени з необхідними налаштуваннями та об'єктами. Крім того, використання SceneTemplate дозволяє забезпечити єдність структури та налаштувань у всіх сценах вашого проекту.

Для створення SceneTemplate в Unity необхідно скомпонувати сцену з необхідними об'єктами та налаштуваннями, а потім зберегти цю сцену як шаблон

за допомогою команди "Create Scene Asset" у вкладці "Assets". Після цього, ви зможете створювати нові сцени на основі цього шаблону за допомогою команди "Create > Scene".

Це робиться для того щоб в подальшому можливо було створювати нові рівні швидко та кожного разу не розставляти об’єкти на сцені.

Наступним кроком ми створюємо префаби. У Unity3D префаб (Prefab) є об'єктом, який містить готову налаштовану копію об'єкта. Він дозволяє створити шаблон об'єкту з усіма його властивостями, компонентами та налаштуваннями, які можна використовувати в будь-якій сцені проекту. Це забезпечує швидке та просте створення декількох копій одного і того ж об'єкта без необхідності встановлювати всі налаштування знову.

Префаб можна редагувати і зберігати, щоб зберегти зміни у всіх його копіях, що були створені в різних сценах. Це дозволяє зменшити час на розробку та забезпечити більшу стабільність проекту, оскільки всі копії префабу будуть мати однакові налаштування та властивості. Крім того, префаби можна використовувати для збереження ресурсів, які часто використовуються в проекті, таких як моделі, текстури та звуки.

Необхідно створити префаби всіх автомобілів, для цього на нульових координатах створюємо порожній об’єкт та називаємо його таким чином, щоб було зрозуміло який саме це автомобіль. (Приклад на рисунку 8)

Рис. 3.2 – Створення префабу

Тепер додаємо на нього Collider та заздалегідь створений скрипт «CarScript.c». Для того щоб додати модель, перетягуємо її у вікно сцени, та у вікні інспектору прикріпляємо його до порожнього об’єкту префабу. Таким чином модель буде прикріплена до нашого префабу і рухатися за ним.

Для того щоб зберегти префаб і в подальшому використовувати його для створення рівнів його необхідно перетягнути у вікно Project Manager, в створену раніше папку Prefabs. Таким чином ми створюємо префаби для всіх автомобілів і тепер можемо приступати до написання скриптів.

3.2. Реалізація програмної частини

Програмна частина гри складається з 6 файлів скриптів:

- CarScript: Скрипт, який відповідає за налаштування певного автомобіля. (вісь його руху, його розмір за клітинками).
- FinishScript: Скрипт, який визначає коли автомобіль швидкої допомоги виходь за межі ігрового поля. Тим самим завершує рівень та відкриває меню завершеної гри.
- GameFieldScript: Головний скрипт, який оброблює натискання на екран, реалізовує механіку сітки руху, переміщує автомобілі.
- UIController: Скрипт обробки натискання кнопки для відкриття головного меню та виводу таймеру на екран.

- MenuController: Скрипт обробки переключення різних панелей меню(налаштування, вибір рівня та інше).
- CarPathfindingScript: Скрипт для переміщення автомобілів в головному між випадковими точками на ігровому полі.

Тепер необхідно більш детально розписати основні файли скриптів.

3.2.1. Створення ігрового поля та автомобілів

Для початку в скрипті ігрового поля необхідно оголосити змінні. Найменування змінних дуже важливий етап, так як в подальшому нам не одноразово прийдеться їх використовувати. Також це досить сильно допомагає в читанні кода іншими програмістами.

Ось список змінних, які знадобляться нам для створення скрипту ігрового поля, щоб автомобілі «прилипали», або краще сказати рухалися по ігровій сітці:

- gridSize: float - ця змінна визначає розмір однієї клітинки сітки в грі. Вона використовується для вирівнювання руху автомобілів та позиціонування їх на сітці.
- mainCamera: Camera - ця змінна зберігає посилання на головну камеру в грі. Вона використовується для отримання інформації про позицію миші та взаємодії з грою через курсор.
- currentCar: CarScript - ця змінна містить посилання на поточний автомобіль, який вибрав гравець або знаходиться під впливом дії гравця. Вона використовується для взаємодії з обраним автомобілем та виконання руху.
- snap: Vector3 - ця змінна зберігає координати точки прив'язки, до якої буде переміщений автомобіль. Вона використовується для обчислення нової позиції автомобіля на сітці під час руху.

