МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка гри для створення та проведення D&D історій на рушії Unity»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Федина Владислав Юрійович

Керівник: кандидат технічних наук, доцент кафедри інформаційних технологій та аналітики даних
Шатний Сергій В'ячеславович

Рецензент: кандидат психологічних наук, доцент, доцент кафедри інформаційних технологій та аналітики даних
Зубенко Ігор Ростиславович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри інформаційних технологій та аналітики даних _____
проф., д.е.н. Кривицька О.Р.
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня бакалавра

Тема: *Розробка гри для створення та проведення D&D історій на рушії Unity.*

Автор: *Федина Владислав Юрійович*

Науковий керівник: *Шатний С.В., кандидат технічних наук, доцент кафедри ЕММІТ.*

Захищена «……»…… 2025 року.

Пояснювальна записка до кваліфікаційної роботи:

Ключові слова: *Unity, D&D, Dungeon and Dragons, ігрові сцени, D&D історії.*

Короткий зміст праці:

У цій кваліфікаційній роботі було створено гру для реалізації настільної гри Dungeon and Dragons за допомогою ігрового рушія Unity. У цій роботі були використані такі програмні засоби, як ігровий рушій Unity, мова програмування C#, середовище програмування Visual Studio. Метою створення гри було розробити внутрішньо-ігрове середовище для повної підготовки та проведення ігрових сюжетів гри Dungeon and Dragons. Під час створення проєкту була використана велика кількість механік взаємодії з ігровими рівнями, як для побудови ігрової сцени, так і для взаємодії з її елементам вже під час ігровго процесу. Також необхідною частиною гри став її інтерфейс, за допомогою якого гравці можуть створювати власні ігрові сцени, об'єднувати їх у готові історії та проводити їх для свого кола гравців.

ANNOTATION
of qualification paper
for bachelor's degree

Theme: *Development of a game to create and run a D&D stories using the Unity engine*

Author: *Vladyslav Fedyna*

Scientific supervisor: *Shatnyi S. candidate of technical sciences, associate professor of the department of EMMIT.*

Defensed «……»…… of 2025.

Explanatory note to the qualification work:

Keywords: *Unity, D&D, Dungeon and Dragons, game scenes, D&D stories.*

Summary of the paper:

In this qualification, a game was created to implement the Dungeon and Dragons board game using the Unity game engine. This work used such software tools as the Unity game engine, the C# programming language, and the Visual Studio programming environment. The goal of creating the game was to develop an in-game environment for the complete preparation and execution of game plots of the Dungeon and Dragons game. During the creation of the project, a large number of mechanics of interaction with game levels were used, both for building the game scene and for interaction with its elements already during the game process. Also a necessary part of the game became its interface, with which players can create their own game scenes, combine them into ready-made stories and conduct them for their circle of players.

ЗМІСТ

КАЛЕНДАРНИЙ ПЛАН **4**
ANNOTATION **6**
ВСТУП **9**
РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ І ЗАСОБІВ РЕАЛІЗАЦІЇ **11**
1.1 Концепція настільної гри D&D(Dungeon and Dragons) **11**
1.2 Аналіз прототипів та порівняння **16**
1.3 Вибір програмних інструментів реалізації **18**
1.4 Порівняльна характеристика інструментів реалізації 22
Висновки до розділу 1 **24**
РОЗДІЛ 2. ПРОЄКТУВАННЯ СТРУКТУРИ ТА ФУНКЦІОНАЛЬНИХ СКЛАДОВИХ ІГРОВОГО ЗАСТОСУНКУ **25**
2.1. Постановка задачі **25**
2.2 Архітектура та механіки ігрового застосунку **26**
2.3 Рольові моделі гравців і їх персонажів та взаємодія з DM 28
2.4 Ієрархія елементів підсистеми вибору та меню **28**
2.5 Система забезпечення ігрового процесу 30
2.6 Опис реалізації ігрових механік **34**
2.7 Проєктування користувацький інтерфейс **38**
Висновки до розділу 2 **39**
РОЗДІЛ 3. ПРОГРАМНЕ І ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ **40**
3.1. Вимоги до технічного та програмного забезпечення **40**
3.2. Опис програмної реалізації 41
3.3 Перспективи та подальші можливості розвитку застосунку 51
Висновки до розділу 3 **52**
ВИСНОВКИ **53**

ВСТУП

Занурючись у захоплюючий світ комп'ютерних ігор, ми спостерігаємо, як ця індустрія, подібно до потужного промислового гіганта, невпинно розвивається і займає все більший сектор сучасної розважальної сфери. Впродовж останніх десятиліть ця галузь набирала обертів, охоплюючи все більше сфер нашого життя та зруйнувавши більшість стереотипів, продовжує залучати все більше шанувальників кожного дня. У цьому контексті, розробка комп'ютерних ігор стала вже звичною всім потужною галуззю, яка створює нові робочі місця, приносить колосальні прибутки, відкриває новий простір для реалізації новітніх ідей, що залучає нових розробників до цієї галузі та надає їм простір для самореалізації.

Протягом наступних десятиліть очікується стрімкий розвиток декількох ключових тенденцій, які вже мають значний вплив на розвиток індустрії комп'ютерних ігор. Зокрема, розвиток технологій віртуальної реальності (VR) та доповненої реальності (AR) стає очевидним. Ці інновації дозволяють гравцям занурюватися у віртуальні світи, взаємодіючи з ними на новому, реалістичному рівні. Це може привести галузь до абсолютно нових висот з неймовірною технологічністю та варіативністю, на рівні фантастичних творів, герої яких потрапляють у віртуальні світи, які стає все важче відрізнити від справжнього. Не менш важливою тенденцією залишається багатокористувацький режим, що об'єднує гравців у віртуальних просторах ігор, що дозволяє отримати новий ігровий досвід.

На фоні цих сприятливих тенденцій розвитку виділяється настільна гра "Dungeons & Dragons", яка є справді унікальною і не має аналогів. Ця гра приваблює не лише захоплюючими пригодами, але й надає абсолютну свободу творчості. Спільнота гравців D&D постійно розвивається, прагнучи вдосконалювати ігровий досвід, а різноманітні інструменти для створення сценаріїв відкривають нові можливості для творчості.

Ці платформи для створення та проведення історій у D&D виступають не лише як засіб технічної розробки, але і як полотно для вираження фантазій та створення унікальних світів. У цьому контексті важливо відзначити як технологічний аспект розробки, так і креативний потенціал гравців. Наприклад, гнучкі рушії, такі як Unity, дозволяють створити такі інструменти, що дозволять реалізувати будь-які творчі ідеї. Ентузіазм спільноти D&D у створенні унікальних наративів може досягнути нового рівня завдяки використанню таких інструментів, що покращить якість створюваних історій, полегшить підготовку гравців до гри, надасть незабутній ігровий досвід, що в свою чергу значно збільшить популярність гри та залучить нових гравців.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ І ЗАСОБІВ РЕАЛІЗАЦІЇ

1.1 Концепція настільної гри D&D(Dungeon and Dragons)

D&D (Dungeons & Dragons) це — настільна рольова гра зображена на Рис. 1.1 [1], що завоювала свою популярність своєю унікальною організацією і перебігом гри. Для того, щоб зіграти в гру гравцям необхідно створити свій чи запозичити один з безлічі готових історій і підготувати власних персонажів. Основною особливістю D&D є управління сценарієм історії і різноманітністю образів, які обмежуються тільки фантазією самих гравців та їхніми діями. Дії гравців можуть суттєво впливати на перебіг історії, адже саме їхні рішення є рушієм сюжету історії, що вони відіграють. Детальніше про історію успіху [1].

Рис. 1.1 Приклад зображення настільної мапи для D&D

Для проведення гри необхідно, щонайменше 2 гравці. Першим гравцем є Dungeon Master(DM) чи Майстер підземелля, який є ключовим гравцем, без якого неможливо

провести партію у D&D. Його задача створити або підібрати вже існуючу історію для інших гравців, також саме DM буде виконувати роль оповідача історії. Окрім створення історії, DM також керує всіма процесами гри, тобто він мусить відігрувати усіх не ігрових персонажів(NPC) і ворогів, встановлювати певні правила гри, накладати чи знімати з гравців різного роду обмеження і бонуси чи негативні і позитивні ефекти. Звичайні ж гравці, повинні створити чи вибрати базового персонажа для гри. Сам процес створення персонажів має цілий перелік правил, але одним з найважливіших правил, є створення персонажу саме під світ у якому проводиться історія. Також гравці можуть виконувати майже будь-які взаємодії між собою, не ігровими персонажами, ворогами чи об’єктами, відповідно до правил, які встановив DM.

Невід’ємною частиною настільної D&D є набір гральних кубиків зображений на Рис 1.2 [2]. Гравці використовують набори різно-полігональних гральних кубиків для виконання майже всіх ігрових дій, пов’язаних з ризиком, корисливими цілями, атаками, захистом і т.д. Наприклад, гравець хоче непоміченим наблизитись до ворога, для цього він кидає двадцатигранний кубик (D20), де відповідно високий показник на кубику означає успіх, а низький — невдачу. Таке правило, у будь-яких редакціях D&D, створює прямий баланс для гравців, щоб можна було розмірено проводити історію з різними по силі та навичками ворогами чи NPC, де гравці не зможуть робити все що їм заманеться і заставляє подумати гравців перед тим чи варта їхня дія того щоб ризикувати. Окрім гравців, DM також використовує ці кубики у важливих сюжетних розвилках, подіях і різних боях де він кидає кубики за ворогів.

Рис. 1.2 Приклад грального кубиків

Гра розвивається в залежності від вибраної історії та подій у які потрапляють чи оминають гравці. А гравці ж вирішують які дії будуть робити їхні персонажі, для досягнення їхньої цілі, що і є головним рушієм історії. У будь якого персонажа є свої персональні, розподілені характеристики, які визначають його навички, особливі уміння, схильності та інші атрибути. Окрім статистичних характеристик, персонажі також мають власні цілі, мотивацію, передісторію, належність до певної раси, професію, що створює елемент рольової гри, де на плечі кожного персонажа падає конкретна роль, як от воїн, що є основною бойовою одиницею, чи бард, що тримає бойовий дух команди і має розвинуті ораторські здібності для ведення важливих

розмов. Така система характеристик надає різним персонажам відповідні ролі, які важливі в тих чи інших ситуаціях, що додає грі великої варіативності та навіть стратегічного елементу.

У D&D також присутня система підняття рівнів для персонажів персонажів за отриманий досвід. Цей досвід можна отримувати за різні дії, як перемога над ворогами, виконання завдань, навіть вдало проведена розмова. Піднявши рівень, гравці можуть отримати одну чи декілька нових навичок, покращувати конкретні характеристики, часом навіть розвивати нові таланти. Це дозволяє розширювати їхні можливості в грі та переносити цих персонажів з однієї історії до іншої, розвиваючи історію одного персонажа.

Всі характеристики, таланти, передісторія, раса, інвентар та інша інформація про персонажа записується у спеціальному “Листі персонажа” приклад якого продемонстрований на Рис. 1.3 [3]. Також при будь якій дії, яка включає в себе взаємодію з предметами, бій, підняття рівня, торгівля, потрібно обов’язково вести відповідні записи та змінювати показники у листі персонажа, для того щоб не втратити жодної інформації.

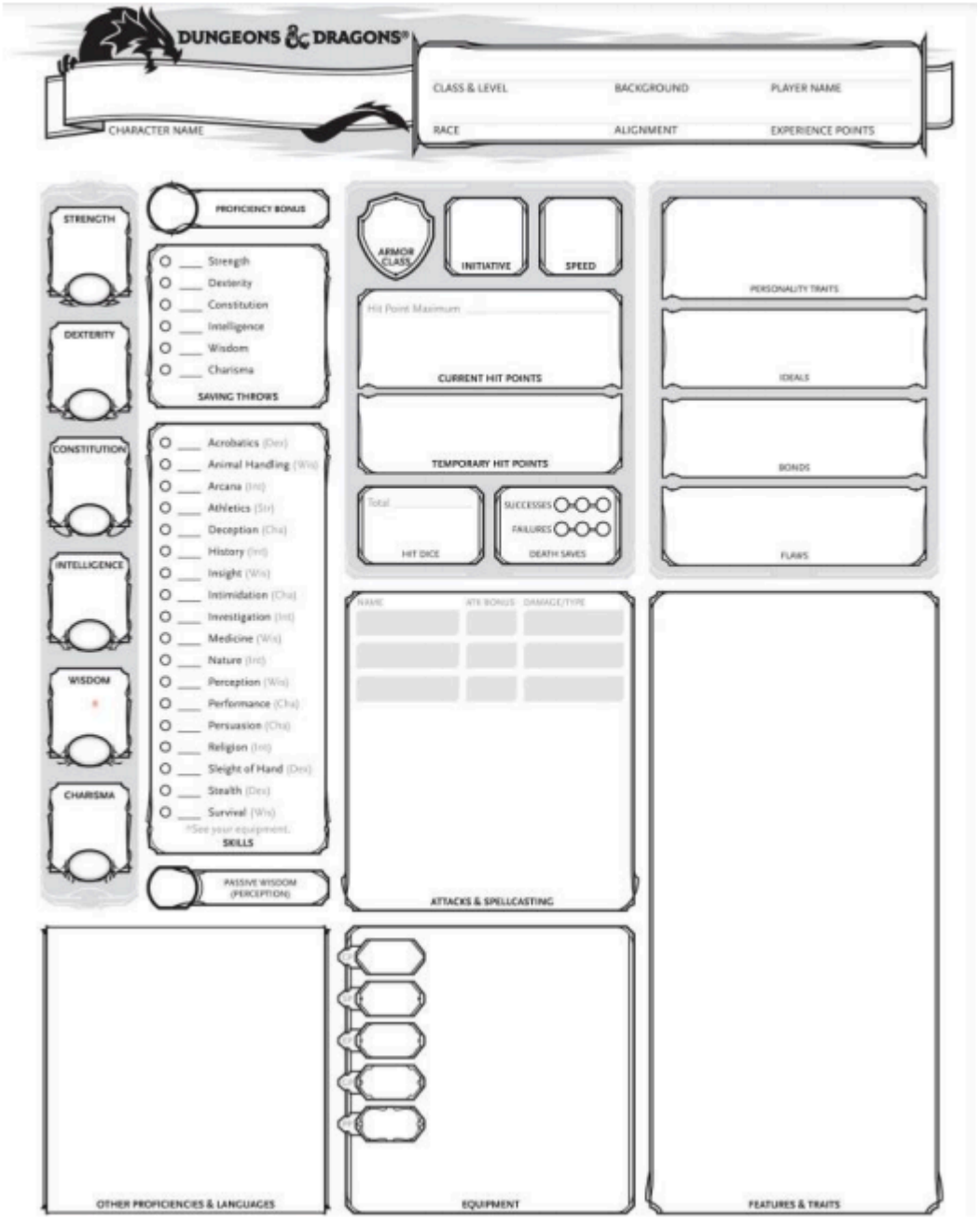

Рис. 1.3 Приклад вигляду листа персонажа

Загалом, D&D(Dungeons & Dragons) — це складна та захоплююча настільна гра з великою комбінацією різних ігрових елементів та механік, де гравці мають можливість зануритись у один з безлічі цих світів та пережити історію від обличчя свого героя, де їхня творчість і винахідливість обмежені лише власною уявою гравців.

