

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка гри жанру "tile-matching" засобами Unity»

Виконав студент групи КН-20-1
ДЗЮБАНЧУК Богдан Сергійович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
СУГОНЯК Інна Іванівна

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
Марина ГРАФ
«25» січня 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: **ДЗЮБАНЧУК Богдан Сергійович**
Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**
Тема роботи: **«Розробка гри жанру "tile-matching" засобами Unity»**,
затверджена наказом закладу вищої освіти від **«25» січня 2024 р., № 22/с**
Термін здачі закінченої роботи: **«05» червня 2024 р.**
Вихідні дані роботи (зазначається предмет і об'єкт дослідження): об'єктом дослідження є процес проектування та розробки гри жанру tile-matching засобами ігрового рушія Unity. Предметом дослідження є використання рушія Unity, мови C# та графічних редакторів в процесі проектування та розробки гри жанру tile-matching.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд, аналіз доступного інструментарію та аналогічного ПЗ.	01.02.2024 – 25.02.2024	Виконано
2	Створення схем та структури застосунку.	26.02.2024 – 02.03.2024	Виконано
3	Розробка програмної частини.	03.03.2024 – 31.05.2024	Виконано
4	Тестування.	01.06.2024 – 08.06.2024	Виконано
5	Попередній захист.	05.06.2024	Виконано

Здобувач вищої освіти **Богдан ДЗЮБАНЧУК**
Керівник **Віталій ЛЕВКІВСЬКИЙ**

РЕФЕРАТ

Кваліфікаційна робота складається з програмного додатку – гри та пояснювальної записки

Пояснювальна записка до кваліфікаційної роботи складається з вступу, основної частини, висновків, списку використаної літератури та додатків.

У першому розділі було проведено аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи, створено дизайн документ та обрано інструменти для розробки гри.

У другому розділі було спроектовано алгоритми роботи та станів програмного застосунку, створено ігрові механіки, графічні елементи та користувацький інтерфейс.

У третьому розділі було проведено тестування застосунку та описано роботу із ним.

Текстова частина викладена на 97 сторінках друкованого тексту.

Пояснювальна записка має 12 сторінок додатків. Список використаних джерел містить 17 найменувань, і займає 2 сторінки. В роботі наведено 65 рисунків та 5 таблиць. Загальний обсяг роботи – 113 сторінок.

Ключові слова: ГРА, TILE-MATCHING, MATCH3, UNITY, 2D, UI, SAI TOOL, ГЕЙМДИЗАЙН, HUD, GUI, GEM.

					КН.КР.Б – 122 – 24 – ПЗ							
Змн.	Арк.	№ докум.	Підпис	Дата								
Розроб.		Б.С. ДЗЮБАНЧУК			«Розробка гри жанру "tile-matching" засобами Unity» Пояснювальна записка			Літ.		Арк.	Аркушів	
Керівник		В.Л.ЛЕВКІВСЬКИЙ								3	100	
Реценз.		І.І. СУГОНЯК						Житомирська політехніка, група КН-20-1				
Н. Контр.												
Затверд.		М.С. ГРАФ										

ABSTRACT

The qualification work consists of a game application and an explanatory note.

An explanatory note to the final thesis consists of an abstract, introduction, main part, general conclusion, a list of used literature and an appendix.

In the first section, we analyzed the existing software on the subject of the qualification work, created a design document and selected tools for developing the game.

In the second chapter, we designed the algorithms of the software application's operation and states, created game mechanics, graphic elements, and user interface.

The third section tests the application and describes how to work with it.

The text part is laid out on 97 pages of printed text.

The explanatory note has 12 pages of annexes. The list of used sources contains 17 names and occupies 2 pages.

The work contains 65 figures and 5 table. The total volume of work is 113 pages.

Keywords: GAME, TILE-MATCHING, MATCH3, UNITY, 2D, UI, SAI TOOL, GAME-DESIGNE, HUD, GUI, GEM.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ.....	10
1.1. Постановка задачі.....	10
1.2. Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи	11
1.3. Концепт-документ (Concept Document).....	15
1.4. Обґрунтування вибору інструментальних засобів	16
Висновки до першого розділу	17
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	18
2.1. Проектування алгоритмів роботи та станів програмного застосунку	18
2.2. Розробка ігрових механік	20
2.2.1. Розробка системи ігрової сітки.....	20
2.2.2. Розробка системи шефів.....	44
2.2.3. Розробка системи клієнтів.....	47
2.2.4. Розробка системи рівнів та їх побудови	51
2.2.5. Розробка системи даних гравця та їх збереження	55
2.2.6. Інтеграція реклами у гру	63
2.3. Створення графічних елементів	65
2.4. Створення користувацького інтерфейсу.....	72
Висновки до другого розділу	85
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ЗАСТОСУНКОМ ТА ЙОГО ТЕСТУВАННЯ	86

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		5

3.1. Порядок встановлення та налаштування системи	86
3.2. Інтерфейс та порядок роботи	86
3.3. Тестування роботи застосунку	93
Висновки до третього розділу.....	96
ВИСНОВКИ.....	97
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	98
ДОДАТКИ.....	100

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

GUI – Graphic User Interface – графічний інтерфейс, елементи за допомогою яких користувач взаємодіє з програмою, отримує данні

HUD – Heads-Up Display – частина графічного інтерфейсу користувача, яка дозволяє отримати різноманітну ігрову інформацію не викликаючи додаткові меню

SDK – Software Development Kit – набір із засобів розробки, утиліт і документації, який дає програмістам змогу створювати прикладні програми за визначеною технологією або для певної платформи

UI – User Interface – інтерфейс, елементи, з якими взаємодіє користувач

ПЗ – Програмне забезпечення

ПК – Персональний комп'ютер

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Смартфони в сучасному світі стали невід'ємною частиною нашого повсякденного життя, проте вони стали не просто засобом зв'язку, але й потужними ігровими пристроями. Популярність мобільних ігор полягає в тому, що вони доступні у будь-який час та в будь-якому місці. Це особливо корисно в сучасному світі, де люди постійно знаходяться в русі і не завжди мають час для довгих сесій ігор на консолях або ПК. Варто примітити, мобільні ігри часто є безкоштовними або доступними за невелику ціну, що також робить їх доступними для широкого кола користувачів. Це привертає увагу великої аудиторії, включаючи тих, хто не звик до ігрової індустрії.

Крім того, мобільні ігри, зокрема ті, що належать до жанру "tile-matching", відіграють важливу роль сьогодні, адже допомагають людям відволіктися від насичених негативом реалій та відпочити. Їх популярність зумовлена унікальною здатністю поєднувати розвагу, розвиток когнітивних навичок і релаксацію. Такі ігри вже десятиліттями приваблюють різні покоління гравців завдяки своїй доступності та простоті. Їх механіка, що базується на швидкому пошуку відповідностей і переміщенні елементів на ігровому полі, забезпечує легке входження для новачків і при цьому залишається захопливою для досвідчених гравців.

Релаксуюча роль ігор у жанрі tile-matching полягає в їх здатності забезпечувати розслаблюючий, але водночас захопливий досвід, який дозволяє гравцям забути про щоденні клопоти. Їх прості, але яскраві візуальні ефекти, приємний звуковий супровід і поступове підвищення рівня складності створюють ідеальне середовище для відпочинку та зняття стресу. Граючи в такі ігри, люди можуть зануритися в процес виконання нескладних, але приємних завдань, що сприяє покращенню емоційного стану та зниженню рівня тривожності.

Водночас, ігри жанру tile-matching виконують важливу функцію, сприяючи розвитку когнітивних навичок, таких як просторове мислення, увага до деталей і здатність до планування. Регулярна гра допомагає гравцям

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		8

покращувати свої аналітичні здібності, вчитися розпізнавати шаблони і вдосконалювати навички вирішення проблем. Це особливо корисно для розвитку дитячих когнітивних навичок, а також для підтримки мозку в активному стані у дорослих. Гравці навчаються оптимізувати свої рухи і розробляти стратегії для досягнення максимальних результатів, що позитивно впливає на їхню здатність приймати рішення і адаптуватися до змінних умов.

Таким чином, ігри в жанрі tile-matching залишаються важливими завдяки своєму балансу між розвагою і розвитком. Вони забезпечують можливість для миттєвого відпочинку та одночасно стимулюють розумову активність, що робить їх невід'ємною частиною сучасної ігрової культури.

Метою кваліфікаційної роботи є створення гри з використанням ігрових механік жанру tile-matching.

Об'єктом дослідження є процес проектування та розробки гри жанру tile-matching засобами ігрового рушія Unity.

Предметом дослідження є використання рушія Unity, мови C# та графічних редакторів в процесі проектування та розробки гри жанру tile-matching.

В рамках розробки кваліфікаційної роботи були опубліковані наукові праці:

Дзюбанчук Б. С., Терещук С. О., Object pool як засіб для оптимізації проєкту. Тези VI Всеукраїнської науково-технічної конференції "Комп'ютерні технології: інновації, проблеми, рішення", 29-30 листопада 2023 р. Житомир: Житомирська політехніка, 2023. С. 239-240

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

Темою кваліфікаційної роботи є «Розробка гри жанру "tile-matching" засобами Unity». Жанр "tile-matching" відноситься до групи головоломок, де основним завданням гравця є збірка плиток, зазвичай у вигляді груп або ланцюжків. Головна механіка полягає в тому, щоб утворити лінію з однакових плиток, яка може бути не тільки горизонтальною чи вертикальною, але і діагональною чи в згинаючому напрямку [1].