Тепер створюємо метод Start, який є важливим для початкової налаштування об'єкта перед початком гри в Unity3D. У цьому методі ми будемо

отримувати посилання на головну камеру, що дозволить нам взаємодіяти з нею під час гри.

Для цього ми використовуємо функцію Camera.main, яка повертає посилання на активну камеру в сцені. Отримане посилання ми зберігаємо у змінній mainCamera, щоб мати доступ до функціональності камери. За допомогою цього посилання ми зможемо, наприклад, отримувати позицію курсора або виконувати інші дії, пов'язані з камерою, під час виконання гри.

Після цього додаємо метод Update, який є необхідним для постійного оновлення логіки гри на кожен кадр. У даному випадку, ми використовуємо його для взаємодії з автомобілями на ігровому полі.

У методі Update ми проводимо перевірку стану кнопок миші та викликаємо відповідні методи залежно від цих станів. Якщо кнопка миші була натиснута (Input.GetMouseButtonDown(0)), ми викликаємо метод GetCar() для отримання автомобіля. Якщо кнопка миші утримується (Input.GetMouseButton(0)), і змінна currentCar не є null, то ми викликаємо метод MoveCar() для переміщення автомобіля. Нарешті, якщо кнопка миші була відпущена (Input.GetMouseButtonUp(0)), ми викликаємо метод ResetCar() для скидання поточного автомобіля.

У методі MoveCar() ми виконуємо рух автомобіля згідно з рухом миші на ігровому полі. Спочатку ми використовуємо промінь (Ray) та луч (Raycast), щоб визначити, на що вказує курсор миші. Метод ScreenPointToRay() дозволяє отримати луч, що починається з позиції курсора миші на екрані та прямує вздовж вісі Z у тривимірному просторі.

Наступною кроком є використання методу Physics.Raycast(), який перевіряє, чи перетинає луч колайдер об'єкта. У нашому випадку, ми шукаємо колайдер з ім'ям шару "GameField". Якщо луч зіткнувся з таким колайдером, ми отримуємо точку зіткнення (hit.point) на поверхні гри.

Наступним кроком є виклик методу SnapToGrid(), який вирівнює отриману точку зіткнення на сітці гри. Цей метод використовується для обмеження руху автомобіля до певних позицій на сітці, що розділяє ігрове поле. Отримана точка прив'язки (snap) буде мати округлені координати відповідно до розміру сітки.

Далі, залежно від напрямку руху поточного автомобіля (currentCar.movementDirection), ми вносимо корекції до координат точки прив'язки. Наприклад, якщо автомобіль може рухатись тільки по осі X, ми залишаємо координату Z незмінною, а якщо рух дозволений тільки по осі Z, то залишаємо координату X незмінною.

Коли ми перевірили можливість руху до нової позиції за допомогою методу CanMoveTo() і переконалися, що автомобіль може пересунутися на цю позицію, ми встановлюємо нову позицію автомобіля за допомогою currentCar.transform.position = snap. Це зміщення автомобіля на нову точку прив'язки, яку ми розрахували раніше.

Отже, метод MoveCar() виконує наступні дії:

1. Визначає точку зіткнення миші з об'єктом шару "GameField" за допомогою променя та луча.
2. Викликає метод SnapToGrid(), щоб отримати точку прив'язки на сітці гри.
3. Виправляє координати точки прив'язки залежно від напрямку руху поточного автомобіля.
4. Перевіряє можливість руху до нової позиції за допомогою методу CanMoveTo().
5. Якщо можливо, переміщує автомобіль на нову позицію шляхом оновлення його координат currentCar.transform.position.

Таким чином, метод MoveCar() відповідає за рух автомобіля на ігровому полі, забезпечуючи обмеження руху на сітці та перевіряючи можливість руху до нової позиції.

Функція *SnapToGrid* використовується для прив'язки позиції автомобіля до сітки на ігровому полі. Основна мета цієї функції - змінити координати позиції автомобіля таким чином, щоб вони були округлені до найближчих значень розміру сітки. Приблизний алгоритм прикріплення автомобіля по сітці:

Спочатку функція приймає два параметри: position, який представляє позиція миші на ігровому полі, і **car**, який містить інформацію про автомобіль.

Наступний кроком ми округлюємо координати за допомогою функції Mathf.Round, координати X і Z позиції автомобіля (position.x і position.z) округлюються до найближчого значення розміру сітки (gridSize). Це забезпечує прив'язку автомобіля до точок на сітці. Координата Y позиції автомобіля залишається без змін, зі збереженням значення змінної car.y. Це необхідно для збереження оригінальної висоти автомобіля після прив'язки до сітки.