1.2 Аналіз прототипів та порівняння

При дослідженні пропозицій, які можна віднайти у магазині Steam чи інтернеті, що дають можливість створювати історій чи асети для D&D, можна побачити, що ця ніша має потужну перспектива для подальшого розвитку. З урахуванням не великої кількості конкурентів та їхнім вкладом у цю нішу, вони змогли здобути певну свою популярність і популяризувати саму настільну гру D&D. Серед конкурентів можна виділити наступних:

Tabletop Simulator: симулятор настільних ігор, дозволяє грати у будь-які класичні і не тільки настільні ігри, приклад настільної гри Го у Tabletop Simulator на Рис. 1.4 [4]. Наприклад інтелектуальні ігри як шахи, шашки, нарди, чи різні карткові ігри, як покер, різноманітні пас’янси, або ж навіть просто збирати пазли чи готувати історії для D&D, з можливістю їх проводити для різних гравців.

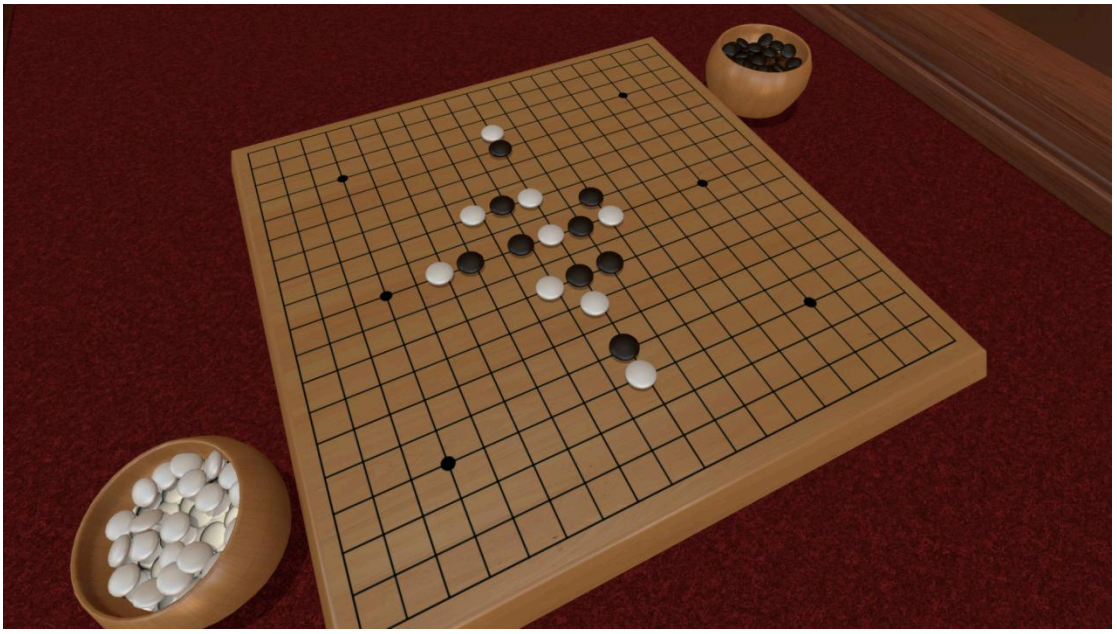

Рис. 1.4 Настільна гра у Tabletop Simulator

Незвичайною особливістю цієї гри є повна свобода дій, тобто усі дії гравців ніяк не обмежуються будь-якими правилами конкретних настільних ігор. Наприклад, при грі в шахи, гравці можуть безперешкодно неправильно походити фігурою чи походити фігурою свого суперника або ж навіть викинути його фігури чи перекинути стіл, що продемонстровано на Рис. 1.5 [4].

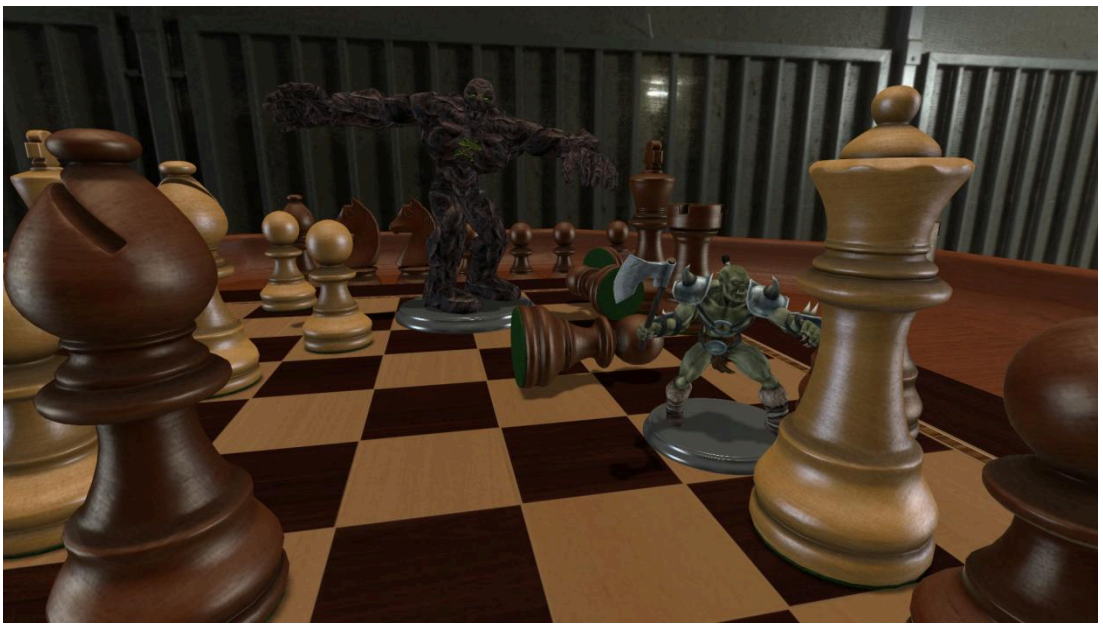

Рис. 1.5 Фігурки для D&D на шахівниці

Canvas of Kings: це справді особлива гра, оскільки вона є потужним інструментом для побудови різних рівнів для D&D історій. Гра, як інструмент, є досить мінімалістичною та простою у використанні. У цій грі можна без проблем малювати рівні за допомогою динамічної взаємодії ассетів між собою, легкої взаємодії з зображенням, нанесенням ліній для вказування напрямку та впорядкування об’єктів і контролювати освітлення на зображенні, що продемонстровано на Рис. 1.6 [5].

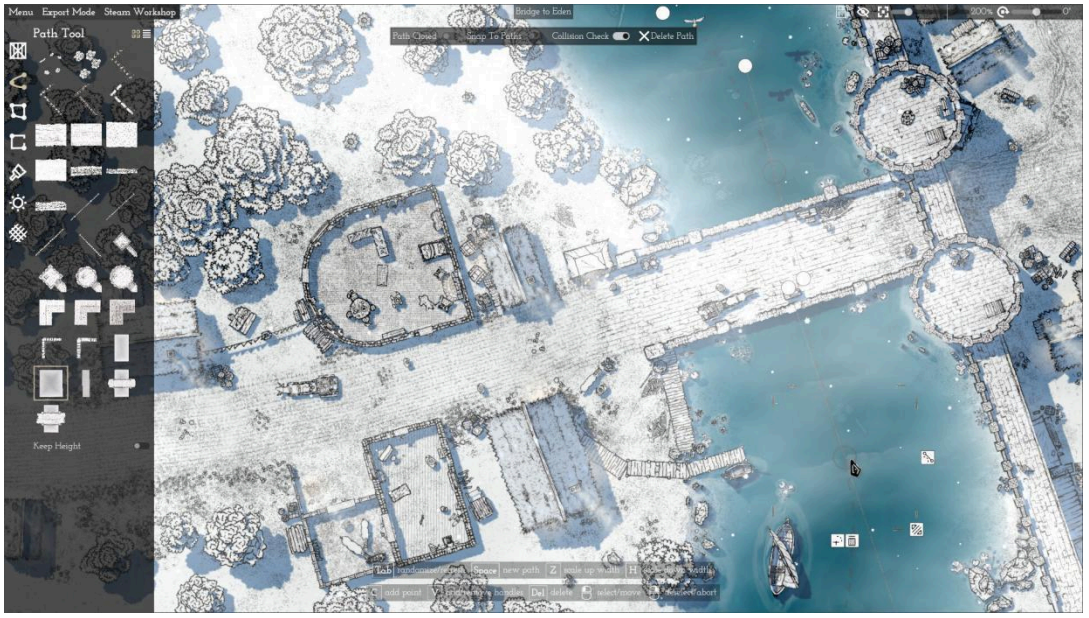

Рис. 1.6 Процес побудови рівня

Roll20: це популярна веб-платформа для проведення ігор D&D або в стилі D&D в браузері. Основною тематикою є ігри за мотивами D&D у класичному його форматі, зі своїми правилами відповідно до редакцій правил D&D чи ігри побудовані за правилами гри Dungeon & Dragons. Головна сторінка веб-платформи зображена на Рис. 1.7 [6].

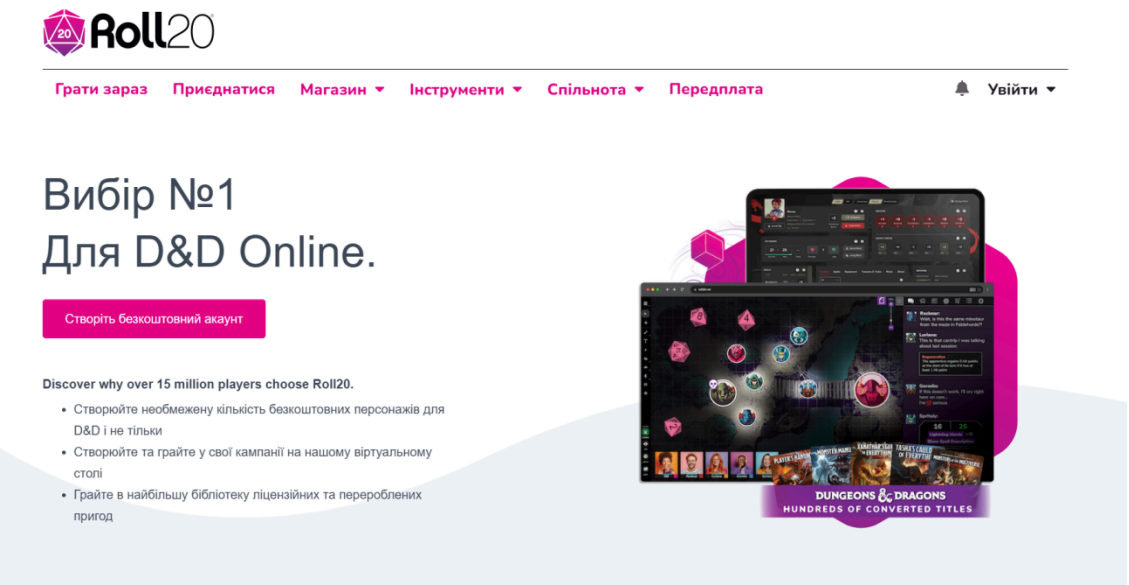

Рис. 1.7 Платформа для проведення ігор у стилі D&D

На цій веб-платформі можна легко створювати D&D історії та проводити їх для не обмеженої кількості гравців. Також платформа містить ,як декілька редакцій самої Dungeon & Dragons, так і деякі інші ігри, створені за мотивами D&D, де ключовою складовою є механіки пов’язані з гральними кубиками, а також присутнє відігравання ролей своїх персонажів.

1.3 Вибір програмних інструментів реалізації

1.3.1 Ігровий рушій Unity

Існує чимало ігрових рушіїв. Наприклад Unreal engine, який є найбільш технологічним та який використовується у великих проектах для відтворення кінематографічних сцен. Чи Valve Sourse Engine який має одну з найкращих

бібліотек для реалізації фізичних явищ. Або Godot Engine, який є повністю модульним рушієм, що дозволяє використовувати тільки необхідні бібліотеки для розробки. Та вибір попав на Unity [8].