При розробці tile-matching-гри на платформі Unity необхідно враховувати кілька важливих аспектів, що дозволять забезпечити якісний ігровий досвід. Тому було виділено такі пункти:

1. Проектування внутрішньоігрової логіки гри:
 - 1.1. Опис загальної структури ігрового циклу у грі.
 - 1.2. Опис основних ігрових механік гри.
 - 1.3. Опис логіки монетизації гри.
2. Розробка основних механік:
 - 2.1. Розробка системи сітки та об'єктів в середині неї. Логічна поведінка ігрових об'єктів: падіння по сітці, взаємодія з іншими об'єктами сітки, взаємодія з гравцем.
 - 2.2. Розробка системи шефів.
 - 2.3. Розробка системи відвідувачів.
 - 2.4. Розробка системи рівнів, їх конструктора.
 - 2.5. Розробка системи ігрових валют та енергії.
 - 2.6. Розробка системи покращення шефів, інтер'єру.
 - 2.7. Розробка системи збереження даних гравця.
 - 2.8. Підключення SDK Unity ads.
3. Створення графічних елементів:
 - 3.1. Забезпечення відповідності загальному стилю гри та концепції.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		10

3.2. Розробка та інтеграція графічних елементів.

3.3. Проектування та реалізація кодової частини взаємодії користувачького інтерфейсу та гравцем.

4. Тестування ігрового продукту:

4.1. Перевірка правильності роботи ігрових механік.

4.2. Визначення, чи є можливість щось покращити.

4.3. виправлення помилок та усунення недоліків.

Зазначені вище пункти докладно описують ключові етапи розробки та реалізації гри у жанрі tile-matching. Загальний підхід до цих пунктів передбачає глибокий аналіз та взаємодію між усіма компонентами гри, а це створює підґрунтя для створення якісного та привабливого геймплейного досвіду для гравців

1.2. Аналіз існуючого програмного забезпечення за тематикою кваліфікаційної роботи

Для аналізу було обрано декілька ігор, що відповідають жанру tile-matching з його основними механіками.

«Best Fiend» (рис. 1.1) - free-to-play мобільна відеогра у жанрах головоломка та «три в ряд» і відповідна медіафраншиза. Станом на червень 2024 року оригінальну гру Best Fiends було завантажено більше 50 мільйонів разів з моменту її запуску у 2014 році на платформі Play Market [2].

Серед основних механік можна виділити те, що розробники змінили систему утворення лінії з однакових гемів, замінивши звичайний спосіб переміщення двох гемів на позиції один одного на утворення лінії власноруч: гравець проводить її так, щоб кожен наступний гем був поруч з поточним і відповідав тому ж типу, що й попередньо обраний (рис. 1.2).

Також основною механікою можна виділити створення загону з комах у боротьбі проти їх ворогів - слимаків, де кожна комаха відповідає конкретному типу гему та має свої власні бонуси й допоміжні геми.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		11

Проходячи рівні, гравець отримує ігрову валюту, яку він може використати для покращення кожної наявної у нього комахи.

На ігровому полі присутні 5 видів гемів. Рівень обмежений кількістю ходів та має цілі для завершення рівня.

Кожна спроба коштує дві енергії, при її максимумі у 15. Енергія поступово відновлюється з часом, або переглянувши рекламу з винагородою.

До мінусів гри можна віднести надокучливі реклами та постійні спливаючі вікна про щоденні, щотижневі або сезонні події, що зменшують задоволення від ігрового процесу.



Рисунок 1.1 – Логотип гри Best Fiends



Рисунок 1.2 – Ігровий процес гри Best Fiends

«Homescapes» (рис. 1.3) - це гра-головоломка Match-3, що зберігає звичайний спосіб утворення лінії з гемів, де основний процес базується на обміні

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		12

двох суміжних елементів, щоб зробити рядок або колонку щонайменше з трьох елементів, яких у грі є 5 видів (рис. 1.4). Станом на червень 2024 року гру було завантажено більше 100 мільйонів разів з моменту її запуску у 2017 році [3].

Кожен рівень у грі обмежений кількістю ходів, проте, на відміну від гри Best Fiend, тут є система зірок: менше ходів потратили – більше зірочок отримали. Задля спрощення ігрового ладу є допоміжні інструменти, які можна отримати за проходження конкретних рівнів, або за внутрішньо-ігрову купівлю .

Зірки основна валюта, щоб рухати сюжет гри. За них здійснюється розблокування недоступних місць, а також можливість видозмінити елементи декору.

Кількість спроб обмежена серцями, що також, як і у передньому прикладі, відновлюються протягом часу і можна отримати за перегляд реклами або за внутрішньо-ігрову купівлю.

Гра не має надокучливої реклами та спливаючих вікон, що зберігає отримання задоволення від ігрового процесу.



Рисунок 1.3 – Логотип гри Homescapes



Рисунок 1.4 – Ігровий процес гри

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Candy Crush Saga (рис. 1.5) - багатоплатформова безкоштовна (free-to-play) відеогра в жанрі «три в ряд», що також зберігає в собі звичайний спосіб об'єднання гемів, яких у грі також 5 (рис. 1.6), розроблена компанією King Digital [4]. Вона є одним з перших та найуспішніших використань бізнес моделі, яка дозволяє пройти гру повністю безкоштовно, проте пропонує можливості, які полегшують її проходження, за додаткову плату.



Рисунок 1.5 – Логотип гри Candy Crush Saga



Рисунок 1.6 – Ігровий процес гри Candy Crush Saga

Отже, проаналізувавши аналоги, врахуємо наступні критерії перед початком створення власної гри:

- простий та зручний інтерфейс;
- наявність 5 різних видів гемів;
- різноманіття ігрового процесу, ускладнена будовою рівня, або розширена додаванням нової ігрової механіки;
- правильно скорегована складність;
- присутнє обмеження у кількості спроб;
- не зловживання рекламою та спливаючими вікнами.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				14
Змн.	Арк.	№ докум.	Підпис	Дата		

1.3. Концепт-документ (Concept Document)

Жанр і цільова аудиторія.

Жанр – tile-matching.

Гра розрахована для людей різного віку через її простоту та релаксуючий ігролад.

Основні особливості гри.

Гра надає гравцеві можливість відчувати себе на місці власника свого літнього кафе.

Гемплей.

Гра матиме 5 видів продуктів: м'ясо, риба, випічка, напої та зелень. Утворення лінії об'єднання буде виконуватись наступним чином: гравець проводить її так, щоб кожен наступний гем був поруч з поточним і відповідав тому ж типу, що й попередньо обраний. Об'єднання лінії матиме своє задоволення, що надаватиметься клієнтам. При обслуговуванні клієнта, гравець отримує винагороду у вигляді ігрової валюти.

Ігрові рівні являють собою календар подій. Кожного дня є свої завдання. Рівні обмежені ходами та мають систему зірок: менше ходів було затрачено – більше зірочок отримано. У грі поступово з'являтимуться нові ігрові механіки, що будуть урізноманітнювати ігролад. На освоєння кожної з них виділено кількість рівнів, що поступово навчатимуть гравців ними користуватись.

Кількість спроб обмежено енергією, що відновлюватиметься протягом часу, або за перегляд реклами. Кожна спроба коштує 2 енергії.

Зірки отримані за проходження рівнів можна буде потратити на покращення самого кафе. Покращення є наступних типів:

- покращення стільців – збільшує кількість отриманих коштів за обслуговування клієнтів;
- покращення столів – збільшує отримувану кількість задоволення клієнтів;
- покращення світильників – збільшує максимальну кількість енергії у грі.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		15

Протягом гри будуть відкриватись шефи, кожен з яких має свій бонус до конкретного типу продукту та властивість, що допомагатиме гравцеві у проходженні рівня. Таких допоміжних властивостей буде декілька, а саме:

- вертикаль – руйнує два геми по обидва боки вертикально від зруйнованої бонусної властивості;
- горизонталь – об'єднує два продукти по обидва боки горизонтально від зруйнованої бонусної властивості;
- ящик – перетворює навколо себе продукти на тип продукту, якому він відповідає;
- каструлька – об'єднує навколо себе продукти у певному радіусі.

Монетизація.

Монетизація здійснюватиметься шляхом підключення SDK Unity ads [5]. Буде використано два види реклами, а саме інтерстиціальна, що відображатиметься на весь екран, які з'являються під час переходу між рівнями, екранами або іншими подіями в грі, та в обмін на нагороду, що дозволяє гравцям отримати ігрові бонуси, валюту або енергію, що стимулюватиме гравців до перегляду реклами.

Музичний супровід.

Так як гра є спокійною і помірною, музичний супровід має бути відповідний. Мелодія не має заважати ігровому процесу, водночас має додавати атмосфери під час ігрового процесу.

Платформа.

Гра буде розроблятися для платформи Android.

1.4. Обґрунтування вибору інструментальних засобів

Існує різна кількість ігрових рушіїв, що мають свої переваги та особливості, але одним з найпопулярніших і широко використовуваних є Unity. Існують інші платформи, такі як Unreal Engine, CryEngine, Godot Engine та Lumberyard, які також пропонують потужні можливості для розробки ігор, однак, Unity відзначається своєю простотою, широким спектром можливостей та

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		16

підтримкою багатьох платформ, включаючи мобільні пристрої. Серед усіх ігрових рушіїв Unity найпростіший у реалізації 2D ігор на мобільну платформу. Це робить Unity хорошим вибором для розробки даної ігри.

Окрім того, Unity пропонує зручне та просте у використанні SDK з рекламою, що дозволяє легко інтегрувати рекламу в гру, а це є важливим аспектом для монетизації гри.

Щодо художніх редакторів, серед них можна виділити такі, як Photoshop, GIMP, Blender, і Krita. Однак, SAI Tools [6] має простий та інтуїтивний інтерфейс, що спрощує роботу художникам-початківцям. Крім того, SAI Tools надає широкі можливості для роботи з пензлями та векторами, що робить їх особливо корисними для створення художньої складової гри.