Після цього оновлені значення координат *position* повертаються як результат функції *SnapToGrid,* що дозволяє їх використання в подальшому коді для переміщення автомобіля до нової прив'язаної позиції на сітці.

Таким чином, функція *SnapToGrid* виконує необхідні обчислення для прив'язки позиції автомобіля до сітки, забезпечуючи його рух по фіксованих точках на ігровому полі.

Функція *CanMoveTo* використовується для перевірки можливості руху до нової позиції автомобіля на ігровому полі. Вона перевіряє наявність колізій з іншими об'єктами, що мають шар "Collidable", у вказаній позиції.

Для цього спочатку створюємо новий тривимірний вектор *boxSize,* який визначає розмір об'ємної коробки, яка охоплює автомобіль. Розміри коробки визначаються залежно від розміру сітки (gridSize) та змешуємо її на половину (boxSize *= 0.5f), оскільки центр коробки буде співпадати з позицією автомобіля.

Тепер використовуємо функцію *Physics.OverlapBox* для отримання масиву колайдерів, які перетинаються з об'ємною коробкою, розташованою у вказаній позиції. Метод *OverlapBox* приймає параметри: позицію (position), розмір

коробки (boxSize), обертання (currentCar.transform.rotation) та шар (LayerMask) з ім'ям "Collidable".

Далі у циклі foreach перевіряється кожен колайдер з отриманого масиву. Якщо колайдер належить іншому об'єкту, відмінному від поточного автомобіля (collider.gameObject != currentCar.gameObject), це означає наявність перешкоди в новій позиції, тому повертається значення *false.* Якщо всі перевірки пройшли успішно, тобто не виявлено колізій з іншими об'єктами, функція повертає значення *true*, що свідчить про можливість руху до нової позиції.

Отже, функція *CanMoveTo* допомагає забезпечити контроль над рухом автомобіля, перевіряючи наявність перешкод у новій позиції на ігровому полі.

Функція *GetCar* використовується для отримання посилання на об'єкт автомобіля, на який наведено курсор миші.

Спочатку створюється промінь (ray) за допомогою Camera.main.ScreenPointToRay, який відповідає променю, що починається в позиції миші на екрані і простягається у світі гри. Використовуючи Physics.Raycast, перевіряється, чи перетинається промінь з яким-небудь колайдером у максимальній відстані 100f. Результат перевірки зберігається в hit. Якщо промінь зіткнувся з яким-небудь колайдером, перевіряється, чи об'єкт колайдера не є порожнім (hit.collider.gameObject != null). Якщо об'єкт колайдера має компонент CarScript, він зберігається в змінну hitCar, яка є типом CarScript. Значення hitCar присвоюється змінній currentCar, що означає, що поточний автомобіль встановлюється на автомобіль, на який наведено курсор миші.

Отже, функція *GetCar* допомагає визначити поточний автомобіль, на який наведено курсор миші, і зберегти посилання на нього для подальшої роботи з ним.

Функція *ResetCar* використовується для скидання поточного автомобіля (currentCar) на значення null. Основна мета цієї функції полягає в тому, щоб вказати, що поточного автомобіля не вибрано, і відновити початковий стан

програми, коли необхідно скинути обраний автомобіль. Змінна currentCar встановлюється на значення null, що означає, що поточний автомобіль не вибрано. Це дозволяє скинути вибір об'єкта автомобіля і показує, що не існує обраних автомобілів.

enum (перерахування) є типом даних в багатьох програмувальних мовах, включаючи C#. Воно дозволяє визначити новий тип, що складається з певного набору значень, які можуть бути присвоєні змінним. Кожне значення в перерахуванні має свій власний ідентифікатор.

У вище наведеному коді визначено перерахування MovementDirection, яке містить три можливі значення: X, Z і Free. Кожне значення представляє напрямок руху автомобіля.

Використання перерахування дозволяє визначити конкретні значення, з якими можна працювати в коді, замість використання числових значень або рядків. Це полегшує зрозуміння коду і запобігає можливим помилкам, пов'язаним з помилковими значеннями.

У коді вище, перерахування MovementDirection використовується для визначення напрямку руху автомобіля в різних частинах програми. Наприклад, в методі MoveCar(), поточний напрямок руху автомобіля (currentCar.movementDirection) може бути перевірений для визначення, які координати потрібно змінити. Це дозволяє зробити код більш зрозумілим та логічним, оскільки можна працювати зі значеннями X або Z замість безладного числового коду або рядків.