Unity - це ігровий рушій для розробки ігор, який використовує мову програмування C#, для різних ігрових платформ, таких як PC, Android, Playstation, Xbox чи MacOS. Ігровий рушій Unity є безкоштовним, поки проєкт, створений за допомогою цього рушія, не приносить 100тис. доларів на рік, що дозволяє розробникам безперешкодно спробувати створювати ігри і вподальшому плані працювати та розвиватися в цій індустрії. Рушій має всі можливості та необхідні компоненти для створення майже всіх видів 3D та 2D ігор, відразу з відповідною візуалізацією, музичним супроводом, звуковими ефектами, освітленням, анімацією, фізикою об'єктів і т.д. Також у Unity є свій магазин “Unity Asset Store” з величезною кількістю різноманітних ассетів різної вартості, завдяки чому, можна знайти необхідні ассети для різних цілей, як для новачків, які хочуть спробувати розробку ігор, так і для більш досвідчених розробників. Також можна підібрати відповідні ассети для вибраного жанру гри, як от 2D чи 3D пісочниця, шутер, хоррор, RPG, головоломки, настільні ігри, стратегії і т.д. Ассети, для втілення своїх ідей, можна обирати відповідно до свого бюджету та рівня обізнаності у сфері розробки ігор. З більшою кількістю ігрових рушіїв можна ознайомитись тут [7].

Переваги використання Unity:

1. **Велика база компонентів**: Unity містить величезну кількість різних компонентів та відповідного функціоналу для створення ігор, незалежно від жанрів чи об’єму створюваної гри.

2. **Простота в освоєнні**: хоч найкращим вибором для створення “AAA ігор” є ігровий рушій Unreal Engine, він потребує значного досвіду створення ігор та потужних знань мови програмування C++, що робить його складним для використання навіть для досить досвідчених розробників. В свою чергу Unity має такий функціонал, що одночасно підійде новачку для освоєння базового

функціоналу рушія, так і для досвідчених розробників, що можуть реалізовувати функціонал Unity на рівні Unreal Engine.

3. **Доступність**: оскільки ігровий рушій Unity є безплатним для використання, це робить його ідеальним вибором для початківців та поодиноких чи невеликих груп розробників, порівняно з тим самим Unreal Engine де використовується щомісячна підписка.

4. **Постійна підтримка рушія**: рушій Unity постійно отримує регулярні оновлення та покращення, які полегшують його використання чи виправляють помилки, що дозволяє йому постійно розвиватись та конкурувати з іншими ігровими рушіями.

Для роботи над даним проєктом було вибрано саме ігровий рушій Unity. Хоч вибір між трьома рушіями Unity, Unreal Engine, Godot, в силу вище перелічених переваг, саме ігровий рушій Unity став найоптимальнішим варіантом. Адже в порівнянні з іншими рушіями Unity має все необхідне для реалізації поставленої задачі та є найлегшим в освоєнні.

1.3.2 Середовище розробки Visual Studio

Інтегрованих середовищ розробок існує просто безліч. Найпопулярніші з них це: PhpStorm, PyCharm, CLion, Visual Studio, Eclipse. Всі ці IDE(integrated development environment) підтримують чималу кількість мов програмування та мають різні переваги один перед одним, що дозволяє вибрати конкретне середовище розробки для власних потреб без перегруженного функціоналу, чи з необхідними для розробки можливостями. Серед цього списку було обрано Visual Studio.

Visual Studio - це інтегроване середовище розробки, розроблене компанією Microsoft для створення різних настільних та мобільних застосунків, веб-сервісів і сайтів чи програмного забезпечення для ОС Windows. Visual Studio окрім підтримки мови програмування C#, також підтримує підтримує такі мови програмування як C++, Python, JavaScript. Також Visual Studio має досить зручний та зрозумілий

функціонал, включаючи редактор коду із зручним підсвічуванням синтаксису та рефакторингом коду, інструменти для відлагодження коду, вбудована підтримка система контролю версій, таких як Git. Детальніше про Visual Studio [9].

Переваги Visual Studio:

1. **Широкий набір інструментів**: Visual Studio надає широкий набір інструментів для розробки, тестування та відлагодження, що покращує процес розробки.

2. **Розширюваність**: Для Visual Studio існує широкий набір плагінів та розширень, за допомогою яких можна налаштовувати середовище під свої потреби та додавати необхідні функції.

3. **Кросплатформеність**: Visual Studio має версії як для Windows так і для macOS, а також підтримує розробку кросплатформенних застосунків за допомогою .NET Core, Xamarin та інших технологій.

1.3.3 Мова програмування C#

Вибір мови програмування C# став автоматичним, оскільки ігровий рушій Unity може працювати тільки з нею. Та якщо відкинути Unity та вибрати серед інших ігрових рушіїв та середовищ розробки, можна використовувати такі мови програмування як Java, C++ чи Pyton.

C# - це об’єктно орієнтована мова програмування, що походить з сімейства мов C. Вона використовується для розробки майже всіх видів програмних продуктів, починаючи зі створення ігор чи настільних застосунків закінчуючи сайтами і веб-сервісами. Мова програмування C# здобула свою популярність через те, що є універсально та досить легкою для вивчення. Microsoft створили C# як просту та сучасну мову програмування для універсального призначення. Вона успадкувала багато корисних можливостей та зручностей від своїх попередників, позбувшись

проблематичних моделей розробки. На протязі всього свого існування, користувачі знайшли їй безліч застосувань в різних сферах програмування, зарекомендувавши себе як універсальна мова програмування для будь-яких потреб. Більше про C# тут [10].

Переваги мови програмування C#:

1. **Об'єктно-орієнтоване програмування**: C# є об'єктно-орієнтованою мовою програмування, що надає всі переваги ООП при розробці будь-якого програмного продукту.

2. **Платформна незалежність**: платформа Microsoft.NET використовується у різних операційних системах Windows, Linux і MacOS, що дозволяє застосункам створеними за допомогою мови програмування C# правильно функціонувати незалежно від ОС.

3. **Висока сумісність з технологіями**: завдяки універсальності C#, при розробці програмних продуктів, є можливість інтегрувати різноманітну кількість спеціалізованих технологій для будь-яких потреб.

1.4 Порівняльна характеристика інструментів реалізації

З урахуванням вище перелічених переваг засобів розробки, було вибрано наступні: Unity, Unity Asset Store, Visual Studio, Jira, - для роботи з скриптами, інтерфейсом та бібліотеками.

Рушій Unity - є одним із найпопулярніших і найкращих програм для створення ігор, різноманітних додатків та створення дизайну для середовища.

Також Unity є потужним графічним рушієм оскільки він підтримує різні ефекти, включаючи освітлення, тіні, текстури та шейдери. Перевагою для розробників також є вбудована система фізики та система анімацій. Система анімацій має вбудовані інструменти для створення складних анімацій для персонажів, об'єктів та інших елементів гри. Завдяки вбудованій системі фізики розробник має можливість

створювати реалістичні фізичні об’єкти, такі як гравітація, колізії та симуляція руху об’єктів. Таким чином наявність такого вбудованого інструментарію значно спрощує розробку і дозволяє її прискорити.

Окрім широкого інструментарію, Unity має хорошу сумісність з середовищем розробки Visual Studio, що в свою чергу дозволяє в повній мірі користуватися зручностями розробки, такими як: підказки, виявлення конкретних помилок, використання Git, автоматичне підключення необхідних бібліотек, використання різних мов програмування.

Ще важливим аспектом є те що Unity використовує мову програмування C#. Завдяки їй можна розробляти високотехнологічні ігри з використанням нових технологій. Також C# дозволяє легко розробляти кросплатформені застосунки для різних операційних систем, як Windows чи Linux, або навіть для мобільних Android.

Висновки до розділу 1

У цьому розділі, було описано внутрішньо ігрові можливості настільної гри Dungeon and Dragons, з урахуванням усіх складових гри, таких як: відігравання ролі персонажа, особливості грального кубиків, спеціальні мапи, вони ж локації, процес побудови гри, листи персонажів.

Був проведений аналіз конкурентів та їх можливостей. З більшою увагою було розглянуто особливі риси і концепції конкурентів, що лежати в основі їхніх продуктів та надавали їм унікальних можливостей.

Також було визначено основний перелік інструментів для розробки ігрового застосунку і було вказано конкретні переваги, що допоможуть у розробці даного застосунку.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СТРУКТУРИ ТА ФУНКЦІОНАЛЬНИХ СКЛАДОВИХ ІГРОВОГО ЗАСТОСУНКУ

2.1. Постановка задачі

Популярність Dungeon and Dragons щодня зростає, відповідно кожен розробник у цій ніші намагається надати все більше корисного функціоналу для своєї гри, щоб зацікавити як можна більше гравців та бути більш конкурентоспроможним. Також слід не забувати про унікальність, адже скільки б хороших функцій та контенту не було б у грі, якщо у грі є ще унікальні механіки, то це беззаперечно зробить гру більш конкурентоспроможною і цікавішою для пересічних гравців.

Для того щоб створити таку гру потрібно правильно розпланувати кожен етап розробки, починаючи від концепції гри і базових механік і закінчуючи рекламною кампанією, відкритим тестуванням і демонстрацією процесу гри цільовій аудиторії. Оскільки D&D є рольовою грою з необмеженою кількістю сюжетів, залучення гравців не буде складною задачею, проте слід не забувати про зручність гри, адже без хорошого і зрозумілого функціоналу гра швидко буде втрачати цільову аудиторію, формуючи про себе погане враження. Щоб уникнути такого результату потрібно зробити гру за наступними пунктами:

1. Придумати принаймні одну унікальну рису для гри, у цьому пункті було вирішено виконувати гру тривимірному середовищі.

2. Продумати увесь мінімально необхідний функціонал для гри, наприклад встановлення об’єктів, підкидування кубика, збереження ігрових рівнів.

3. Підібрати від одного до трьох наборів простих асетів, для створення бібліотеки необхідної для проектування рівнів.

2.2 Архітектура та механіки ігрового застосунку

Цікаві ідеї та унікальні елементи гри беззаперечно важливі, але все ж без належних механік гра буде не повноцінною. Тому планування основних механік є одним з найважливіших етапів розробки. Для того щоб зрозуміти, які саме механіки потрібні для реалізації гри, необхідно оцінити увесь базовий функціонал настільної гри Dungeon and Dragons. Оцінивши навіть одну редакцію D&D, стає зрозуміло що в реалізації всіх правил взаємодій, описаних в цій редакції, немає потреби, адже регулюванням гри займається Майстер Підземелля. Тому слід зосередитися на технічних механіках, необхідних для простого функціоналу, які наведені нижче:

1) Побудова мапи. В грі необхідно мати спеціалізований конструктор для створення своїх власних ігрових рівнів. Це одна з ключових механік, необхідних для побудови свого сюжету. Вона буде включати в себе бібліотеку елементів, для побудови рівня та систему збереження рівнів.

2) Компілювання рівнів. Другою необхідною механікою буде формування сюжетів вже зі створених рівнів.

3) Переміщення між рівнями. Механіка потрібна для самого процесу гри в D&D, щоб змінювати локації.

4) Гральні кубики. Гральні кубики є головним елементом самого процесу гри у D&D, який вирішує перебіг всіх взаємодій у грі.

Архітектура гри, зображена нижче(Рис. 2.1), це ігрова діаграма, яка показує взаємодію одних механік з іншими та показує залежність між ними. В самій грі архітектура реалізована відповідними папками зі скриптами для реалізації тих чи інших механік. Архітектура включає в себе наступні пункти: Level Creation, Level Saver, Player Move, Story Builder, Dices, Game Menu, Story Processing.

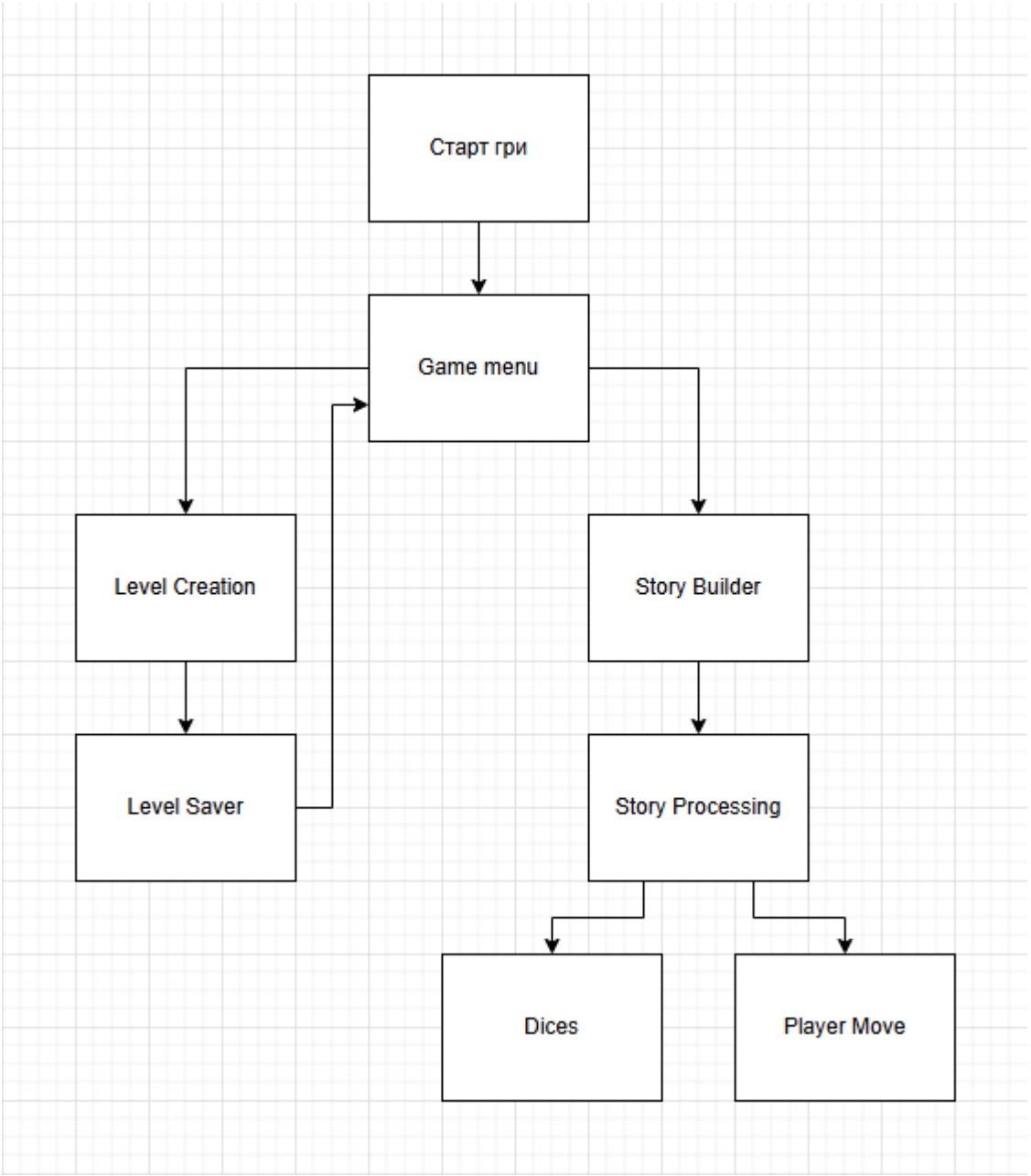

Рис. 2.1 Ігрова діаграма з архітектурою гри.