Висновки до першого розділу

В даному розділі було проведено детальний аналіз процесу проектування програмного забезпечення для гри у жанрі tile-matching. Визначено практичне завдання, що включає в себе продумування внутрішньоігрових механік, реалізацію основних механік гри, створення графічних елементів та тестування.

Було проаналізовано три популярні гри у вибраному жанрі, визначеної особливості й переваги. Створено концепт-документ, в якому визначено жанр та аудиторію, особливості, геймплей, сюжет та платформу. Обґрунтовано вибір інструментарію, а саме вибір Unity серед рушіїв гри, через його простоту у розробці мобільних 2D ігор, та вибір SAI tools серед графічних інструментів, адже він став оптимальним вибором для роботи з графікою, враховуючи його інтуїтивний інтерфейс та розширені можливості. Створено основу для подальшого розвитку та реалізації проекту, забезпечуючи чітке розуміння завдань та етапів роботи над грою у жанрі tile-matching.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		17

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Проектування алгоритмів роботи та станів програмного застосунку

Перед початком створення ігрового застосунку, доречним буде створити діаграму варіантів використання [7], що представляє собою UML-діаграму (рис. 2.1), яка в графічному вигляді відображає вимоги до розроблюваної системи.



Рисунок 2.1 – Діаграма варіантів використання

Діаграма дає зрозуміти як гравець, що являє собою зовнішньою сутністю, взаємодіє з системною послідовністю дій для досягнення певних цілей, а саме:

1. Покращення шефів доступне після переходу у відповідне меню, що також надає можливість отримати інформацію про усіх шефів доступних наразі.
2. Покращення або зміна вигляду інтер'єру потребує переходу у відповідну сторінку у застосунку.
3. Проходження рівнів доступне з відповідного меню. Також там гравцеві доступний перегляд реклами для отримання бонусів.

Після чого приступимо до створення діаграми активностей (рис. 2.2), що описує, як діяльність координується для виконання дії, яка може бути на різних рівнях абстракції [8].

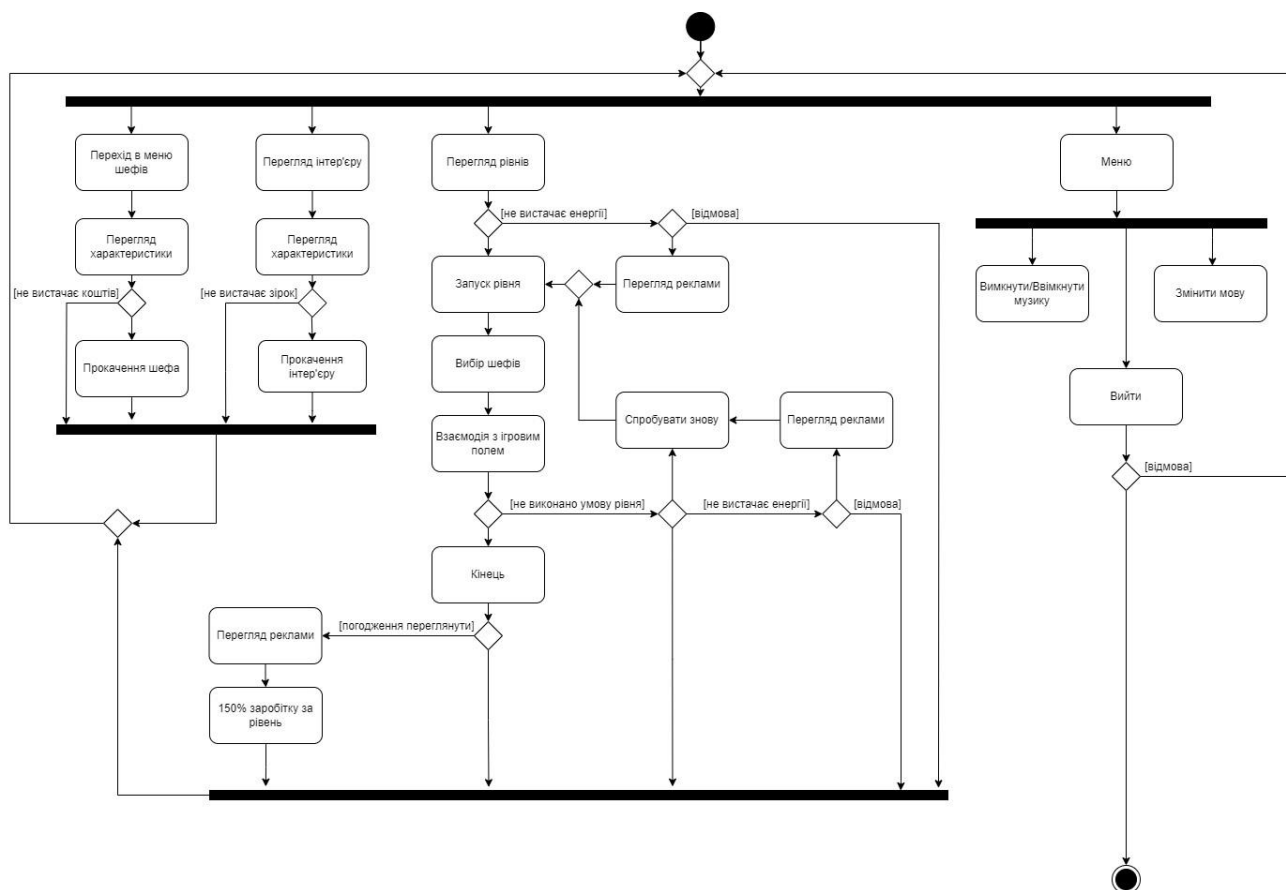


Рисунок 2.2 – Діаграма варіантів використання

Опис діаграми активностей:

1. Перехід в меню шефів. Гра надає можливість відкривати поступово нових шефів, що будуть доступні у відповідному меню. Гравець має можливість переглянути характеристику шефів та покращити їх, якщо вистачає у нього ігрової валюти.

2. Перехід на сторінку інтер'єру. Гра надає можливість покращувати рівні інтер'єру, якщо у гравця достатня кількість зірок, що взамін надаватиме гравцеві додаткові бонуси. Відповідне меню надає можливість гравцеві це зробити. Крім того зміна вигляду інтер'єру доступна у цьому ж меню.
3. Доступ до рівнів гравець отримує при переході у меню рівнів. Якщо у гравця не достатня кількість енергії, йому пропонується переглянути рекламу для доступу до рівня. Після початку рівня гравець обирає шефів та приступає до виконання завдань рівня. Якщо завдання не було виконано за відведену кількість ходів, гравцеві надається можливість спробувати знову, якщо у нього вистачає на це енергії, якщо ні – пропонується перегляд реклами. Після успішного завершення рівня, гравцеві пропонується отримати 150% заробітку також за перегляд реклами. З цього меню гравець може повернутись у головне меню.
4. Меню налаштувань дає можливість змінити мову та вимкнути, або ввімкнути звук, а також вийти з гри, що і є кінцем діаграми активностей.

Після проаналізування вище описаних діаграм, переходимо безпосередньо до розробки ігрових механік.

2.2. Розробка ігрових механік

Розробка ігрових механік охоплює велику кількість частин, тому вирішено поділити цей процес на підпункти, що конкретно й детально описуватимуть відповідні моменти розробки. Усі анімації у грі будуть здійсненні за допомогою бібліотеки DOTween [9], що надає можливість створювати анімації всередині коду.

2.2.1. Розробка системи ігрової сітки

Для початку потрібно виділити основні елементи, що будуть задіяні у системі ігрової сітки, а саме:

- сітка;

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		20

- клітинка сітки;
- об'єкт клітинки.

Створимо три сценарії, що відповідатимуть даним елементам та перейдемо до реалізації сітки.

Сітка являє собою двовимірний масив, що матиме свою висоту та ширину. Цей масив заповнюватиметься елементами, що являють собою клітинками.

Додаємо конструктор, що буде приймати у себе висоту, широту та масив клітинок. Лістинг коду:

```
public Grid(int width, int height, GridCell[,] cells)
{
    Width = width;
    Height = height;
    Cells = cells;
}
```

Для зручності керуванням сіткою та отриманням потрібної інформації додамо методи, що дозволять це зробити. Отже, метод перевірки, чи являються координати в межах сітки. Лістинг коду:

```
private bool IsCoordInRange(int x, int y)
{
    return x >= 0 && y >= 0 && x < Width && y < Height;
}
```

Метод отримання клітинки по її координатах. Лістинг коду:

```
public GridCell GetCell(int x, int y)
{
    if (IsCoordInRange(x, y))
    {
        return Cells[x, y];
    }
    Debug.LogError($"Координати не в межах сітки {x}, {y}");
    return null;
};
```

Метод отримання усього рядка відповідної клітинки. Лістинг коду:

```
public List<GridCell> GetRow(GridCell cell)
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				21
Змн.	Арк.	№ докум.	Підпис	Дата		

```

{
    List<GridCell> rowCells = new List<GridCell>();
    int y = cell.y;
    for (int x = 0; x < Width; x++)
    {
        if (x == cell.x) continue;
        rowCells.Add(GetCell(x, y));
    }
    return rowCells;
}

```

Метод отримання частини рядка, починаючи від клітинки. Метод приймає у себе клітинку, кількість потрібних об'єктів, параметр, що вказує на те, що йде запит по обидва сторони від клітинки, чи в окремому напрямку, що вказує наступний параметр. Визначається початкова клітинка відліку та кінцева, після чого здійснюється заповнення динамічного масиву відповідно умовам функції continueCondition. Лістинг коду:

```

public List<GridCell> GetRow(GridCell cell, int count, bool
isTwoWayCheck = true, bool isRightCheck = false)
{
    int x = cell.x;
    int y = cell.y;
    int startX = isTwoWayCheck ? x - count : (isRightCheck ? x :
x - count);

    Func<int, bool> continueCondition = isTwoWayCheck ?
        (checkX => checkX < x + count)
    :
        (checkX => isRightCheck ?
checkX < x + count : checkX < x);
    List<GridCell> resultCells = new List<GridCell>();
    for (int checkX = startX; continueCondition(checkX); checkX++)
    {
        if (IsCoordInRange(checkX, y))
        {

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				22
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        if (checkX == cell.x) continue;
        resultCells.Add(GetCell(checkX, y));
    }
}
return resultCells;
}

```

Метод отримання усього стовпця та його частини відповідної клітинки, реалізований аналогічно методам отримання рядка та його частини.

Метод отримання сусідніх клітинок відповідної клітинки та клітинок, що навколо відповідної клітинки.

Перейдемо до реалізації елементів клітинки, створивши базовий клас, що містить у собі поле, яке інформує про те, чи елемент піддається гравітації. Усі рухи будуть реалізовані бібліотекою DOTween, що надасть візуальної зрозумілості процесу руху елементів по сітці шляхом відслідковування завершення руху. Об'єкт має у собі метод появи у сітці, що приймає у собі місце імітації появи у заданій точці і клітинку, у яку буде здійснено рух, та метод руху елемента у вказану клітинку. Методи змінюють батьківський об'єкт та рухають локально елемент у його центр. Усі два елементи повертатимуть значення Tween [10], яке й має в собі інформацію про рух даного елемента. Лістинг відповідних методів:

```

virtual public Tween Spawn(Transform spawnPos, GridCell
gridCell)
{
    transform.position = new
Vector2(gridCell.transform.position.x, spawnPos.position.y);
    return SetGridCoord(gridCell);
}
virtual public Tween SetGridCoord(GridCell gridCell)
{
    transform.SetParent(gridCell.transform);
    transform.localScale = Vector2.one;
    return transform.DOLocalMove(Vector2.zero, 0.075f);
}

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		23

Для майбутнього перевизначення також додано два методи знищення об'єкту та його очищення. Варто відмітити, що знищення здійснюватиметься асинхронно, та прийматиме у себе функцію, що здійснюватиметься під час знищення, та об'єкт з яким буде йти взаємодія.

Клітинка має свої координати, що зберігатимуться у відповідних полях x та y. Також зберігає у собі об'єкт, що являє собою елементом клітинки. Існує декілька варіантів присвоєння об'єкта, а саме: створення нового елементу та зміна позиції у сітці існуючого. Для цих варіантів й створимо відповідні методи, що здійснюватимуть створення або рух елементу, та збереження його у цій клітинці. Так як методи, які знаходяться у елементах, повертають значення Tween, то й ці методи повертатимуть таке ж значення. Лістинг коду:

```
private void SetGridObject(GridObject gridObject)
{
    GridObject = gridObject;
    Gem gem = GridObject as Gem;
    if (gem != null)
    {
        Gem = gem;
    }
}

public Tween SpawnObject(Transform spawnPoint, GridObject
gridObject)
{
    SetGridObject(gridObject);
    return gridObject.Spawn(spawnPoint, this);
}

public Tween SetObject(GridObject gridObject)
{
    SetGridObject(gridObject);
    return gridObject.SetGridCoord(this);
}
```

Також реалізуємо декілька методів для руйнування елементу в середині клітинки, очищення клітинки й очищення поля елементу. Пам'ятаємо, що

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		24

знищення клітинок здійснюється асинхронно, тому цей метод також повертатиме Task, якщо клітинка є не порожньою, під час виклику методу методу Destroy, передається функція SetEmpty та місце, куди рухатиметься зруйнований елемент. Крім того метод знищення викликатиме відповідну подію, якщо було зруйновано саме елемент типу Gem.

Також створимо клас PointManager, що буде обробляти події натискання, тримання й відпускання вказівника. У ньому реалізуємо інтерфейси з UnityEngine.EventSystems [11], а саме: PointerDownHandler, IPointerEnterHandler, IPointerUpHandler. Додамо мето налаштування, який прийматиме в себе функції, які будуть викликатись під час виконання відповідних дій. Лістинг коду описаних методів у менеджері:

```
public void Setup(Action down, Action enter, Action up)
{
    Down = down;
    Enter = enter;
    Up = up;
}

public void OnPointerDown(PointerEventData eventData)
{
    Down?.Invoke();
}

public void OnPointerEnter(PointerEventData eventData)
{
    Enter?.Invoke();
}

public void OnPointerUp(PointerEventData eventData)
{
    Up?.Invoke();
}
```

Також створимо сценарій для глобальних ігрових подій, використавши патерн одинак [12] та спостерігач [13]. Він міститиме в собі потрібні ігрові

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		25

події, які викликатимуться з різних місць та оброблятиметься у тих сценаріях, які підписані на ці події.

Одразу перейдемо до реалізації двох ігрових механік пов'язаних з клітинками сітки, які додадуть різноманітності для ігрового досвіду гравця.

Почнемо з механіки очищення «підлоги» клітинки. Для цього нам потрібно сценарії для самої підлоги, об'єкта сценарію, що міститиме в собі інформацію окремого типу підлоги, об'єкт сценарію, що міститиме в собі усю інформацію про типи підлоги, та менеджера, що буде на ігровому об'єкті клітинки.

Тип підлоги визначається кольором, тому для об'єкту сценарію створимо відповідні поля: тип та масив кольорів. Для об'єкта, який зберігатиме усі типи підлог разом додамо зручний метод, який надаватиме можливість отримати потрібну інформацію за типом підлоги. Лістинг коду цього об'єкту:

```
[SerializeField] FloorS0[] floors;

public FloorS0 GetByType(FloorType type)
{
    foreach (FloorS0 flor in floors)
    {
        if (flor.type == type)
            return flor;
    }
    return null;
}
```

Сам сценарій підлоги матиме метод налаштування, який приймає в себе об'єкт сценарію з відповідними даними для потрібного типу. Кількість кольорів у масиві означає те, скільки продуктів потрібно об'єднати на клітинці, щоб очистити її повністю. Так як ми завчасно створили подію в сценарії клітинки, що повідомляє про знищення гему на ній, додамо метод у сценарій підлоги, що підписуватиметься на відповідну подію та метод, що буде оброблювати це. Щоразу як на клітці об'єднується продукт – колір змінюється на наступний до тих пір, поки не закінчиться масив. Як тільки досягаємо кінця масиву

викликається подія, що повідомляє про те, що підлога була очищена. Лістинг коду обробки події знищення гему на клітинці:

```
private void OnGemDestroyedNierby()
{
    hp--;
    if (hp <= 0)
    {
        OnFloorDestroy?.Invoke();
        EventManager.instance.OnFloorCleaned?.Invoke();
        cell.OnGemDestroyinCell -= OnGemDestroyedNierby;
        Destroy(gameObject);
        return;
    }
    image.color = sprites[hp - 1];
}
```

Перейдемо до реалізації менеджера. Він буде зберігати в собі інформацію про поточний тип підлоги у клітинці та матиме два методи: створює екземпляри по типу та очищає клітинку від них. Для створення екземплярів потрібні також два поля, які будуть містити в собі префаб об'єкту та об'єкт сценарію, який містить усю інформацію про типи.

Підключимо цей менеджер до сценарію клітинки. Додамо метод, що викликатиме метод створення екземпляру підлоги.

Наступною механікою є стінки. Для цього нам також потрібно сценарії безпосередньо самої стінки, об'єкта сценарію, що міститиме в собі інформацію про тип стінки, та об'єкт, який міститиме у собі інформацію про усі типи стінок, та менеджер, який буде прикріплений до ігрового об'єкту клітинки.

Кількість здоров'я стінки буде визначатись кількістю спрайтів відведених під конкретний тип.

Сама стінка матиме поля, що зберігатиме інформацію про напрям та її тип. Метод налаштування буде приймати у себе інформацію з об'єкту сценарію, напрям та тип. Також невідмінною частиною є метод підписання на події знищення гемів у клітинці, у якій знаходиться стінка, та клітинці, що знаходиться

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		27

поруч відповідно напрямку стінки. Цей метод прийматиме у себе масив усіх клітинок, на події яких і буде здійснюватися підпис. Метод, що оброблятиме подію буде змінювати спрайт на наступний у масиві до моменту завершення його. Якщо досягнуто кінець масиву – викликається подія, що повідомляє про це. Лістинг коду даного методу:

```
private void OnGemDestroyedNierby()
{
    hp--;
    if (hp <= 0)
    {
        OnBorderDestroy?.Invoke(direction);
        foreach (GridCell c in cells)
        {
            c.OnGemDestroyinCell -= OnGemDestroyedNierby;
        }
        Destroy(gameObject);
    }
}
```

Перейдемо до реалізації менеджера. Він зберігатиме у собі словник усіх стінок, ключами якого будуть напрямки. Метод додання стінки до клітинки та підписом на її подію про знищення після чого йде здійснення виклик методу підпису у стінці, передаючи потрібні клітинки. Для цього потрібний допоміжний метод, що отримуватиме інформацію з сітки про потрібні клітинки відповідно напрямку, який буде додано. Лістинг даного методу:

```
private List<GridCell> GetCells(Direction dir)
{
    int x = cell.x;
    int y = cell.y;
    List<GridCell> cells = new() { cell };
    switch (dir)
    {
        case Direction.Top:
            if (y < grid.Height - 1)
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		28