Клас CarScript є класом компонента Unity, який успадковується від класу MonoBehaviour. Цей клас використовується для представлення автомобіля в грі.

У класі CarScript визначено приватне поле _movementDirection з атрибутом [SerializeField], яке представляє напрямок руху автомобіля. Використання атрибута [SerializeField] дозволяє відображати це поле в інспекторі Unity, щоб його можна було налаштувати безпосередньо в редакторі.

Крім того, в класі визначено властивість movementDirection, яка забезпечує доступ до значення поля _movementDirection. Властивість є доступною тільки для читання (get), що означає, що інші частини програми можуть отримати значення напрямку руху автомобіля, але не можуть його змінити.

Клас CarScript використовується для надання інформації про напрямок руху автомобіля і надає доступ до цієї інформації іншим частинам програми. В інших частинах програми, наприклад, у методі MoveCar(), можна отримати значення напрямку руху поточного автомобіля, використовуючи currentCar.movementDirection. Це дозволяє виконувати різні дії в залежності від напрямку руху автомобіля. Наприклад, змінювати координати руху по осі X або Z, або виконувати певні операції, специфічні для кожного напрямку руху. Клас CarScript допомагає узгоджувати інформацію про напрямок руху автомобіля та забезпечує зручний доступ до цієї інформації в інших частинах програми.

3.2.3. Розробка скриптів, функції для створення багатокліткових автомобілів

Слідкуючи блок-схемам та заздалегідь спроектованому алгоритму починаємо доповнюємо існуючі функції та створюємо нові.

1. Функцію **MoveCar** доповнимо наступним чином:

Додаємо змінну gizmoPos. Ця змінна використовується для збереження позиції сіткової точки, до якої буде переміщено автомобіль. Її значення обчислюється на основі snap та додаткових перераховувань, враховуючи положення меша автомобіля, якщо воно є присутнім.

Додаємо умову, яка перевіряє розміри автомобіля (currentCar.xSize і currentCar.zSize). Якщо розміри дорівнюють 3 (як в даному випадку), то виконується додаткова логіка. В даному випадку, якщо у автомобіля є дочірній об'єкт, знаходиться його позиція меша та додається до gizmoPos, при цьому Y-координата встановлюється на 0. Зміни значення gizmoPos в залежності від напрямку руху: В розділі switch додано зміни значення координат gizmoPos відповідно до напрямку руху автомобіля (currentCar.movementDirection).

Замість безпосереднього встановлення позиції автомобіля (currentCar.transform.position = snap), введемо нові змінні targetPoint і movingCar. Ці змінні використовуються для вказівки мети переміщення та визначення, який саме автомобіль буде переміщено. Це дає можливість використовувати інші частини коду для актуалізації позиції автомобіля.

Усі ці зміни допомагають покращити логіку переміщення автомобіля, враховуючи додаткові умови та параметри, такі як розмір автомобіля.

2. Функцію **CanMoveTo** доповнимо наступним чином:

Введемо змінні xdistance і zdistance, які обчислюють відстань між позицією position та поточною позицією автомобіля currentCar.transform.position по осях X та Z відповідно. Це здійснюється шляхом створення векторів з різниці координат та виміру їх відстані (magnitude).

Змінимо умову перевірки на руху до нової позиції. Замість безпосереднього порівняння розмірів автомобіля з розміром об'єкта колайдера, застосовується перевірка відстаней xdistance і zdistance відповідно до розмірів автомобіля (currentCar.xSize * gridSize та currentCar.zSize * gridSize). Це дозволяє точніше визначити, чи вистачає місця для руху автомобіля в нову позицію. У разі виконання перевірки відстаней, якщо вистачає місця для руху, обчислюється boxSize аналогічно до початкової версії коду. Потім за допомогою Physics.OverlapBox перевіряється наявність колайдерів в заданій області position, використовуючи SizeScale для масштабування розміру коробки. Якщо знайдено колайдер, який не належить поточному автомобілю, повертається значення false. Якщо відстані xdistance або zdistance не відповідають розмірам автомобіля, виконується оператор return false, оскільки рух до нової позиції неможливий.

Створюємо нову функцію **SizeScale**.