Джерело: розроблено автором

Архітектура гри відображає порядок реалізації механік і відповідно залежність конкретної механіки від іншої, що дає розуміння де одна механіка переходить до іншої. Це означає що одні механіки не можуть існувати без інших, тому дуже важливо, правильно описати архітектуру, щоб не перетворити структуру гри у “клубок ниток”.

Організація скриптів відповідає архітектурі гри, що реалізована папками, де в кожній папці знаходяться скрипти, які пов’язані саме з конкретною механікою. Level Creation відповідає за побудову рівнів, Level Saver - за систему збереження рівнів, Player Move - за переміщення фігурок, Story Builder - за збір історії, UI - за увесь функціональний інтерфейс, Dices - за роботу гральних кубиків, Game Menu - за систему ігрового меню, Story Processing - за процес проведення ігрової сесії.

2.3 Рольові моделі гравців і їх персонажів та взаємодія з DM

Для гри в партію D&D всі гравці повинні підготувати власних персонажів. Всю цю інформацію гравці записують у свої листи персонажів. Це є одним із найважливіших аспектів гри, оскільки в листі персонажа описується інформація для відігравання ролі.

Базуючись вже на заданих гравцями рис характеру, віри, професії, переконань власного персонажа, гравці можуть мати різні можливості і взаємодії. Це поширюється як на неігрових персонажів так і на інших гравців. Наприклад два гравці які мають професії лицар високого походження і злодій, повинні погано ладнати, оскільки у них дуже сильно відрізняються світогляди, чи навпаки гравці з класами священика і паладина, маючи схожі світогляди і віру, будуть добре ладнати і часто можуть мати подібні ідеї, пропозиції чи вирішення проблем. Детальніше дізнатися про ігрові класи можна тут [11].

Також багато дій гравців можуть обмежуватись DM-ом, оскільки вони можуть порушувати загальний хід подій чи суперечити логічним рішенням, які спеціально були підготовлені для конкретної ситуації. А ще всі взаємодії без винятків проходять через DM-а і повністю ним контролюються.

2.4 Ієрархія елементів підсистеми вибору та меню

Перед створенням головного меню необхідно розподілити ігровий функціонал гри на частини. Для цього потрібно виокремити конкретні ігрові механіки, що пов’язані між собою, та згрупувати. Це продемонстровано на наступній діаграмі(Рис. 2.2)

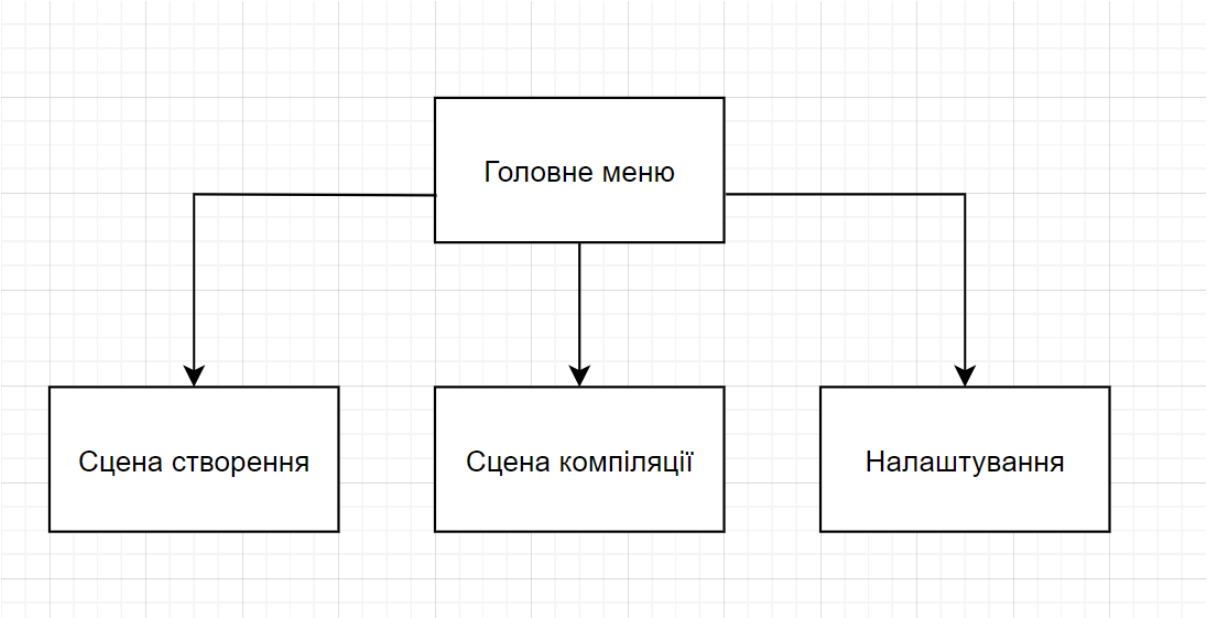

Рис. 2.2 Структура гри

Джерело: розроблено автором

З такою структурою головного меню, було розподілено два ігрових режими, де в першому гравці можуть створювати свої рівні і зберігати їх, а в другому відповідно використовувати їх для побудови своїх сюжетів. Прототип меню, що відповідає даній задачі, зображений нижче(Рис. 2.3)

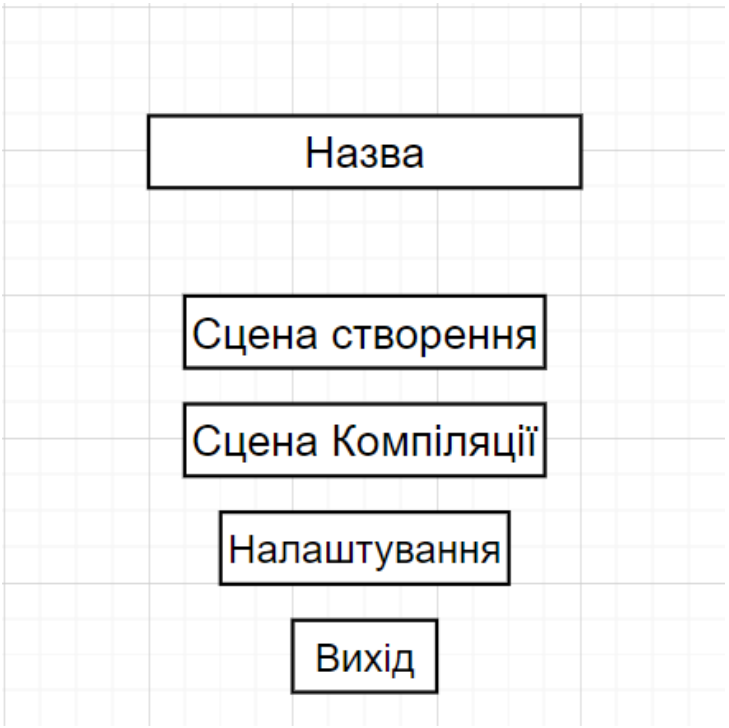

Рис. 2.3 Макет головного меню

Джерело: розроблено автором

2.5 Система забезпечення ігрового процесу

Система забезпечення ігрового процесу - це комплексна структура ігрових модулів, що відповідають за реалізацію своїх ігрових режимів. Кожен ігровий модуль містить необхідні одну або декілька підсистем для реалізації ігрових механік. Загалом можна розділити ігровий процес на 3 частини, де перша це реалізація бібліотеки елементів для побудови рівнів, друга це збереження побудованих рівнів та їх редагування, а третя створення ігрових сюжетів вже зі створених рівнів та їх проведення.

Відповідно до архітектури гри всі ігрові механіки були розділені та згруповані так, щоб жодна з механік одного ігрового процесу ніяк не взаємодіяла з механіками інших ігрових процесів.

2.5.1 Підсистема бібліотеки елементів

Для створення бібліотеки елементів було використано набір підготовлених префабів. Префаб - це спеціально сформований шаблон об'єкта, який можна повторно використовувати вже із заданими параметрами. Його можна використовувати для швидкого та зручного будівництва сцен та використовувати при генерації мап.

Префаби зручні тим, що їх можна використовувати необмежену кількість разів, замість дублювання одного і того ж об’єкта, також їх можна оновлювати в реальному часі, тобто змінивши сам шаблон вже всі виставлені префаби також зміняться за шаблоном. Однією з корисних властивостей префабів є незалежність від рівня, тобто якщо змінювати саму сцену, чи об’єкти навколо префабів то це ніяк на нього не вплине. Приклад продемонстрований на Рис. 2.4 детально можна розглянути тут [12].

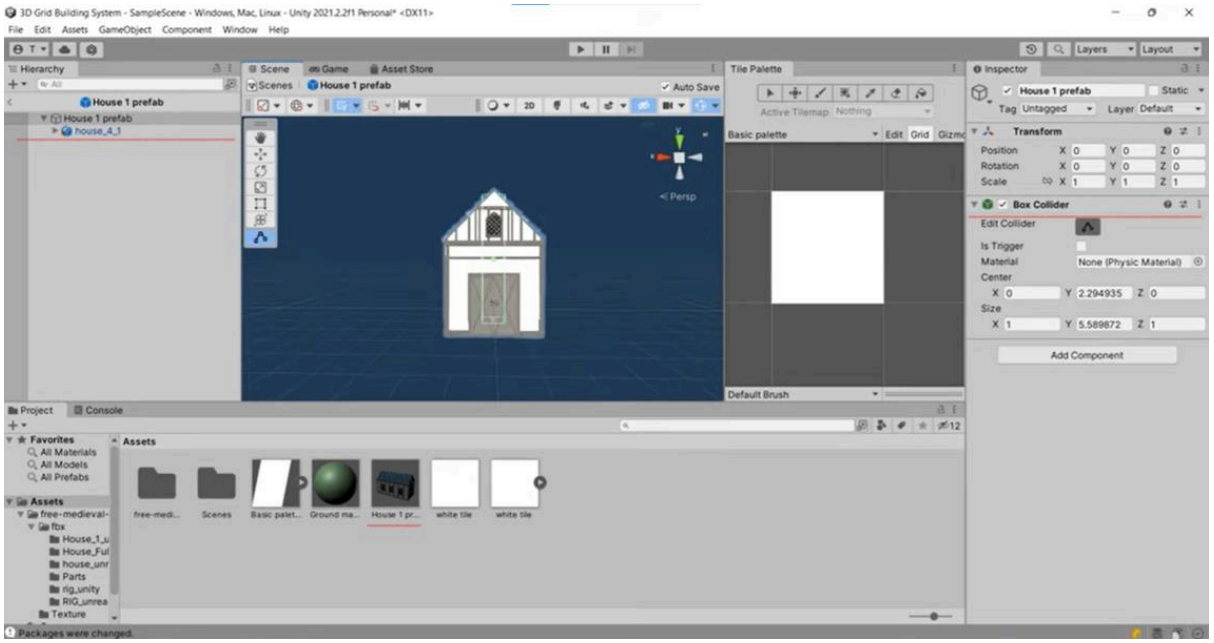

Рис. 2.4 Приклад роботи з префабом

Як приклад, представимо що у вас є ворог, який має певну поведінкою, властивості, показники і ми хочемо щоб цей ворог був у декількох місцях гри, для цього потрібно створити префаб ворога і його можна без проблем розміщувати

різних рівнях гри, а властивості вже конкретних ворогів можна коригувати в залежності від обставин. Такий принцип можна застосовувати до будь-яких ігрових об’єктів, чи то звичайні елементи ландшафту, чи складні структури, чи, як було вище згадано, вороги.

Для самої реалізації цієї системи необхідно підготувати певну кількість таких елементів та зібрати їх у список. Наступним кроком буде створити систему виклику елементів для їхньої появи на сцені і відповідно систему їхнього видалення, якщо ці елементи були зайвими.