```

        cells.Add(grid.GetCell(x, y + 1));
        break;
    case Direction.Bottom:
        if (y > 0)
            cells.Add(grid.GetCell(x, y - 1));
        break;
    case Direction.Left:
        if (x > 0)
            cells.Add(grid.GetCell(x - 1, y));
        break;
    case Direction.Right:
        if (x < grid.Width - 1)
            cells.Add(grid.GetCell(x + 1, y));
        break;
    }
    return cells;
}

```

Додамо метод очищення клітинки від стінки за потрібним напрямком, та метод отримання інформації про усі наявні стінки. Також допоміжний метод, що повідомлятиме про наявність стінки за конкретним напрямком.

Підключимо цей менеджер до сценарію клітинки. Додамо метод, що викликатиме метод створення екземпляру стінки за напрямом й типом та метод, що перевірятиме наявність стінки у потрібному напрямку.

Перейдемо до створення елементу у клітинки типу Gem. Для цього потрібно створити об'єкт сценарію, що міститиме у собі інформацію про тип гему, його вигляд та кількість очок, які буде надано після зруйнування, та об'єкт сценарію, що міститиме у собі усю інформацію про кожен тип гему.

Сценарій Gem наслідуватиме поведінку сценарію GridObject, перевизначивши й додавши нові методи. Метод налаштування приймає інформацію про тип гему та змінює спрайт об'єкту на відповідний спрайт гему та вказуватиме, що об'єкти такого типу піддаються гравітації.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Потрібно врахувати, що іноді об'єкти такого типу будуть забрані з ігрового поля без отримання очок за них, тому буде для цього буде перевизначено два методи Destroy і Clear, з поміжним методом, який пригодиться у майбутньому. Лістинг частини коду:

```

    override public async Task Destroy(Action callback, Transform
    target)
    {
        Deactivate();
        await MoveTo(target).AsyncWaitForCompletion();
        OnGemDestroy?.Invoke(this);
        EventManager.instance.OnGemDestroy?.Invoke(this);
        DeactivateGem();
        callback();
    }
    override public void Clear()
    {
        DeactivateGem();
    }
    protected void DeactivateGem()
    {
        OnGemDeactivate?.Invoke(this);
    }
    virtual protected void Deactivate()
    {
        arrow.SetActive(false);
        SetActive(false);
        transform.localPosition = Vector2.zero;
        transform.localScale = Vector2.one;
        transform.localRotation = Quaternion.identity;
    }

```

Також у методі знищення здійснюється виклик методу руху об'єкту до позиції, який використовує бібліотеку DOTween та повертає значення Tween для відслідковування завершення цієї дії.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		30

Геми - елемент, що буде постійно руйнуватись і з'являтись протягом ігрового процесу. Тому доречно використати патерн проектування «пул об'єктів» [14], що завчасно створюватиме екземпляри й керуватиме ними, що значно підвищить продуктивність гри, адже одною з основних переваг даного патерну є отримання об'єкта з пула, що відбувається за передбачуваний час, коли створення нових об'єктів може займати чимало часу. Для його реалізації потрібно мати узагальнюючий клас, що матиме в собі потрібні методи, а саме:

- ініціалізація пулу;
- взяття об'єкта з пулу: при потребі у використанні об'єкта, візьміть його із пулу, якщо немає доступного – створіть новий та поверніть, якщо не існують обмежень у кількості об'єктів.
- повернення об'єкта в пул: після закінчення використання об'єкта поверніть його назад у пул, де він буде готовий до наступного використання.

Лістинг коду узагальнюючого класу:

```
public class PoolBase<T>
{
    private readonly Func<T> preloadFunc;
    private readonly Action<T> getAction;
    private readonly Action<T> returnAction;

    private Queue<T> pool = new Queue<T>();
    private List<T> active = new List<T>();

    public PoolBase(Func<T> preloadFunc, Action<T> getAction,
        Action<T> returnAction, int preloadCount)
    {
        this.preloadFunc = preloadFunc;
        this.getAction = getAction;
        this.returnAction = returnAction;
        if (preloadFunc == null)
        {
            return;
        }
    }
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				31
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        for (int i = 0; i < preloadCount; i++)
        {
            Return(preloadFunc());
        }
    }
    virtual public T Get()
    {
        T item = pool.Count > 0 ? pool.Dequeue() : preloadFunc();
        getAction(item);
        active.Add(item);
        return item;
    }
    public void Return(T item)
    {
        returnAction(item);
        pool.Enqueue(item);
        active.Remove(item);
    }
    public void ReturnAll()
    {
        foreach (T item in active)
            Return(item);
    }
}

```

Перейдемо до реалізації пулу саме гемів. Для цього створимо відповідні методи завантаження, де буде створюватись екземпляр гему, метод взяття й повернення його. Методи є статичними й є аргументами у конструкторі класу, що має наступний лістинг коду:

```

public GemPool(Transform pos, Gem prefab, GemSO so, int
preloadCount) :
    base(() => Preload(pos, prefab, so), (gem) => GetAction(gem,
pos), (gem) => ReturnAction(gem, pos), preloadCount)
{ }

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		32

Повернення гему назад у пул здійснюється після його знищення або деактивації. Для цього здійснюється підписання на відповідну подію у гемі.

Так як гемів 5 типів, доречно створити сценарій, що міститиме в собі усі пули кожного типу. Словник зручний тип в якому буду зберігатись пули за ключами, які визначають типи гемів. Так як геми з'являються у випадковому порядку, додамо метод, що повертає випадковий пул. Переглянути усю діаграму класів [15] системи пулу об'єктів можна на рисунку 2.3. Лістинг цього методу наступний:

```
public GemPool GetRandomPool()
{
    int randomIndex = Random.Range(0, keysList.Count);

    GemType randomKey = keysList[randomIndex];
    return gemPools[randomKey];
}
```

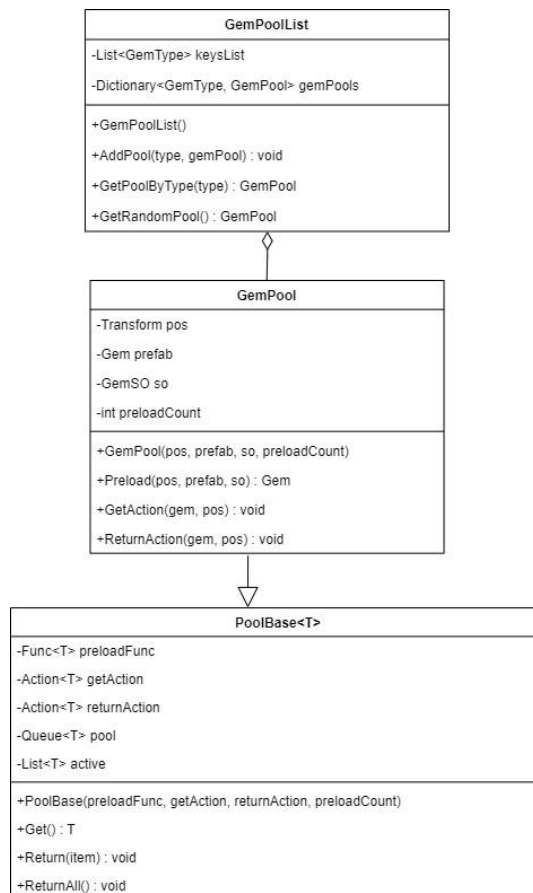


Рисунок 2.3 – Діаграма класів пулу об'єктів

Тепер перейдемо до реалізації сценарію лінії об'єднання. Вона матиме наступні поля:

- тип гему якого здійснюватиметься об'єднання;
- масив усіх об'єднаних клітинок;
- поточна активна клітинка;
- стек шляху, що пройшов бонусний гем – деякі види гемів рухатимуться за лінією об'єднання, для цього їй потрібен стек клітинок, по яким він рухався, що б у разі чого, повернути його на місце;
- черга з активних об'єктів сітки – так як вони активуються після руйнування усієї лінії;
- подія, що повідомлятиме про завершення руйнування лінії.

У конструкторі даного класу здійснимо підпис на глобальну подію активації активних об'єктів. Метод, що оброблятиме дану подію додаватиме до черги об'єкт, якщо його немає у черзі уже.

Перейдемо до методів утворення лінії. Створення нової лінії відбувається надходженням клітинки та створенням нових екземплярів стека, масивів та черги, визначенням якому типу відповідатиме дане об'єднання, клітинка передається до методу додання її у масив. Також викликається метод, що створюватиме шлях для бонусних гемів, якщо клітинка має його. Цей метод також робить елементи усередині клітинки активними, і, якщо лінія має більше однієї клітинки, у попередній клітинці викликається метод, що робить активною стрілку, яка повертається у сторону, в яку було продовжено об'єднання. Це зроблено для уникнення плутанини з тим, як гравцеві повернутись назад.

Метод для стрілки приймає у себе наступну клітинку. Допоміжним методом LookAt визначається у якому напрямку було здійснено об'єднання, після чого відповідно цьому напрямку визначається кут, на який потрібно повернути стрілку. Стрілка стає активною.

Повернемося до утворення лінії. Наступні клітинки, що будуть надходити до лінії будуть перевірятись на те, чи відповідає тип гему типу лінії, чи клітинка

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		34

знаходиться поруч з активною, чи можливо взагалі здійснити об'єднання. На цій перевірці зупинимось детальніше.