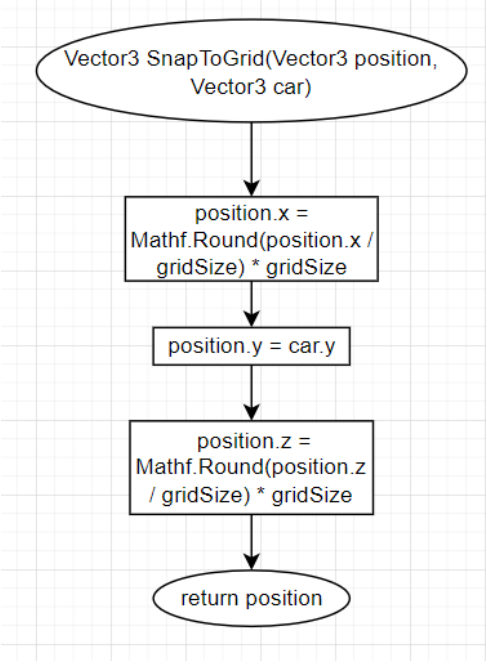


Рис.3.3 Функція SnapToGrid

Ця функція застосовує масштабування розміру вхідного вектора input в залежності від розмірів автомобіля currentCar. Вона використовується для зміни розміру області, яка перевіряється на колізії за допомогою функції Physics.OverlapBox.

Основна мета цієї функції - зменшити розмір області перевірки колізій, коли автомобіль має розмір менше 2x2 клітинок на полі. Зменшення розміру області допомагає врахувати розмір самого автомобіля при перевірці колізій і уникнути неправильних результатів. У функції спочатку перевіряється розмір автомобіля currentCar по осях X і Z. Якщо розмір перевищує 1, значення відповідної координати вхідного вектора input множиться на 0.9f, в іншому випадку на 0.75f. Це здійснює масштабування розмірів області перевірки колізій відповідно до розмірів автомобіля.

Отриманий масштабований вектор input повертається з функції для подальшого використання в функції CanMoveTo, де він використовується при перевірці колізій автомобіля з іншими об'єктами.

Також, для плавного переміщення автомобіля ми доповнили функцію **Update**, додавши до неї виклик функції **SmoothMovement.**

3. Функція **SmoothMovement** відповідає за плавний рух автомобіля. Основна ідея полягає в поступовому переміщенні автомобіля з його поточного положення до цільової точки. У цій функції спочатку перевіряється, чи є задана цільова точка (targetPoint) і чи є вказівник на рухомий автомобіль (movingCar). Якщо хоча б одна з цих умов не виконується, функція просто повертається.

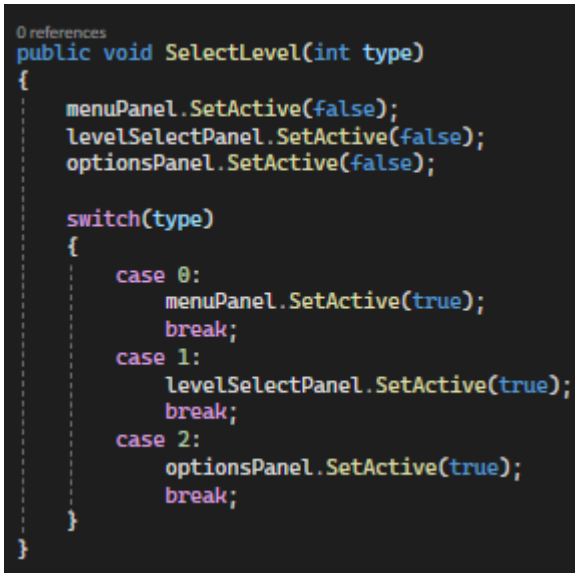
Потім застосовується плавне переміщення з поточного положення автомобіля до цільової точки за допомогою Vector3.Lerp. Цей метод використовується для виконання плавного інтерполяції між двома точками з використанням часу, представленого Time.deltaTime і параметра movementSmoothness. Після кожного кадру перевіряється, чи автомобіль досягнув цільової точки. Якщо так, то цільова точка скидається на початкове значення Vector3.negativeInfinity, а вказівник на рухомий автомобіль стає null, що показує, що рух завершено.

Таким чином, функція **SmoothMovement** забезпечує плавний рух автомобіля від його поточного положення до заданої цільової точки з використанням інтерполяції.

Після внесення коректив до початкового коду програми ми можемо створювати автомобілі з багатоклітковим розміром та їх плавним переміщенням. Нам залишається лише створити префаби цих автомобілів та розставити їх на сценах різних рівнів.