2.5.2 Підсистема збереження рівнів

Є декілька способів реалізації механіки збереження інформації про рівні. Наприклад JsonSerialization, PlayerPrefs чи бінарна серіалізація. В даному проєкті використаний спосіб з Json серіалізацією. Json серіалізація працює наступним чином: збираються дані об’єкта, які містять всю необхідну інформацію про місцезнаходження об’єкта і автоматично нумерується. Ці дані конвертуються в Json рядки та записуються у файл. Тепер, при завантажені гри, всі об’єкти, які були записані в цей Json файл, будуть появлятися на сцені, при її завантаженні. Схематичний приклад зображено нижче(Рис. 2.5)

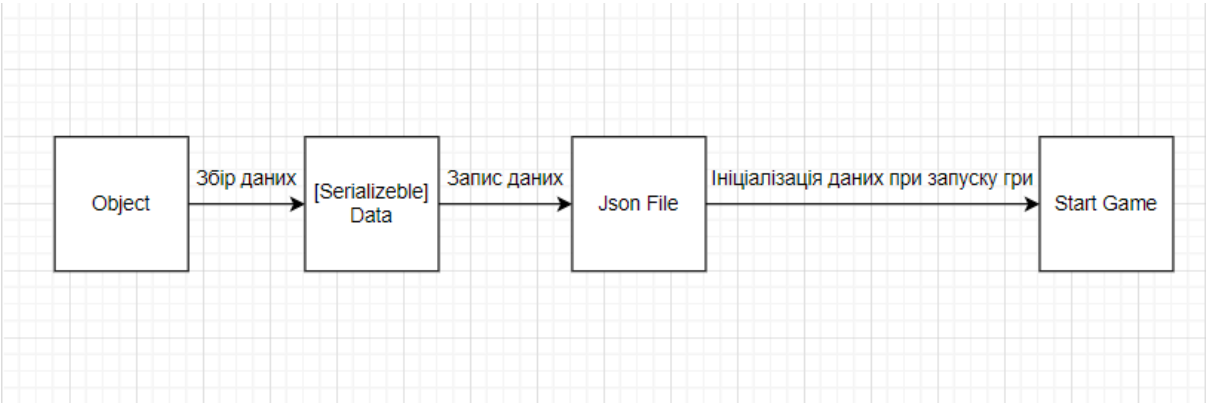

Рис. 2.5 Зображення процесу серіалізації та ініціалізації даних

Джерело: розроблено автором

Реалізована ця механіка за допомогою 3 скриптів: “Serializator”, “LevelSaver”, “SaveSystem”. Скрипт “Serializator” є основою системи збереження, його задача

полягає у зчитуванні даних об’єктів і конвертації їх у звичайні дані, які можна записати у Json рядки. Наступний скрипт “PlacebleObjectData”, відповідає за вид об’єктів, які можна встановити на сцену, тобто всі об’єкти.

Наступний скрипт “SaveSystem” відповідає за роботу з файлами. Його задача полягає у створенні необхідних директорій, записі даних до цих директорій та зчитування уже записаних даних раніше.

2.5.3 Підсистема побудови сюжетів

Система для побудови сюжетів потрібна для того, щоб відбирати необхідні створені рівні гравців та створювати з них ланцюжки рівнів, які уособлюють собою сюжетні лінії. Сам механізм будується на системах вибору рівнів і системі переходу між рівнями. Схема роботи концепції зображена нижче(Рис. 2.6)

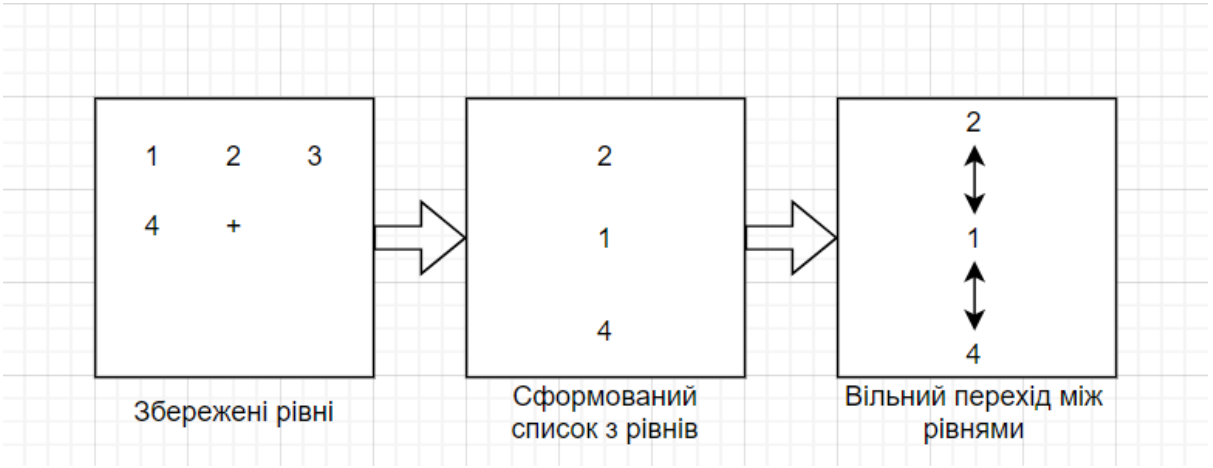

Рис. 2.6 Схема концепції побудови сюжетів

Джерело: розроблено автором

Сама система побудови сюжетів побудована наступним чином: у спеціалізованій директорії, можна вибирати та використовувати для побудови своїх сюжетів. За це відповідає скрипт “LevelLoad”, який може завантажувати наступні чи попередні рівні. А список і порядок рівнів буде визначатися гравцем, який вибирає їх зі своєї директорії, за допомогою скрипту “LevelSelector”. Саме за такої структури в майбутньому можна буде гнучкіше працювати з рівнями, додаючи нові концепції під рівнів чи створюючи лабіринти з самих рівнів.

2.6 Опис реалізації ігрових механік

Тепер, сформувавши структуру кожного ігрового процесу та перелік необхідних механік, потрібно розробити самі механіки.

2.6.1 Механіки побудови рівнів

Механіки побудови рівнів включають у себе можливості для визначення положення на сітці, встановлення об’єктів з вже описаної вище бібліотеки елементів, візуалізацію сітки для встановлення об’єктів, обрахування розмірів об’єктів для уникнення колізій, візуалізацію об’єктів перед встановленням, видалення об’єктів. Загалом це є комплекстна механіка, результатом роботи якої є коректне встановлення і видалення об’єктів на сцені.

Ключовою механікою тут є правильне визначення клітинок. Основною проблематикою таких систем є робота саме з світовою системою координат, яка використовує числа з дробовою частиною. А наша задача встановлювати об’єкти в конкретні клітинки, які визначаються системою координат з цілих чисел.

Найпоширенішим способом вирішенням цієї проблеми є методи WorldToCell() і CellToWorld(), основна задача яких конвертувати координати з однієї системи в іншу. Детальний опис та застосування описані тут [13].

Метод WordToCell() конвертує значення світових координати у цілі числа, відкидаючи дробову частину. Тобто ми будемо отримувати координати конкретної клітинки. А метод CellToWorld() відповідно повертає світові координати.

Нижче зображено приклад розпізнавання клітинок (Рис. 2.7). На ньому видно, що точка А знаходиться на графіку за координатами, що менші за (1;1). В цьому випадку точка А знаходиться на клітинці (0;0). Відповідно точка В що знаходиться в координатах менші за (3;1), означає що координати клітинки в якій вона знаходиться це (2;0). І оскільки в одній клітинці може знаходитись безліч точок, відповідно група

точок C, D знаходиться в одній клітинці з координатами (2;2), як і група C1, D1 за координатами (-3; 0).

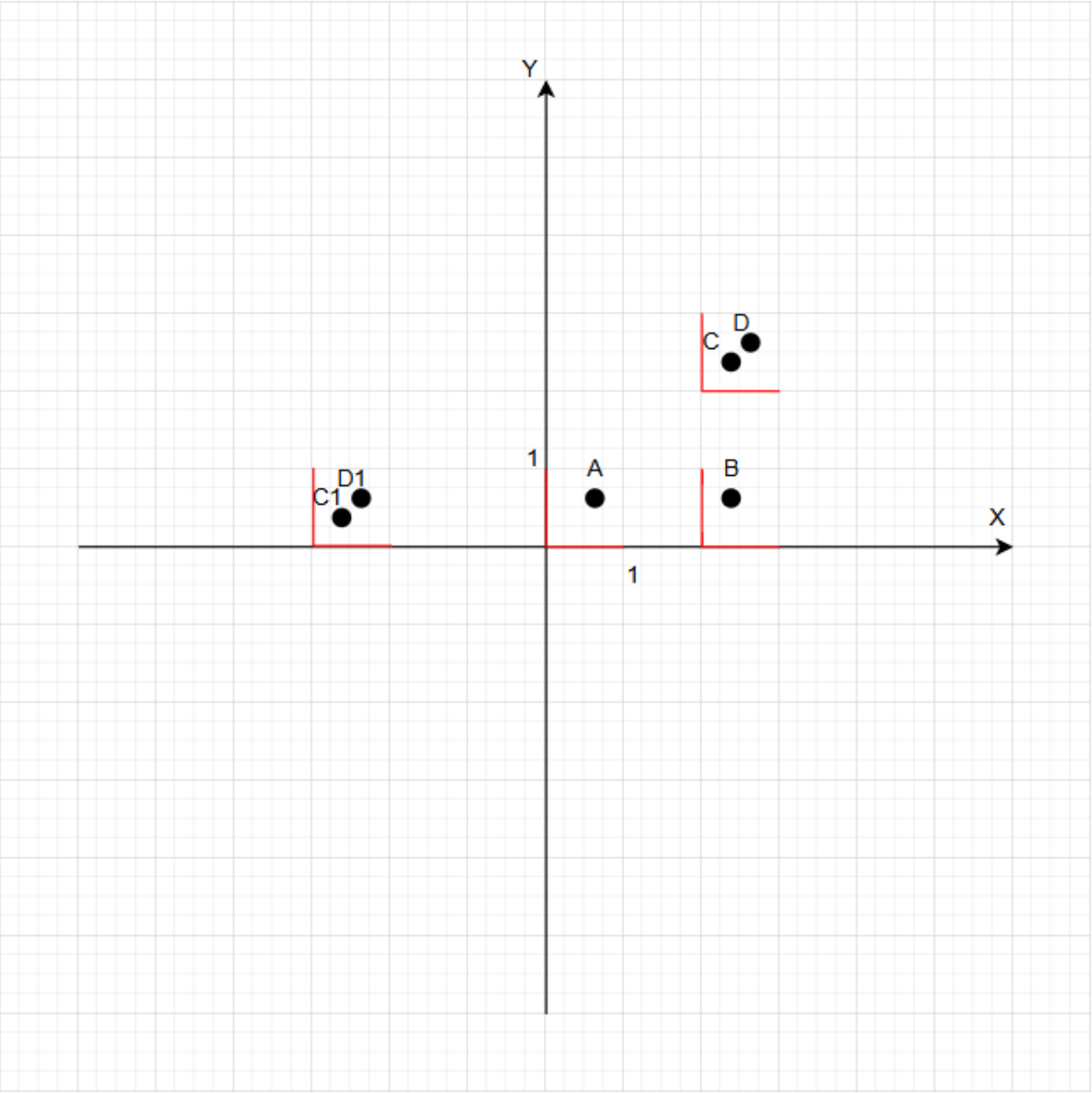

Рис. 2.7 Зображення прикладу роботи розпізнавання клітинок

Джерело: розроблено автором

2.6.2 Механіки роботи з рівнями

Механіки роботи з рівнями взаємодіють саме з сценами, які будуть зберігатися, редаговуватись чи перезаписуватися. Також в цей перелік буде входити робота з сюжетами, адже ця механіка працює тільки з готовими рівнями.

Загальний принцип полягає у записі параметрів у JSON форматі та збереженні їх у файлі, а при використанні вже збережених рівнів записані дані десеріалізуються та викликають відповідні об’єкти у відповідних місця.

2.6.3 Механіки для проведення гри

Перелік механік для проведення гри включає в себе роботу з фігурками гравців, гральни кубиками, переходами між рівнями. Загалом всі необхідні механіки для комфортного проведення гри. Наприклад механіки підкидування кубиків, переміщення фігурок чи візуалізація переходу між рівнями.

Загалом способів реалізації механіки підкидування кубиків існує чимало, проте з урахуванням, що гральні кубики стають все круглішими з більшою кількістю сторін, можуть виникати труднощі з поведінкою кубиків і розумінням того, яке значення випало на тому кубику.

З урахуванням цих недоліків для реалізації цієї механіки хорошим способом буде використати випадкові числа. Наприклад нам потрібно підкинути двадцятигранний кубик, для цього можна зробити простий алгоритм де випаде одне число від 1 до 20. Це найпростіший спосіб реалізації, тому щоб це виглядало живіше, можна додати анімацію, у окремому вікні, де буде підкидуватись кубик, а потім на фоні кубика буде показуватись відповідне число. Відповідно було розроблено простий алгоритм генерації випадкового (Рис. 2.8), що зображений нижче. Більше про генерацію випадкових чисел тут [14].