Визначаємо напрям, у якому здійснюється об'єднання. Відповідно напрямку йдуть наступні перевірки:

- якщо з'єднання здійснюється по одній із осей, до прикладу управо, перевіряється чи не є у активній клітинці стінка у тому напрямку та у наступній клітинці стінка протилежного напрямку;
- якщо з'єднання здійснюється по діагоналі, у перевірці будуть задіяні 4 клітинки: поточна, сусідня поточної в сторону з'єднання, наступна та клітинка, що знаходиться під поточною. Після чого з'єднання вважається неможливим, якщо:
 - поточна й сусідня мають стінки у напрямку який здійснюється (наприклад, здійснюється з'єднання у лівий нижній кут, якщо ці клітинки матимуть нижні стінки – з'єднання неможливе);
 - наступна й та, що під поточною мають стінки у протилежному напрямку який здійснюється;
 - у поточній утворюється кут у сторону з'єднання, або у наступній кут у протилежну сторону (наприклад, здійснюється з'єднання у лівий нижній кут, якщо поточна має нижню й ліву стінку, або наступна має верхню й праву – утворюються кути, що унеможливають з'єднання);
 - поточна має стінку у напрямку який здійснюється та сусідня їй має об'єкт типу Wall;
 - наступна клітинка має стінку протилежну напрямку який здійснюється та клітина, що під поточною має об'єкт типу Wall;

Якщо перевірка успішна – клітинка додається до масиву. Якщо попередня клітинка має бонусний гем, здійснюється зміна позицій між цією й новою клітинкою, утворюється стек з шляху такого типу гему.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		35

Якщо клітинка уже є у масиві, перевіряється чи вона передостання у ньому, якщо так – здійснюється повернення по лінії: клітинки стають не активними, а стрілки не вказують у їхньому напрямку.

При об'єднанні або поверненні здійснюється глобальний виклик події, що повідомляє про те, що кількість змінилась. Це додано для того, щоб показувати гравцеві візуально їх комбінацію.

Також головною складовою є знищення цієї лінії. У циклі викликається відповідний метод у кожній активній клітинці, а саме DestroyGridObject. Очікуємо завершення асинхронних методів знищення. Якщо у ході цього знищення активується відповідний об'єкт, він додається до черги. Після йде асинхронна перевірка на те, чи не порожня черга активних об'єктів, якщо такий існує – здійснюється його активація, після чого знову викликається цей же метод.

Проте, якщо лінія має менше трьох об'єднань, вона просто деактивується, бонусні геми повертаються на свої місця.

Лістинг даного методу руйнування лінії:

```
public async void DeactivateCells()
{
    if (gridCells.Count >= 3)
    {
        List<Task> tasks = new ();
        foreach (GridCell cell in gridCells)
        {
            tasks.Add(cell.DestroyGridObject());
        }
        await Task.WhenAll(tasks);

        EventManager.instance.OnComboChanged?.Invoke(gridCells.Count);
        await CheckForActiveObject();
        OnLineDestroy?.Invoke();
    }
    else
    {
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				36
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        foreach (GridCell cell in gridCells)
        {
            cell.SetGemActive(false);
            cell.SetGemArrow(false, null);
        }
        RemoveBGRoad();

    }
    activeGC = null;
    BGroad.Clear();
    gridCells.Clear();
    CheckComboDebug();

}

```

Перейдемо до розробки менеджера усієї системи сітки. Він контролюватиме систему об'єднання, саму сітку та пули гемів. Тому початкове налаштування створює нові екземпляри потрібних класів та підписує метод обробки подію завершення руйнування лінії.

Налаштування клітинок здійснюється пошуком у вкладених об'єктів із відповідним типом, потім у циклі здійснюється налаштування клітинок. Після усього у сітку передаються усі клітинки.

Важливо додати поле, що інформуватиме про те, чи зайнятий зараз менеджер. Це запобігає спробі змінити сітку, коли сітка знаходиться в стані змінення. А також поле, що інформуватиме про те, що лінія уже створена.

Реалізуємо методи, що оброблятимуть натискання на клітинки та методи, що утворюватимуть лінію. Для цього нам потрібно метод початку нової лінії, додання нової клітинки та руйнування лінії. Лістинг коду даних методів:

```

public void StartLine(GridCell gridCell)
{
    if (isBusy) return;
    IsLineActive = true;
    GemLine.NewGemMatch3Line(gridCell);
}

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				37
Змн.	Арк.	№ докум.	Підпис	Дата		

```

public void AddActiveGridCell(GridCell gridCell)
{
    if (isBusy) return;
    if (gridCell.IsEmpty()) return;
    GemLine.AddMatch3Gem(gridCell);
}
public void TryToDestroyLine()
{
    if (isBusy) return;
    IsLineActive = false;
    GemLine.DeactivateCells();
}

```

У сценарії клітинки додамо також відповідні методи, які будуть передані у PointManager. Лістинг коду цих методів наступний:

```

public void OnPointerDown()
{
    if(IsHasGem())
        gridManager.StartLine(this);
}
public void OnPointerEnter()
{
    if (gridManager.IsLineActive && IsHasGem())
    {
        gridManager.AddActiveGridCell(this);
    }
}
public void OnPointerUp()
{
    if (gridManager.IsLineActive)
    {
        gridManager.TryToDestroyLine();
    }
}

```

Метод обробки події зруйнування лінії викликатиме систему падіння гемів, здійснюється у три цикли, а саме:

- 1) Падіння вертикально вниз. Починаючи з лівого нижнього кута сітки здійснюється перевірка на наявність об'єкта, що піддається гравітації. Як тільки цей елемент знайдено циклічно він переміщається до низу, перевіряючи чи клітинка порожня й належить межах сітки, а також перевірка на наявність у поточній клітці нижньої стінки, а у нижній - верхньої, до моменту повного падіння. Після чого продовжується пошук наступного елемента. Якщо верхня клітинка даного стовпчика порожня – береться гем з пулу й далі повторює поведінку падіння описану вище. Повторюється цей цикл до тих пір, поки верхня клітинка буде заповнена.
- 2) Падіння діагонально. Йдеться пошук елемента, що піддається гравітації, перевіряючи зліва на право. Цього разу елементи перевіряють додатково можливість впасти вниз ліворуч або праворуч, проте надаючи перевагу падінню вниз. Перевірка падіння схожа на перевірку можливості проведення лінії.
- 3) Падіння діагонально. Йдеться пошук елемента, що піддається гравітації, перевіряючи з права на ліво. Ця додаткова перевірка у іншому напрямку гарантує перевірку усіх можливих ситуацій для падіння елементів, що піддаються гравітації.

Після падіння об'єктів здійснюється перевірка на можливість створити лінію з щонайменше трьох елементів з однаковим типом. Для цього проводиться перевірка, яка перевіряє кожен елемент і елементи навколо нього, якщо серед них є 3 однакових типи здійснюється перевірка за допомогою методу з сценарію об'єднання лінії, яка перевіряє чи ці геми можна об'єднати між собою. Якщо таких пар об'єднань більше двох – об'єднання можливе. Якщо ні – здійснюється перетасування елементів у середині сітки.

Тасування здійснюється шляхом отримання усіх гемів у сітці та відсортуванням за типами. Якщо у кожному типі кількість елементів менше

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		39

трьох у випадковому типі додається 1 гем, а у іншому 1 гем забирається. Надалі здійснюється саме перетасування, що змінює положення гемів між собою до моменту можливого утворення лінії. Лістинг коду тасування елементів:

```
public static List<T> Shuffle<T>(List<T> objects)
{
    for (int t = 0; t < objects.Count; t++)
    {
        T tmp = objects[t];
        int r = Random.Range(t, objects.Count);
        objects[t] = objects[r];
        objects[r] = tmp;
    }

    return objects;
}
```

Після усіх процесів менеджер стає не зайнятим, що надає можливість керувати ігровими елементами в середині сітки, й викликається глобальна подія завершення ходу. Лістинг коду методу падіння гемів:

```
private async Task FallGem()
{
    List<Task> tasks = new();
    for (int x = 0; x < grid.Width; x++)
    {
        tasks.Add(FallCollumn(x, 1));
    }
    await Task.WhenAll(tasks);

    for (int x = 0; x < grid.Width; x++)
    {
        await FallInColumDiagonal(x, 1);
    }

    for (int x = grid.Width - 1; x >= 0; x--)
    {
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		40

```

        await FallInColumDiagonal(x, 1);
    }

    await SpawnBonusGem();
    await CheckForContinue();
    isBusy = false;
    EventManager.instance.OnTurnEnded?.Invoke();
}

```

У грі буде чотири типи бонусних гемів, але для початку реалізуємо сценарій базового класу для них, що наслідуватиме клас Gem. Такі об'єкти взаємодіятимуть з клітинками, тому їм варто зберігати значення клітинки, у якій вони знаходяться – ця інформація отримуватиметься під час руху по сітці перевизначивши метод призначення нової клітинки. А також важливим є додання поля, що інформуватиме про те, що бонусний гем уже активовано, для уникнення безкінечного зациклення, коли бонусні гери активують один одного.

Одними з бонусів буде «горизонталь» та «вертикаль», що будуть здійснювати однакові дії, проте у різних осях, для цього визначивши відповідне поле. Перевизначимо метод руйнування: позначаємо, що об'єкт активовано, далі отримуємо масив клітинок у зазначеному радіусі та викликаємо у них метод руйнування, перевіривши, чи елемент належить типу Gem та якщо це бонусний гем, який ще не активований. Після чого здійснюється базова поведінка знищення. Лістинг коду методу руйнування:

```

    public override async Task Destroy(Action callback, Transform
    target)
    {
        isActivated = true;
        cells = isHorizontal ? cell.GetRow(gemCountToDestroy) :
        cell.GetColumn(gemCountToDestroy);
        DeactFoBG();
        Quaternion rotation = transform.rotation;
        Shake();
        List<Task> tasks = new();
        foreach (GridCell cell in cells)

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				41
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    {
        if (cell.IsEmpty() || ( cell.GridObject is BonusGem bonus
&& bonus.isActivated))
        {
            continue;
        }
        tasks.Add(cell.DestroyGridObject(transform));
    }
    await Task.WhenAll(tasks);
    await base.Destroy(callback += () => { BGReset(rotation); },
target);
}

```

Іншим бонусом є «каструлька», що здійснює сході дії з попереднім бонусом, проте отримуючи клітинки навколо поточної у певному радіусі.