3.2.4. Створення ігрової логіки для меню та завантаження рівнів

1. Файл **MenuController** відповідає за керування меню гри. Створюємо три змінні-поля, що вказують на об'єкти панелей меню: menuPanel, LevelSelectPanel та optionsPanel.



```
0 references
public void SelectLevel(int type)
{
    menuPanel.SetActive(false);
    levelSelectPanel.SetActive(false);
    optionsPanel.SetActive(false);

    switch(type)
    {
        case 0:
            menuPanel.SetActive(true);
            break;
        case 1:
            levelSelectPanel.SetActive(true);
            break;
        case 2:
            optionsPanel.SetActive(true);
            break;
    }
}
```

Рис 3.4 Функція SelectLevel

Створюємо публічну функцію **SelectLevel**. Ця функція приймає параметр type, який вказує на тип панелі меню, яка має бути активована. Залежно від значення type, вона активує відповідну панель меню та деактивує інші панелі.

Функція switch(type) здійснює перевірку значення type і включає певну панель меню, залежно від вибраного типу:

- case 0: Активує menuPanel.
- case 1: Активує levelSelectPanel.
- case 2: Активує optionsPanel.

Ця функція дозволяє змінювати панель меню, вибираючи потрібну опцію залежно від переданого типу.

Також для виходу з гри необхідно створити функцію **EndGame.**

Ця функція викликається при натисканні кнопки "вийти з гри". Вона використовує Application.Quit() для закриття програми або гри. Таким чином, вона припиняє виконання програми та виходить з неї.

Таким чином ми створили клас **MenuController**, який повністю задовольняє потребам контролеру меню гри.

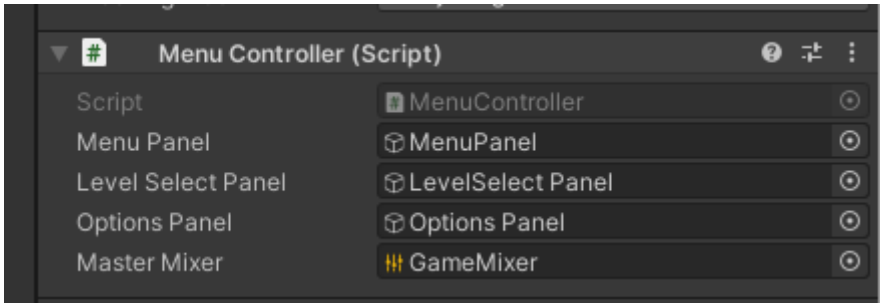


Рис 3.5 Скрипт MenuController прикріплений до ігрового об’єкта UI

2. Створюємо функцію **LoadScene.** Ця функція, відповідає за завантаження рівня гри залежно від переданого ідентифікатора сцени (sceneID).

Для завантаження рівня ми використовуємо функцію SceneManager.LoadScene. Це функція з Unity API, яка використовується для завантаження сцен. Вона отримує параметр sceneName (назва сцени) та loadMode (режим завантаження сцени).

LoadSceneMode.Single - це значення перерахування з Unity API, що вказує на режим завантаження сцени. Single означає, що сцена буде завантажена і замінить поточну активну сцену.

Таким чином, коли ця функція викликається з певним значенням sceneID, вона формує назву сцени за допомогою рядка-шаблону, наприклад, якщо sceneID дорівнює 1, то назва сцени буде "level1". Потім вона використовує SceneManager.LoadScene() для завантаження цієї сцени в режимі Single, замінюючи поточну активну сцену на нову.

Ця функція дозволяє динамічно завантажувати рівні гри на основі переданого ідентифікатора сцени, що полегшує розширення та керування рівнями у вашій грі.