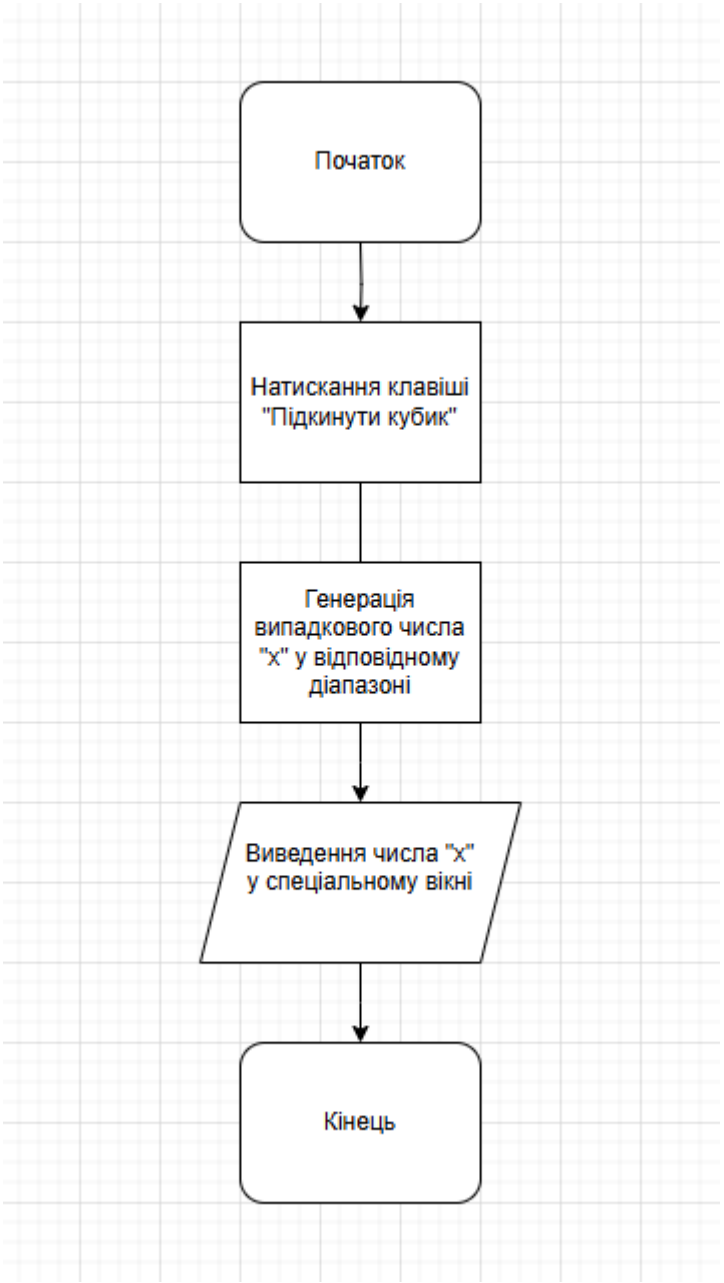

Рис. 2.8 Алгоритм генерації випадкового числа для грального кубика

Джерело: розроблено автором

Робота з фігуркаими гравців також має чимало незручних моментів, тому в даному в даному випадку найкращим варіантом буде звести переміщення фігурки до простого перетягування за курсором.

Перехід між рівнями викликає найменше проблем, оскільки зміною рівнів займається група скриптів для побудови сюжетів, а сам скрипт для візульного переходу буде викликати відповідні ефекти для пом’якшенняя сприйняття.

2.7 Проєктування користувацький інтерфейс

Ще одним з ключових елементів гри, є користувацький інтерфейс. Зрозуміло, що з відсутністю інтерфейсу буде взагалі неможливо управляти процесами гри, чи то буде побудова та збереження рівнів, чи це сам процес гри в D&D. Відповідно для кожної задачі потрібен, свій, спеціалізований інтерфейс, який буде задовільняти потреби конкретного елементу гри.

Для розробки усіх необхідних інтерфейсів потрібно звернутись до архітектури гри, де вже правильно згруповані механіки, до яких можна розробити спеціалізовані інтерфейси.

Загалом потрібно розробити три спеціалізованих інтерфейси для кожної ключової механіки.

1) Перший інтерфейс буде відповідати за побудову рівнів. Він повинен включати в себе бібліотеку елементів, кнопки збереження та кнопки завантаження.

2) Другий буде відповідати за збір історії. У цьому інтерфейсі потрібно мати можливість бачити всі раніше створені рівні, також мати візуалізацію процесу збору сюжету з цих рівнів.

3) Третій необхідний для проведення самої історії. Цей інтерфейс повинен містити всі необхідні компоненти для самої гри. В цей перелік входять кнопки для переключення рівнів і вікно для підкидування кубика.

Також, всі інтерфейси повинні містити кнопки швидкого виходу з ігрового режиму, тобто надавати можливість гравцю екстрено покинути гру. Більше про роботу з користувацьким інтерфейсом у Unity можна дізнатися тут [15].

Висновки до розділу 2

У цьому розділі було розглянуто особливості розробки гри, яка одночасно є інструментом та середовищем для реалізації ігрового процесу для всесвіту D&D. Також були оцінені виклики та проблеми реалізації механік гри таких як: системи побудови та збереження рівнів, алгоритму управління процесом проведення гри, систему компілювання рівнів.

Також було розглянуто концепт гри, завдяки якому було правильно розподілено її функціонал, що в свою чергу дозволило розділити розробку гри на певні групи та етапи. З таким підходом розробка структуризовується і відповідно прискорюється. Це доводить факт користі даного методу розробки.

РОЗДІЛ 3. ПРОГРАМНЕ І ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Вимоги до технічного та програмного забезпечення

3.1.1 Характеристика технічного забезпечення платформи розробки та тестування

Для розробки гри було визначено такі вимоги до програмного забезпечення:

1. Unity -версія 2022.3.7f1.
2. C# - мова програмування.
3. Microsoft Visual Studio - середовище для написання коду.
4. Unity Asset Store - магазин ассетів, інструментів та бібліотек.

Також для планування і розподілення задач до виконання було використано середовище Jira.

Вимоги до технічного забезпечення:

1. Процесор: Intel Core i7-10750H
2. Відеокарта: NVIDIA GeForce GTX 1650Ti
3. Операційна система: Windows 10/11 64-розрядна
4. Оперативна пам’ять: 16 ГБ

3.1.2 Мінімальні вимоги до забезпечення

Визначений перелік мінімально необхідних вимог для запуску ігрового застосунку:

1. Операційна система Windows 10
2. Процесор: Intel Core i3-9100
3. Відеокарта: NVIDIA GeForce GTX 1050
4. Оперативна пам’ять: 8 ГБ
5. Місця на жорсткому диску: 1 ГБ

3.2. Опис програмної реалізації

3.2.1 Unity Asset Store

Проаналізувавши магазин Unity Asset Store та його безкоштовні ассети було обрано тему середньовіччя, яку добре можна реалізувати з доступними безкоштовними ассетами [16].

Головними критеріями для вибору цієї тематики були модель персонажів, середовище, моделі будинків, саунд візуальні ефекти. Обрані ассети в грі було обрано в офіційному магазині Unity - Unity Asset Store, посилання на них можна переглянути нижче в додатку.

3.2.2 Головне меню

Для розробки головного меню гри було використано такі елементи, як полотно(canvas) та його елементи панель(panel) і кнопки(button), текстові поля набору TextMesh Pro, фонове зображення, і спеціальний скрипт розроблений для головного меню (Рис. 3.1).

Рис. 3.1 Головне меню

Головне меню гри складається з групи таких об’єктів: заголовок “D&D Structurer”, кнопка “Створити сцену” і кнопка “Вийти”. Група цих об’єктів має назву “MainMenu”(Рис. 3.2)

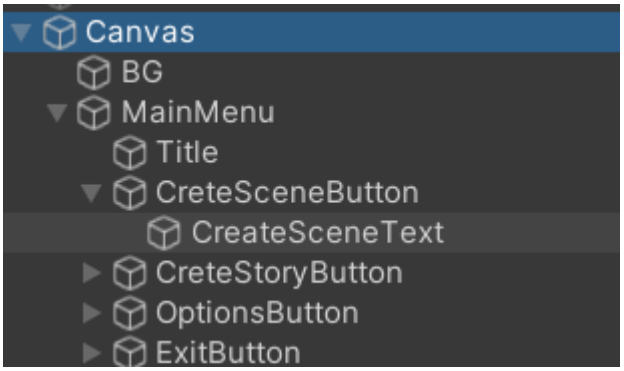

Рис. 3.2 Група об’єктів MainMenu, яка відповідає за функціонал головного меню

Джерело: Розроблено автором

До цієї групи об’єктів приєднаний скрипт “MainMenu”, який містить необхідні методи для функціоналу кнопок головного меню (Рис. 3.3).

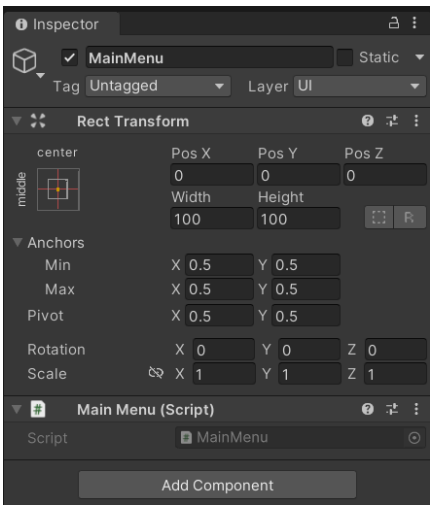

Рис. 3.3. Приєднаний скрипт MainMenu до групи об’єктів MainMenu

Джерело: Розроблено автором

В скрипті MainMenu є 2 методи розроблені для двох кнопок головного меню відповідно, метод “CreateScene” переносить гравця на іншу сцену, на якій можна будувати свої сцени, метод “QuitGame” закриває застосунок.

Для використання цього скрипту необхідно для кожної кнопки присвоїти відповідний метод скрипту. Щоб це зробити потрібно вибрати кнопку головного меню, у вікні ієрархії, у розділі “Button”, до події OnClick(), приєднати групу головного меню і вибрати, написаний раніше, відповідний метод. Приклад на основі

кнопки “Create Scene” (Рис. 3.4)

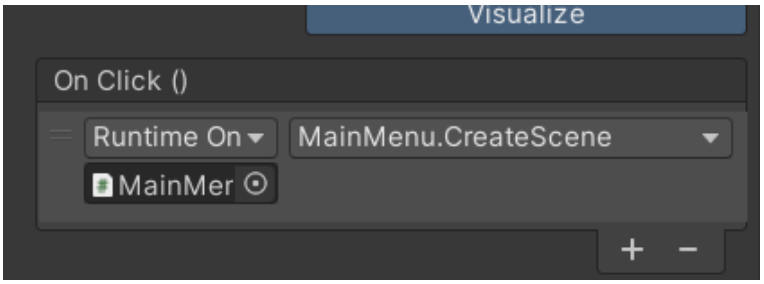

Рис. 3.4. Реалізація функціоналу кнопки Create Scene.

Джерело: Розроблено автором

Реалізація візуалізації взаємодії з кнопками відбувається через вікно Інспектора. Щоб створити необхідні візуальні ефекти, у розділі “Image” Потрібно вибрати будь який колір, який підходить. Далі в розділ “Button” налаштувати 3 пункти: “Normal Color”, “Highlighted Color” і “Pressed Color”. В кожному параметрі потрібно виставити різний ступінь прозорості, в першому 0, в другому можна встановити від 30 до 80, в третьому ж від 80 до 170. Приклад продемонстровано на кнопці “Create Scene” (Рис. 3.5)

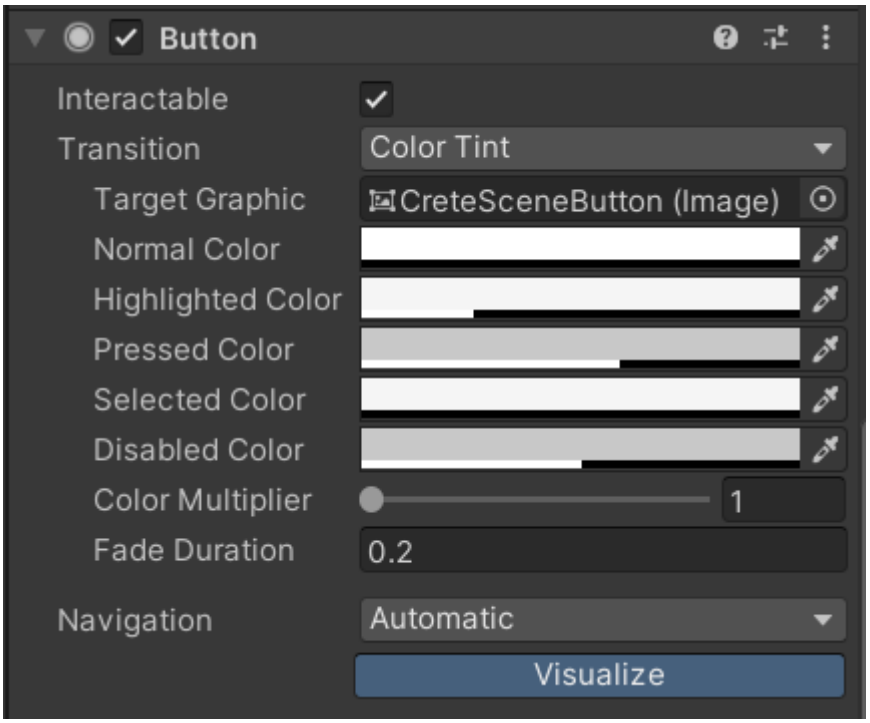

Рис. 3.5. Реалізація візуалізації взаємодії з кнопкою “Create Scene”

Джерело: Розроблено автором

3.2.3 Управління камерою

Для реалізації спостереження та переміщення по сцені було прийнято рішення розмістити спеціальний об’єкт, який називається “Camera Conteiner”. Всередині цього об’єкта поміщена звичайна камера та приєднаний скрипт “CameraMovement” (Рис. 3.6). Така структура об’єкту потрібна, щоб полегшити сприйняття та розділити управління параметрами камерою і її швидкістю.