Ще у грі є бонус «ящик», що відрізняється від інших бонусів тим, що він не рухається за лінією об'єднання та не потребує з'єднання у 3 геми для активації, так як він активується будь коли при натисканні на нього. Він перетворює геми навколо себе у геми типу, якому він відповідає (наприклад, перетворює усі геми навколо себе на салат). Для нього потрібне перевизначення методу активації об'єкта, під час якого він зберігатиме інформацію про клітинки й геми у них до заміни, задля того, щоб повернути їх коли бонус стає неактивним.

Перейдемо до реалізації системи активних елементів сітки. Їх особливістю є те, що на них впливає об'єднання гемів поруч з ними. Тому вони повинні підписуватись на відповідні події у сусідніх клітинках. Також поле, що інформуватиме про те, що об'єкт уже активовано.

У грі буде декілька видів таких елементів, одним з яких є гем, що руйнується. Він може слугувати як тимчасова перешкода, або елемент для виконання завдання. Для цього додамо відповідне поле, що інформуватиме про те, чи існує цей елемент після знищення на сітці. Далі просто додаємо обробник події, що повідомляє про знищення гемів у сусідніх клітинках, а саме змінюючи вигляд на наступний у масиві значень, якщо масив завершується і це є елементом

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		42

для завдання, викликається глобальна подія, що повідомляє про це. Лістинг коду даного методу:

```
override public void OnGemsDestroyInNeighboringCells()
{
    if (isReady)
    {
        CountToActivate--;
        isReady = false;
        if (CountToActivate >= 0)
        {
            image.sprite = allState[CountToActivate];
        }
        if (CountToActivate <= 0)
        {
            if (!existAfterDestroy)
            {
                cell.Clear();
                return;
            }
            if(inside!=null)inside.DOAnchorPosY(50,
2).SetEase(Ease.OutElastic).OnComplete(()=>Destroy(inside.gameObject
));
            fon.sprite = allState[0];

EventManager.instance.OnDestroyableDestroy?.Invoke();
            image.gameObject.SetActive(false);
            Unsubscribe();
        }
    }
}
```

Наступним є елемент, що об'єднуватиме усі геми в стороні, у яку направлений, тому при налаштуванні цього об'єкту здійснюється пошук усіх клітинок, що відповідають умовам: об'єкт розташований горизонтально, чи вертикально, та чи здійснюється об'єднування по обидва боки від елемента. При

готовності до активації глобально про це інформується. А сама активація здійснюється у сценарії об'єднання лінії, що було описано раніше.

Діаграма класів елементів сітки зображена на рисунку 2.4.

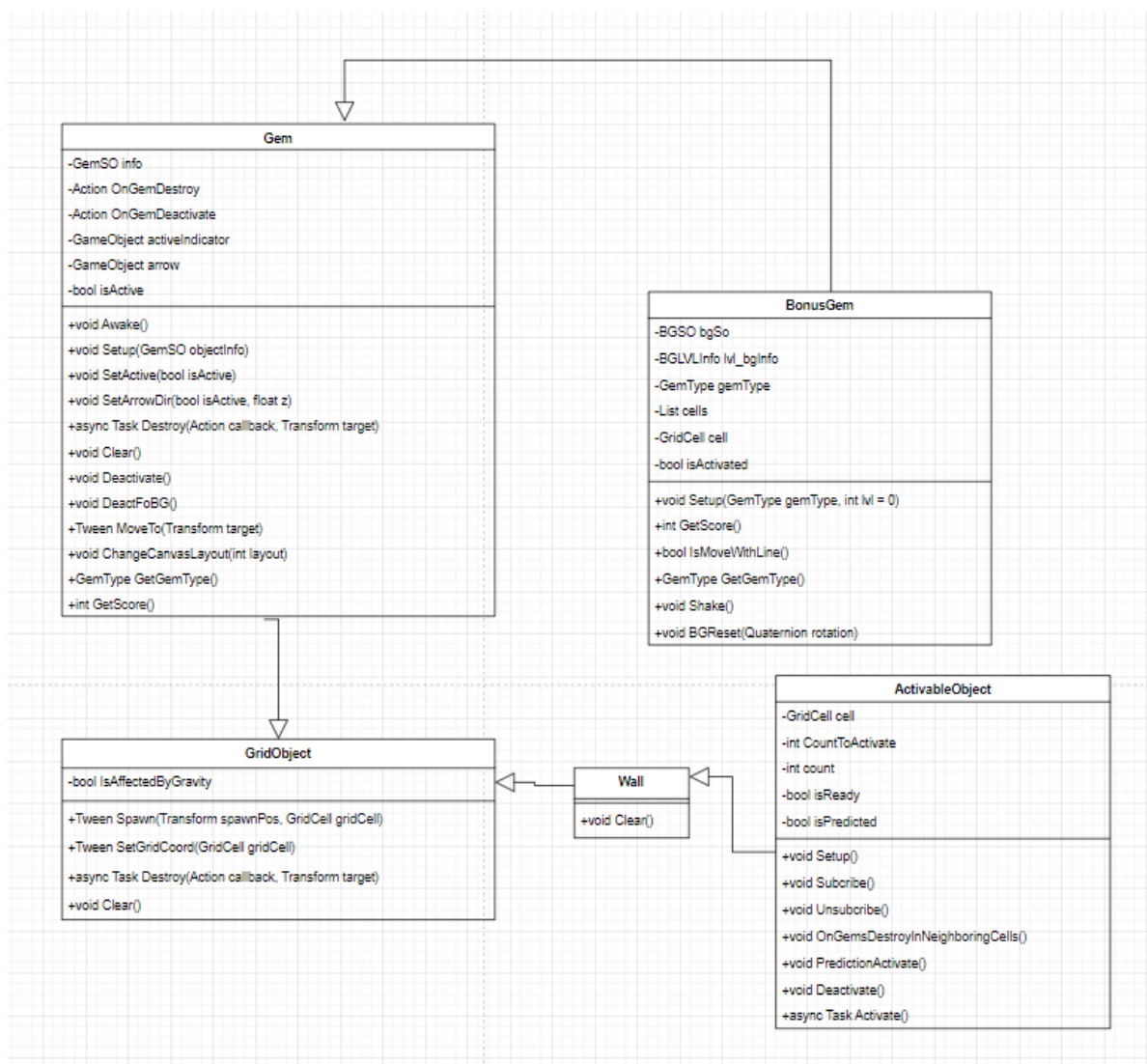


Рисунок 2.4 – Діаграма класів елементів сітки.

Здійснивши усі кроки, система сітки й ігрових елементів у ній готова.

2.2.2. Розробка системи шефів

Шефи відіграють важливу роль у грі, адже вони надають бонуси до лінії об'єднання відповідного типу гему, окрім того за певну кількість об'єднань на полі з'являється бонусні геми. Шефів можна покращувати за ігрову валюту.

Спочатку реалізуємо об'єкт сценарію, що міститиме в собі інформацію про шефа й його рівнів, а саме: ім'я, спрайт, тип гему, за який він відповідає, бонусний гем, яким він володіє та масив інформації про кожен рівень.

Наступним реалізуємо сценарій самого шефа. Метод налаштування буде здійснено підпис на глобальну подію, що інформує про знищення гему та подію, що інформує про завершення ходу, та створення ігрового об'єкту під час якого йде підпис на подію його активації, яка повертатиме бонусний гем до шефу.

Метод, що обробляє подію знищення гему, перевіряє чи гем відповідає типу шефу й далі викликає глобальну подію методу, що повертає очки з бонусом від шефу. Після, якщо бонусний гем не на ігровому полі, йде зарядження його властивості. При повному заряді бонусний гем передається менеджеру сітки, який після падіння гемів розміщує його на ігровому полі. Лістинг коду активації й деактивації властивості шефу:

```
public void ActivateBG()  
{  
    EventManager.instance.OnBonusCharged?.Invoke(bg);  
    bg.ChangeCanvasLayout(2);  
    bg.transform.position = transform.position;  
    bg.gameObject.SetActive(true);  
    bg.OnGemDeactivate += (gem) => DeactivateBG((BonusGem)gem);  
    isActive = true;  
    isReady = false;  
}  
public void DeactivateBG(BonusGem bg)  
{  
    bg.gameObject.SetActive(false);  
    bg.transform.SetParent(poolPos);  
    countToUltimate = 0;  
    isActive = false;  
    bg.isActivated = false;  
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		45

Для того щоб назначити шефа на робоче місце додамо відповідний метод, що отримуватиме шефа й змінювати спрайт на спрайт шефа.

Усі шефи будуть зберігатись у об'єкті сценарію. Додамо можливість отримати масив шефів відповідного типу.

Створимо менеджер шефів, що й буде назначати шефів й налаштовувати після початку гри. Варто додати механіку збереження вибраних шефів, щоб не повторювати однакові дії при проходженні рівнів. Для цього використаємо PlayerPrefs, що й зберігатиме масив типів робочих місць й обраного шефа. Під час початку рівня, при наявності збереження, шефи назначаються до своїх відповідних місць. Лістинг коду методу, що відслідковує початок гри:

```
private void OnStartGameHandler()
{
    chiefChanger.OnChiefChoose -= SetChiefToPlace;
    ChiefPlaceSave data = new()
    {
        data = new()
    };
    foreach (var place in chiefPlaces)
    {
        place.OnPlaceClick -= ShowChiefMenu;
        if (place.chief == null)
        {
            place.Setup();
            continue;
        }
        place.Setup(gemPoolTransform,
        GetBGByType(place.chief.bgType));
        data.data.Add(new ChiefPlaceData() { gemType =
        place.gemType, place = place.playerData });
    }
    string chiefs = JsonUtility.ToJson(data);
    PlayerPrefs.SetString("ChiefPlace", chiefs);
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		46

```

    PlayerPrefs.Save();
}

```

2.2.3. Розробка системи клієнтів

Однією з головних механік гри є годування відвідувачів. Об'єднуючи продукти вони утворюють рівень задоволення клієнтів. Шефи додають бонус до цього рівня. Крім того, для різноманіття ігрового досвіду було вирішено додати вподобання:

- клієнт полюбляє якийсь вид продукту: отримує додатково бонус до рівня задоволення від конкретного типу продукту;
- клієнт недолюблює якийсь вид продукту: отримує меншу кількість задоволення від конкретного типу продукту;
- клієнт не любить якийсь вид продукту: втрачає задоволення, коли об'єднується конкретний тип продукту.