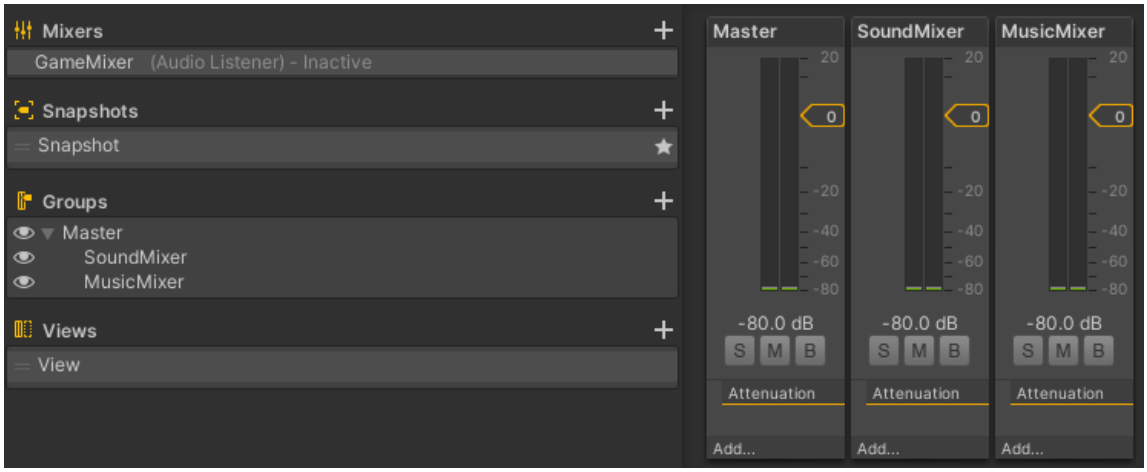


Рис. 3.6 Звуковий мікшер гучності звуків та музики у грі

3. Створення функцій **SetSound та SetMusic**. Ці функції відповідають за керування гучністю звуку для звукових ефектів та музики в грі. Вони використовуються для зміни гучності звуку на основі значення, отриманого з компонента Slider через параметр value.

Функція masterMixer.SetFloat("Sound", volume) коду використовує API аудіо мікшера Unity для зміни гучності звукових ефектів. "Sound" вказує на параметр гучності, який треба змінити в аудіо мікшері. volume - це значення гучності, отримане з компонента Slider. Передавши нове значення гучності, ви змінюєте рівень гучності для звукових ефектів.

Функція masterMixer.SetFloat("Music", volume) коду аналогічний попередньому, але змінює гучність музики в аудіо мікшері. "Music" - це параметр гучності музики, який треба змінити.

Компоненти Slider використовуються для взаємодії з користувачем та регулювання гучності звуку в реальному часі. Коли користувач переміщає повзунок Slider, значення його параметра value змінюється. Ці значення передаються у функції SetSound(float volume) та SetMusic(float volume), де вони використовуються для зміни гучності звуку за допомогою аудіо мікшера.

В результаті, використовуючи ці функції, ви можете налаштувати гучність звукових ефектів та музики відповідно до значень, встановлених користувачем за допомогою Slider.

ВИСНОВКИ

Дипломна бакалаврська робота з питань розробки програмного забезпечення для логічної гри в мобільному додатку "Parking Logic Game" була успішно виконана. Проект пройшов через різні етапи, починаючи з дослідження предметної області і вивчення загальних принципів роботи мобільних додатків та логічних ігор.

У першому розділі була проведена аналітична робота, де було вивчено платформу мобільних додатків, а також загальні принципи роботи логічних ігор. Було проведено порівняльний аналіз існуючих логічних ігор для встановлення функціональних вимог до розроблювального додатку.

У другому розділі були описані призначення та вимоги до розроблювального додатку "Parking Logic Game". Було розглянуто вибір мови програмування, середовища розробки, бази даних та огляд інтерфейсу для реалізації функціоналу гри. Відповідний вибір складових сприяє успішній реалізації проекту і досягненню бажаного результату.

У третьому розділі описані етапи розробки додатку "Parking Logic Game". Починаючи з його створення в мобільному середовищі, було охоплено різні аспекти розробки, включаючи логіку гри, інтерфейс користувача, обробку вхідних даних та зберігання рекордів у базі даних. Всі етапи розробки були успішно пройдені і випробувані для забезпечення якості і готовності додатку до використання.

Результатом проєкту є логічна мобільна гра "Parking Logic Game", яка повністю задовольняє описані вимоги та особливості. Гра пропонує користувачам виконувати логічні завдання різних рівнів складності та може допомогти швидко скоротати час очікування. Додаток забезпечує безперебійну роботу і може взаємодіяти з багатьма користувачами одночасно.

Мобільна гра "Parking Logic Game" має перспективу для подальшого розвитку і розширення функціоналу. Це дасть можливість оптимізувати іполегшити вирішення ще більшого спектру логічних завдань в мобільних додатках.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Апаратне та програмне забезпечення ПК [Електронний ресурс] - Режим доступу: http://eprints.zu.edu.ua/18/1/Konspect_modul_1_Windows.pdf.

2. Керівництво для настільних ПК [Електронний ресурс] - Режим доступу:https://docs.microsoft.com/enus/dotnet/desktop/wpf/overview/?view=netdesktop-5.0.

3. Гуманітарний портал [Електронний ресурс] - Режим доступу: https://gtmarket.ua/concepts/7091. Unity Game Development Cookbook - Paris Buttfield-Addison, Jon Manning, Tim Nugent, 2014.

4. Пасічник В. В. Організація баз даних і знань. / Пасічник В.В., Резніченко В.А. – К.: BHV, 2006.