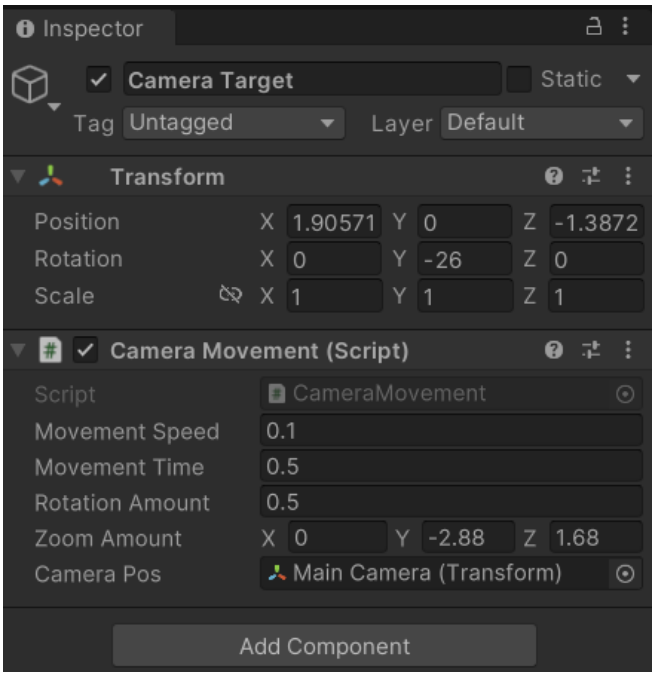

Рис. 3.6 Реалізація механіки управління камерою

Джерело: Розроблено автором

Переміщення працює за допомогою зміни координат в просторі, відповідно параметр “speed” дозволяє маніпулювати швидкістю переміщення камери. Сам спосіб переміщення через зміну координат в просторі не потребує різних взаємодій з фізикою, в якій немає необхідності і є простішим в реалізації.Також важливим аспектом є переміщення камери саме по глобальних координатах і в площині осей X та Z, це дозволяє переміщатися паралельно відносно площини де повинні встановлюватися об’єкти.

3.2.4 Розробка інтерфейсу

Розробка інтерфейсу цього меню схожа до розробки головного меню. Першим необхідним елементом буде “Canvas”. Просте створене полотно автоматично буде синхронізуватисяз камерою, якщо його не приєднати до будь якої камери. Це необхідно для правильного відображення інтерфейсу (Рис. 3.7).

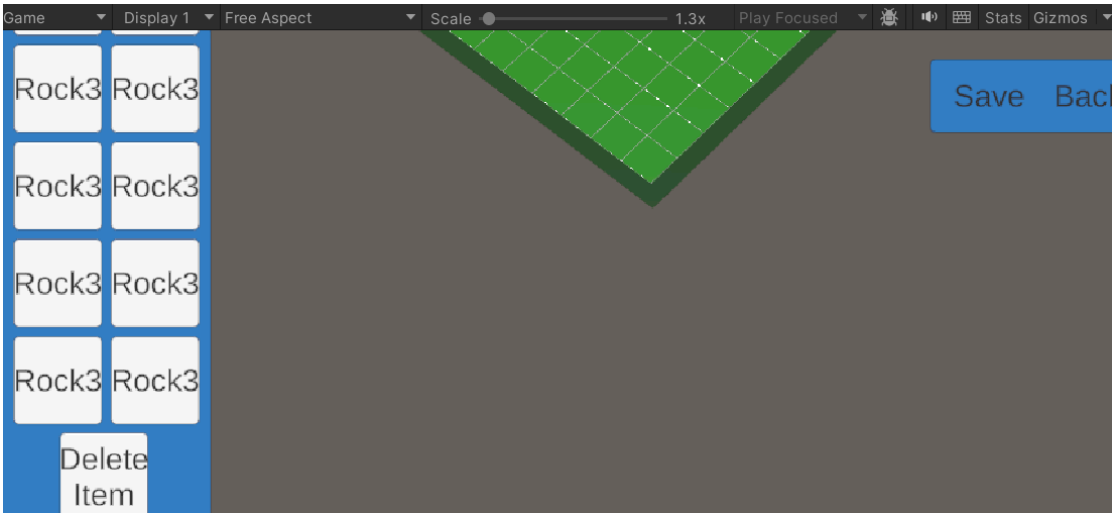

Рис. 3.7 Приклад вигляду синхронізованого полотна з камерою

Джерело: Розроблено автором

Механіка роботи інтерфейсу працює так само, як і головне меню гри. Тому на полотно потрібно поставити елементи кнопок та групу елементів під назвою “Scroll” в якій знаходяться фон списку і, будуть знаходитись його елементи (Рис. 3.8).

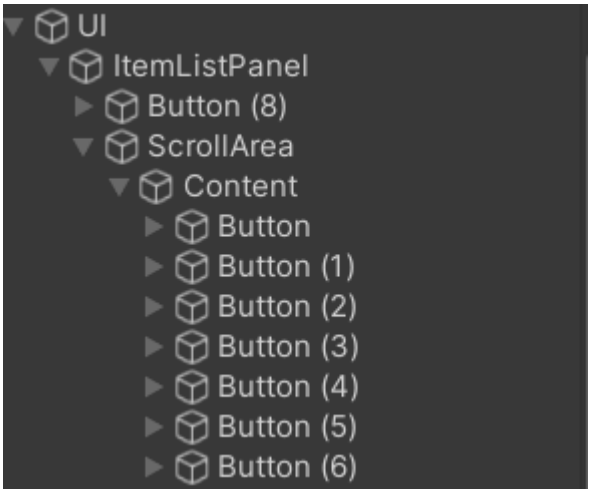

Рис. 3.8 Макет інтерфейсу інструменту.

Джерело: Розроблено автором

Механіка роботи з кнопок і елементів списку працює ідентично до механіки кнопок головного меню гри (Рис. 3.4).

3.2.5 Створення механіки для виставлення об’єктів на сцену та їхнього переміщення

Реалізації цієї механіки потребує потужної підготовки. Перший етап підготовки, це створення елементу платформи з компонентом “Grid”. Щоб його створити і почати роботу з ними, необхідно створити платформу будь якого виду, наприклад можна використати куб, задавши йому потрібні нам розміри. Другим елементом буде створення пустого елементу до якого буде додано компонент “Grid”(Рис. 3.9).

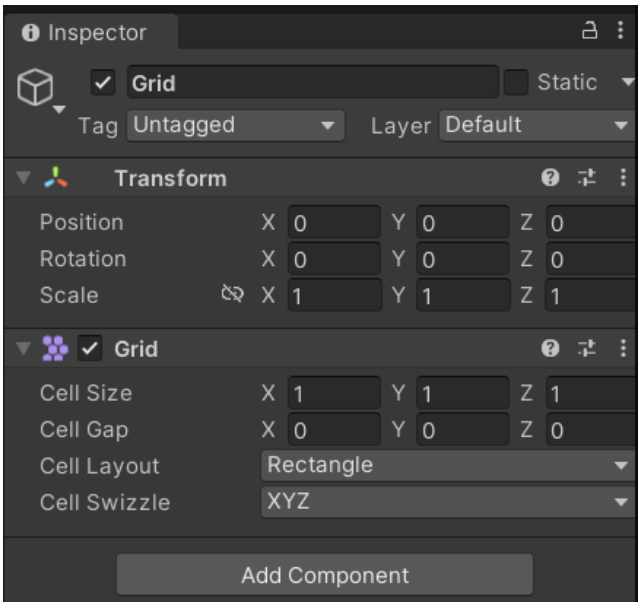

Рис. 3.9 Порожній елемент з компонентом “Grid”

Джерело: Розроблено автором

Наступним кроком буде об’єднання елементів порожнього елементу та елементу платформи. Для цього потрібно створити ще один порожній об’єкт, який буде вміщати в собі всі елементи необхідні для функціонування платформи (Рис. 3.10).

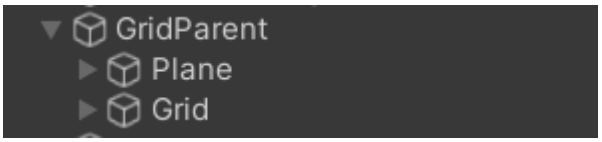

Рис. 3.10 Шлях створення Tilemap

Джерело: Розроблено автором

Далі необхідно налаштувати шари у вікні інспектора. Для коректної роботи механік потрібно створити свій власний шар, який буде пов’язаний тільки з системою встановлення. Для цього у вікні інспектора потрібно знайти випадаючий список з назвою “Layer” та вибрати порожній пункт де потрібно ввести назву свого шару (Рис. 3.11).

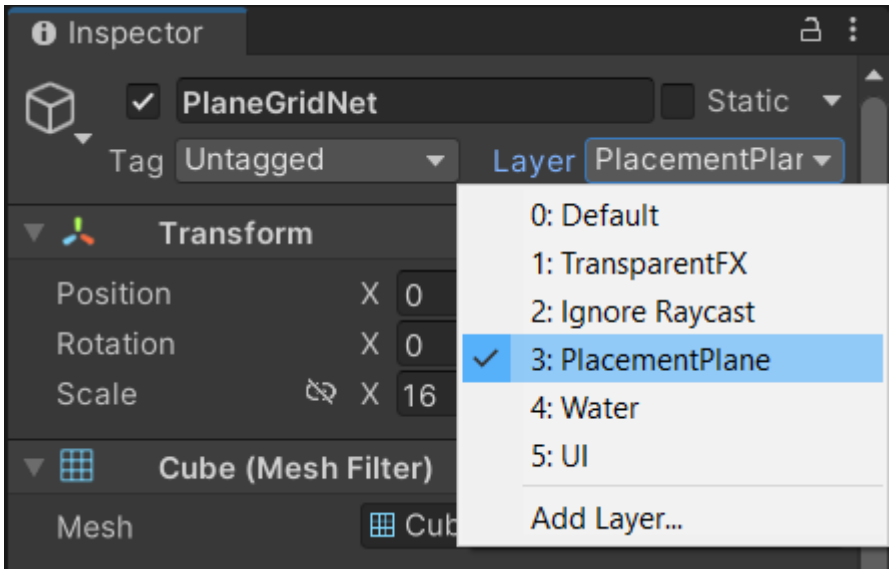

Рис. 3.11 Шар для механіки встановлення об’єктів

Джерело: розроблено автором

Після реалізації платформи наступним кроком буде створення префабів(prefab). Саме ці об’єкти будуть з’являтися на сцені. Кожен префаб повинне бути правильно вирівняти для встановлення на сітку. Найпростіший спосіб реалізації це створення батьківського елементу у якому буде знаходитись сама модель, що буде зміщена відповідно до свого розміру (Рис. 3.12).

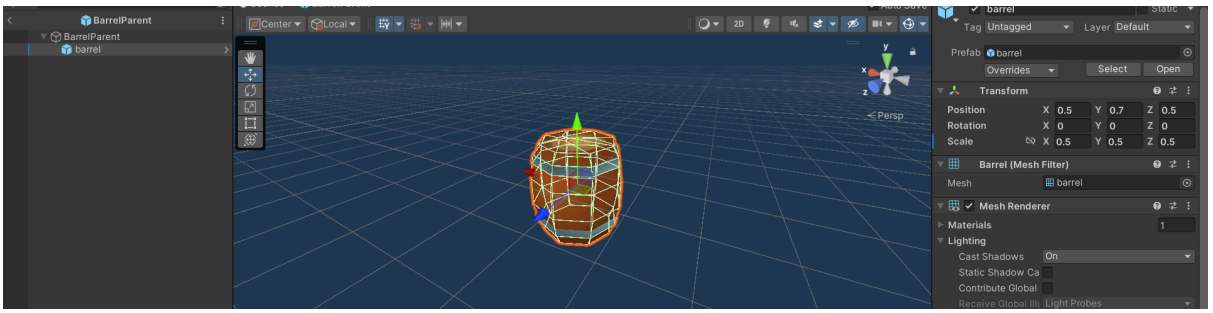

Рис. 3.12 Приклад створення і підготовки префаба

Джерело: створено автором

Далі потрібно створити скрипти “PlacebleObjectSystem”, “ObjectDrag” і “PlacebleObject”, які будуть відповідати за саме встановлення об’єктів на сцену. Методи скрипту “PlacebleObjectSystem”, як “GetMouseWorldPosition” і “SnapCoordinateToGrid” відповідають за знаходження місця появи об’єкту. Метод “InitializeWithObject” відповідає за появу об’єкта на сцені (Рис. 3.13). Методи скрипту “ObjectDrag” відповідають за механіку переміщення об’єкту по сцені (Рис. 3.13). Скрипт “PlacebleObject” потрібен, щоб розрахувати кількість місця, яка буде зайнята поставленим об’єктом, та щоб правильно встановлювати об’єкти, без конфліктів один з одним.

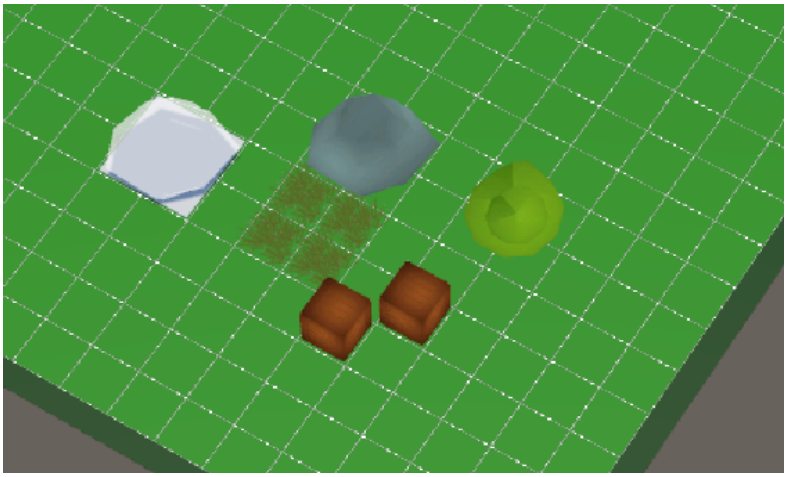

Рис. 3.13 Приклад роботи механіки встановлення об’єктів на сцену

Джерело: розроблено автором

3.2.6 Механіка роботи гральних кубиків

У даному проєкті було використано алгоритм лінійний конгруентний генератор (LCG) бібліотеки System.Random мови програмування C#. Оскільки він є одним з найпростіших способів генерації випадкового числа та належить до базових бібліотек мови C#. Також для забезпечення генерації випадкового числа у цьому проєкі використовується поточний час як сід. Оскільки, поточний час завжди є унікальним значенням, то генерація випадкового числа буде відбуватися завжди з унікального значення. Це робить генерацію випадкового числа ще біш не передбачуваною.