5. Кравець П.о. К 77 об'єктно - орієнтоване програмування: навч. посібник / П.О. Кравець. - Львів: Видавництво Львівської політехніки, 2012.

6. Joe Hocking , Unity in Action: Multiplatform Game Development in C# - // Joe Hocking,- 2018.

7. Alan Thorn , Pro Unity Game Development with C#/ Alan Thorn,- 2014.

8. C# для початківців [Електронний ресурс] - Режим доступу: https://www.cyberforum.ua/csharp-beginners/thread2471856.html

9. P. Doran Unity Game Development Blueprints / John P. Doran- 2014.

10. DataSets, DataTables, and DataViews [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/dataset-datatable-dataview/.

11. Learning Unity Android Game Development - Thomas Finnegan, 2017.

12. Основні команди SQL | Trroger [Електронний ресурс] Режим доступу: https://trroger.ua/translations/sql-recar/.

13. Simon Jackson , Mastering Unity 2D Game Development -/Simon Jackson, -2016

14. Керівництво з програмування на C# [Електронний ресурс] Режим доступу: https://msdn.microsoft.com

15. Maciej Szczesnik , Unity 5.x Animation Cookbook" / Maciej Szczesnik, 2016.

16. ADO.NET | DataSet - Professor Web [Електронний ресурс] Режим доступу: https://professorweb.ua/my/ADO_NET/base/level2/2_1.php.

17. Unity UI Cookbook - Francesco Sapio, 2018.

18. Бази данних і мова SQL [Електронний ресурс] Режим доступу: https://function-x.ua/sql_join.html

19. Unity 2D Game Development Cookbook - Claudio Scolastici, 2015.

20. DataSet In C# - C# Corner [Електронний ресурс] Режим доступу: https://www.c-sharpcorner.com/article/dataset-in-C-Sharp/.

21. Unity 3D and PlayMaker Essentials: Game Development from Concept to Publishing - Jere Miles, 2016.

22. Пауерс Л., Снелл М. Microsoft Visual Studio 2008 = Microsoft Visual Studio 2008.

23. Створення додатків Windows Forms у Visual Studio з C # [Електронний ресурс] - Режим доступу: https://docs.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2019

24. Unity 2018 Shaders and Effects Cookbook: Transform your game into a visually stunning masterpiece with over 70 recipes - John P. Doran, 2018.

25. DataSet и DataTable [Електронний ресурс] Режим доступу: https://metanit.com/sharp/adonet/3.6.php.

26. Buttfield-Addison , Unity 2020 Cookbook: Over 160 recipes to take your 2D and 3D game development to the next level /Paris Buttfield-Addison, Jon Manning, Tim Nugent, Paris -2020.

27. "Unity Game Development Cookbook: Essentials for Every Game" - Paris Buttfield-Addison, Jon Manning, Tim Nugent, 2019.

28. Unity 2D Game Development Cookbook: Over 50 hands-on recipes that leverage the features of Unity to help you create 2D games and game prototypes - Claudio Scolastici, 2020.

29. Троелсен, Ендрю. Т70 Мова програмуванняC# 2010 і платформа .NET 4.0, 5 - е изд. : Пер. с англ. : ооо " Вільяме", 2011.

30. DataSet Class [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.data.dataset?view=net-5.0.

31. ADO.NET | DataSet - Professor Web [Електронний ресурс] Режим доступу: https://professorweb.ua/my/ADO_NET/base/level2/2_1.php.

32. DataGridView Class (System.Windows.Forms) | Microsoft Docs [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.datagridview?view=net-5.0.

33. Зубенко В. В. Програмування : рубрика довідки (свідоцтво Міністерства освіти і науки України) / В. В. Зубенко, Л. Д. Омельчук. – К. : ВПЦ «Київський університет», 2012.

34. Ісаченко А.Н. Моделі даних і СУБД / О.М. Ісаченко, Л.В. Бондаренко - БДУ, 2010.

35. Елемент управління dataGridView | BestProg [Електронний ресурс] Режим доступу: https://www.bestprog.net/ua/2018/02/17/the-datagridview-control_ua/.

36. Reading Data from Data Grid View [Електронний ресурс] Режим доступу: https://stackoverflow.com/questions/11469675/reading-data-from-data-grid-view.

37. Database and mov SQL [Електронний ресурс] Режим доступу: https://function-x.ua/sql_join.html.

38. DataSet Class [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/ai/system.data.dataset?view=net-5.0.

39. DataSets, DataTables і DataViews [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/dataset-datatable-dataview.