Алгоритм реалізації отримання випадкового числа був описаний у другому розділі за блок схемою зображеною вище (Рис. 2.8).

Цей метод дозволяє записати поточний час у значення, яке буде сідом для для генерації випадкового числа у заданому діапазоні.

3.2.7 Механіка збереження рівнів

Збереження рівнів є однією з найважливіших механік, оскільки на виді збереження буде базуватись і система реалізації завантаження рівнів для їхнього подальшого використання.

При реалізації цієї механіки потрібно вирішити одну задачу із записам даних у файл рівня. Оскільки при простих способах збереження виникає проблема запису даних у один й той самий файл, це призводить до втрати попередньо збережених даних.

Для вирішення цієї задачі було вирішено зробити систему збереження зі спеціальними слотами для рівнів, які і в подальшому допоможуть і у системі створення сюжетів, тому що усі збережені рівні вже будуть розподілені у спеціальних директоріях за однією структурою.

Для реалізації цієї механіки потрібно підготувати спеціальне меню зі слотами. Створення такого меню нічим не відрізняється від створення головного меню (Рис.

3.1). Це меню одноково має свою групу елементів відповідальну за функціонал (Рис. 3.2), свій скрипт (Рис. 3.3) і спосіб реалізації (Рис. 3.4), як у головного меню. Таке меню продемонстровано нижче (Рис. 3.14)

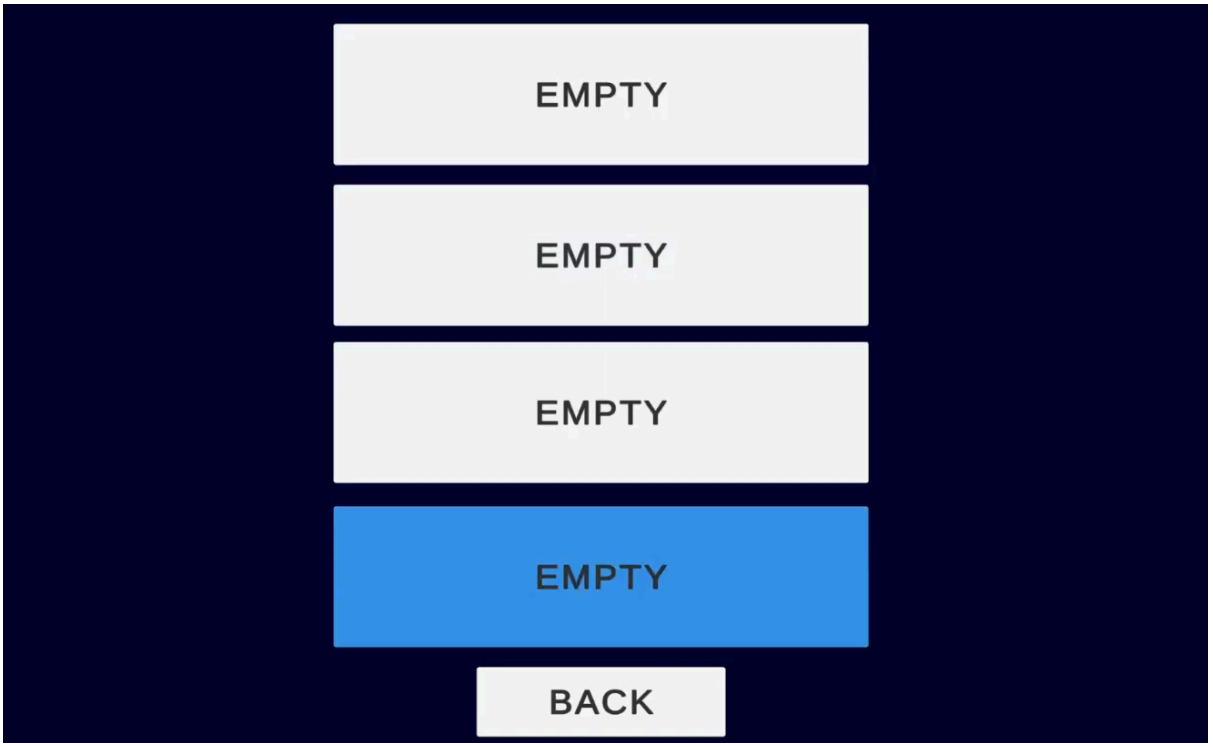

Рис. 3.14 Вигляд меню для слотів збереження

Джерело: розроблено автором

Наступним кроком буде реалізація системи збереження описана у другому розділі по схемі зображеною вище (Рис. 2.7).

3.2.8 Система переходів між сценами

Система переходу між сценами це остання механіка, що необхідна для реалізації ігрового процесу у D&D. Реалізується ця механіка на базі вже збережених слотів з готовими рівнями, схема роботи якої описана у другому розділі за схемою зображеною вище (Рис. 2.6).

3.3 Перспективи та подальші можливості розвитку застосунку

Хоч цей ігровий застосунок для створення та організації гри в D&D вже має необхідний функціонал, він також має чи малий потенціал для розвитку. В короткостроковій перспективі сюди вже можна буде додати велику кількість маленьких покращень, як покращені налаштування звуку, камери, стилів інтерфейсу чи навіть кубиків. А на подальші перспективи можна реалізувати мережеву гру, зробити спеціальні асети для гральних кубиків і реалізувати їхнє фізичне підкидування, розробити додатковий інтерфейс для листів персонажів, додати варіативні сцени для побудови рівнів, чи навіть створити систему наратора за допомогою штучного інтелекту.

Також значною мірою в розширені функціоналу значно можуть допомогти відгуки гравців, які можуть дати свої незалежні відгуки вказавши на хороші риси гри так і на недоліки, які варто буде виправити.

Висновки до розділу 3

В цьому розділі було описано використані засоби розробки було для створення гри, також були сформовані списки системних та технічних вимог для розробки гри і подальшого її запуску на своєму пристрої.

Також були описані способи реалізацій механік гри, описуючи наступні механіки та елементи гри: система бібліотеки елементів, система збереження рівнів, користувацький інтерфейс, механіка випадкового числа для кубиків, встановлення об’єктів на сцені, механіка переходів між рівнями, механіка побудови сюжетів, камера для огляду сцени, головне меню, UI.

Окрім реалізованих механік було розглянуто перспективи для подальшого розвитку гри та додавання нових механік.

ВИСНОВКИ

Під час роботи над даним проєктом було розроблено унікальну гру для створення та проведення D&D історій на ігровому рушії Unity. Основною ідеєю, цієї гри, було полегшення процесу розробки ігрових сцен для створення історій та їх проведення у світах Dungeons & Dragons. Ця гра значно спрощує та прискорює процес створення ігрової сцени і покращує отриманий досвід від сюжетів D&D та надає гейм-мастерам та гравцям нові можливості для реалізації ідей та концепцій у віртуальному ігровому середовищі.

Процес розробки гри включає в себе роботу з інтерфейсами, які потрібні для створення Головного меню гри чи інтерфейсу самого інструменту для створення сцен. Роботу з камерами, для покращеної взаємодії з різними об’єктами на сцені. Також робота з бібліотекою шаблонних об’єктів за допомогою систем координат, які реалізуються з “TileMap” та “TilePallete”, які необхідні для реалізації механік виставлення об’єктів на сцену. Створення та робота з префабами, які необхідні для створення бібліотеки елементів для побудови сцен.

Ця гра, що має функціонал для побудови рівнів і побудови з цих ж рівнів D&D історій, розроблений за допомогою на рушія Unity, досягнувши своїх цілей, відкриває для себе нові перспективи для покращень та подальших досліджень у сфері інтерактивних ігор. Його внесок полягає в заохоченні та розширенні творчих здібностей гейм-мастерів та гравців у створенні унікальних історій у світі Dungeons & Dragons.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GEEKACH [Електронний ресурс] URL: https://geekach.com.ua/nastolnye-igry/rolevye-igry/dungeons-dragon/

(дата звернення)

2. Mimic фантастична крамниця [Електронний ресурс] URL: https://mimic-dnd.com/nabir-akrylovykh-hralnykh-kubykiv-chervonyi

(дата звернення)

3. dicegeeks [Електронний ресурс] URL: https://www.dicegeeks.com/dnd-5e-character-sheet/

(дата звернення)

4. Steam [Електронний ресурс] URL: https://store.steampowered.com/app/286160/Tabletop_Simulator/

(дата звернення)

5. Steam [Електронний ресурс] URL: https://store.steampowered.com/app/2498570/Canvas_of_Kings/

(дата звернення)

6. Roll20 [Електронний ресурс] URL: https://roll20.net

(дата звернення)

7. Вікіпедія [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Список_ігрових_рушіїв

(дата звернення)

8. Unity [Електронний ресурс] URL: https://unity.com

(дата звернення)

9. Microsoft [Електронний ресурс] URL: https://visualstudio.microsoft.com

(дата звернення)

10. Microsoft [Електронний ресурс] URL: https://learn.microsoft.com/uk-ua/visualstudio/get-started/csharp/?view=vs-2022

(дата звернення)

11. Fandom [Електронний ресурс] URL: https://nri.fandom.com/uk/wiki/Класи_D%26D

(дата звернення)

12. itch.io [Електронний ресурс] URL: https://tamara-m.itch.io

(дата звернення)

13. Unity Documentation [Електронний ресурс] URL: https://docs.unity3d.com/6000.1/Documentation/ScriptReference/GridLayout.html

(дата звернення)

14. Microsoft [Електронний ресурс] URL: https://learn.microsoft.com/en-us/dotnet/api/system.random?view=net-9.0

(дата звернення)

15. Unity [Електронний ресурс] URL: https://unity.com/features/ui-toolkit

(дата звернення)

16. Unity Asset Store [Електронний ресурс] URL: https://assetstore.unity.com

(дата звернення)

ДОДАТОК А

Код скрипту InputManager:

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

public class InputManager : MonoBehaviour
{
    [SerializeField] private Camera sceneCamera;
    private Vector3 lastPosition;
    [SerializeField] private LayerMask placementLayermask;

    public event Action OnClicked, OnExit;

    private void Update()
    {
        if(Input.GetMouseButtonDown(0))
            { OnClicked?.Invoke(); }
        if (Input.GetKeyDown(KeyCode.Escape))
            { OnExit?.Invoke(); }
    }

    public bool IsPoinerOverUI()
        => EventSystem.current.IsPointerOverGameObject();
    public Vector3 GetSelectedMapPosition()
    {
        Vector3 mousePos = Input.mousePosition;
        mousePos.z = sceneCamera.nearClipPlane;
        Ray ray = sceneCamera.ScreenPointToRay(mousePos);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit, 100, placementLayermask))
        {
            lastPosition = hit.point;
        }
        return lastPosition;
    }

}
```

Код скрипту MainMenu:

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

public class MainMenu : MonoBehaviour
{
    public void CreateScene()
    {
        SceneManager.LoadScene(1);
    }

    public void CreateStory()
    {
        SceneManager.LoadScene(2);
    }

    public void QuitGame()
    {
        Application.Quit();
    }

}
```

Код скрипту CameraMovement:

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

public class CameraMovement : MonoBehaviour
{
    [SerializeField] private float movementSpeed;
    [SerializeField] private float movementTime;
    [SerializeField] private float rotationAmount;
    [SerializeField] private Vector3 zoomAmount;

    private Vector3 newPosition;
    private Quaternion newRotation;
    private Vector3 newZoom;
```

```
    [SerializeField] private Transform cameraPos;


    private void Start()
    {
        newPosition = transform.position;
        newRotation = transform.rotation;
        newZoom = cameraPos.localPosition;
    }

    private void Update()
    {
        MovementInput();
        RotationInput();
        ZoomCamera();
    }

    public void MovementInput()
    {
        if (Input.GetKey(KeyCode.W))
        {
            newPosition += (transform.forward * movementSpeed);
        }
        if (Input.GetKey(KeyCode.S))
        {
            newPosition += (transform.forward * -movementSpeed);
        }
        if(Input.GetKey(KeyCode.D))
        {
            newPosition += (transform.right * movementSpeed);
        }
        if(Input.GetKey(KeyCode.A))
        {
            newPosition += (transform.right * -movementSpeed);
        }
        transform.position = Vector3.Lerp(transform.position,
newPosition, Time.deltaTime + movementTime);
    }

    public void RotationInput()
    {
        if (Input.GetKey(KeyCode.Q))
```

```
        {
            newRotation *= Quaternion.Euler(Vector3.up *
rotationAmount);
        }
        if (Input.GetKey(KeyCode.E))
        {
            newRotation *= Quaternion.Euler(Vector3.up *
-rotationAmount);
        }
        transform.rotation = Quaternion.Lerp(transform.rotation,
newRotation, Time.deltaTime + movementTime);
    }

    public void ZoomCamera()
    {
        if (Input.GetAxis("Mouse ScrollWheel") > 0f)
        {
            newZoom += zoomAmount;
        }
        if (Input.GetAxis("Mouse ScrollWheel") < 0f)
        {
            newZoom -= zoomAmount;
        }
        cameraPos.localPosition = Vector3.Lerp(cameraPos.localPosition,
newZoom, Time.deltaTime + movementTime);
    }
}
```

Код класу ObjectsDatabaseSO:

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

[CreateAssetMenu]
public class ObjectsDatabaseSO : ScriptableObject
{
    public List<ObjectData> objectsData;
}

[Serializable]
```

```
public class ObjectData
{
    [field: SerializeField]
    public string Name {  get; private set; }
    [field: SerializeField]
    public int ID { get; private set; }
    [field: SerializeField]
    public Vector2Int Size { get; private set; } = Vector2Int.one;
    [field: SerializeField]
    public GameObject Prefab { get; private set; }
    [field: SerializeField]
    public Quaternion Quaternion { get; private set; }
}
```