Виділимо основні частини, які потрібні для реалізації цієї механіки: менеджер клієнтів, сценарій клієнта, об'єкт сценарію клієнта, рівень задоволення та вподобання.

Перейдемо до реалізації об'єкту сценарію клієнта. Він міститиме у собі базовий вигляд та масив аксесуарів, що можуть додаватись до нього. Аксесуари містять свій тип, щоб уникнути вибору, до прикладу, двох капелюхів одночасно. Лістинг коду об'єкта сценарію для клієнта виглядає наступним чином:

```

public class CustomerSO : ScriptableObject
{
    public Sprite baseImg;
    public List<Accessory> accessories;
}
public struct Accessory
{
    public AccessoryType type;
    public Sprite sprite;
}
public enum AccessoryType
{
    hat,
    chest
}

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		47

Перейдемо до реалізації сценарію рівня задоволення. Це буде рядок стану, що відображатиме потрібну кількість до повного задоволення та заповнюватиме відповідну шкалу. У сценарії присутня подія, що повідомляє про заповнення шкали. Серед основних методів виділимо метод налаштування, що визначає потрібну кількість задоволення, за присутності наяву кількість задоволення, зміна візуального відображення та метод зміни рівня шкали, що змінює поточний рівень додаючи, або віднімаючи значення, що було отримане, та з наступним оновленням візуального відображення. Лістинг методу налаштування та методу зміни рівня:

```
public void Setup(int maxValue, int currentValue)
{
    this.maxValue = maxValue;
    progressValue = currentValue;
    OnChangeProgressValue();
}
public void AddProgress(int value)
{
    progressValue += value;
    if (progressValue >= maxValue) OnComplited?.Invoke();
    progressValue = Mathf.Clamp(progressValue, 0, maxValue);
    OnChangeProgressValue();
}
```

Заповнення шкали відбувається зміною масштабу розміру відповідного об'єкту наступним чином: визначається який відсоток має поточний рівень відносно потрібного, значення, що отримане й є масштабом об'єкту. Лістинг коду цього методу має наступний вигляд:

```
private void OnChangeProgressValue()
{
    string textStr = $"{maxValue - progressValue}";
    text.text = textStr;
    float curentPercent = (float)progressValue /
(float)maxValue;
    progress.transform.DOScaleY(curentPercent, 0.2f);
}
```

Перейдемо до реалізації механіки задоволення. Для цього нам потрібно менеджер, який буде містити в собі інформацію у типі словника, який зберігатиме кожне вподобання відповідно до якогось типу продукту. У метод налаштування отримується інформація про які типи вподобання має поточний

клієнт, до кожного з низ випадковим чином обирається тип продукту. Створюються екземпляри вподобань і додаються до словнику.

Щодо сценарію самого вподобання, то воно містить у собі метод, який обраховуватиме рівень задоволення відповідно вподобанням до конкретного типу продукту та повертатиме його. Лістинг коду цього методу:

```
public int Bonus(int sat)
{
    int value = type switch
    {
        FastidiousnessType.fine => sat * (100 + bonusPercent) /
100,
        FastidiousnessType.meh => sat * (100 - bonusPercent) /
100,
        FastidiousnessType.bad => -(sat * bonusPercent / 100),
        _ => sat
    };
    return value;
}
```

Перейдемо до реалізації сценарію клієнта з методу налаштування. Цей метод отримує шкалу статусу та сценарій вподобань. Після чого визначається який рівень задоволення потрібний та які вподобання має клієнт, підпишемося на подію заповнення шкали задоволення. Також реалізуємо метод відображення. Так як клієнт матиме свої аксесуари, кожен аксесуар буде створюватись на об'єкті клієнта й зберігатись у масиві.

Додамо метод появи та уходу клієнта. Вони здійснюватимуть однакові анімаційні рухи, проте у різні точки. Також після завершення анімації будуть виконувати різні функції, отож зробимо узагальнений метод для цього, що прийматиме у себе значення чи входить клієнт і які дії буде виконано після завершення анімації. Лістинг коду методу виглядає наступним чином:

```
public void MoveAnim(bool isEnter, Action func)
{
    Tween wobbling = transform.DOLocalMoveY(50,
0.15f).SetEase(Ease.Linear).SetLoops(-1, LoopType.Yoyo);
    transform.DOLocalMoveX(isEnter?0:1000,
2f).SetEase(Ease.Linear).OnComplete(() =>
    {
        wobbling.Kill();
        func();
    });
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		49

Виклик методу входу здійснюється при старті гри або задоволення попереднього клієнту. Виклик методу виходу здійснюється в наслідок реакції на подію заповнення шкали задоволення, на яку було здійснено підпис у налаштуванні. Лістинг коду цих методів:

```
public void MoveIn()
{
    MoveAnim(true, () =>
    {
        statusBar.Setup(necessarySat);
        UIfast.Setup(fastManager.fastidiousnesses);
        statusBar.SetActive(true);
        transform.DOLocalMoveY(0, 0.2f);
        OnCustomerReady?.Invoke(this);
    });
}
public void MoveOut()
{
    statusBar.SetActive(false);
    MoveAnim(false, () => {
        foreach (var img in accessory)
        {
            Destroy(img.gameObject);
        }
        OnExitAnimFinished?.Invoke(this); });
}
```

Перейдемо до реалізації самого менеджера. Для оптимізації затрат ресурсів було вирішено додати сюди систему пулу об'єктів. 2 об'єкти, що будуть чергуватись між собою, проте по-різному налаштовуватись, а саме отримували базовий вигляд клієнту та масив аксесуарів, що на ньому.

Менеджер підписується на глобальну подію, що відслідковуватиме утворений рівень задоволення від одного продукту в сумі з бонусом від шефа. Це зроблено для того, щоб контролювати великі рівні задоволення: при великому об'єднанні може утворитись великий рівень задоволення, що не повністю потрібний конкретному відвідувачу, тому залишок перенесеться до наступного.

Менеджер також підписується на глобальну подію початку рівня, щоб налаштувати й відправити першого клієнта.

Ознайомитись детальніше з лістингом коду менеджера клієнтів можна у додатку А.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		50

2.2.4. Розробка системи рівнів та їх побудови

Розробку системи рівнів варто почати з об'єкту сценарію, що буде включати у себе усю інформацію щодо рівня. Потрібна інформація для рівня:

- місяць;
- день місяця;
- масив усіх налаштованих клітинок, що включатимуть у себе масив стінок й їх напрямку, тип підлоги, префаб об'єкта й тип гему, якщо об'єкт є типом Gem;
- інформація про ходи у рівні, що включає у себе максимум ходів на рівень, максимальна кількість ходів для отримання трьох, двох й однієї зірок;
- масив завдань на рівень, а саме тип та кількість;
- масив видів клієнтів, що вміщують в собі також інформацію про їх шанс появи, мінімальне й максимальне значення потрібного рівня задоволення та бонус до отримання грошей за обслуговування;
- кількість ігрової валюти, що буде отримано за обслуговування клієнта;
- кількість ігрової валюти, що буде одноразово отримано за проходження рівня;
- шеф, що розблокується після проходження рівня, якщо такий потрібен;
- підказки, що включають у себе відео та текст підказки.

Для зручності створення ігрового поля створимо нову сцену й додамо елементи цього поля. Для роботи з цим нам потрібні наступні сценарії: менеджер редактора сітки, відповідна клітинка, палітра об'єктів.

Почнемо з сценарії клітинки сітки редактора. Цей елемент міститиме в собі менеджер підлоги й стінок, інформацію про префаб об'єкту й тип гему, якщо об'єкт являється типом Gem, подія, що інформуватиме про натискання на клітинку.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		51

Серед методів виділимо методи налаштування, які отримують інформацію про клітинку й створюють відповідні елементи у ній. Методи для контролю підлоги й стінок, а саме додання й прибирання відповідних елементів. Метод присвоєння об'єкту й, якщо цей об'єкт типу Gem, тип гему.

Перейдемо до реалізації менеджера редактора сітки. Схоже налаштування зі методом налаштування менеджера сітки, де отримуються ігрові об'єкти відповідного типу та передаються у створений екземпляр сценарію сітки, у клітинки передається інформація для їх налаштування.

Для присвоєння об'єктів клітинки потрібно створити палітру відповідну. Цей сценарій відображатиме список усіх наявних ігрових елементів, а також додаткові елементи такі як, типи стінок й підлоги. У елементів списку є подія, що інформує про те який саме об'єкт був обраний.

Менеджер підписується на події палітри й клітинки, щоб зберігати інформацію про активний об'єкт для присвоєння та активну клітинку для редагування. Додано відповідні методи редагування клітинки, прив'язані до натискання відповідних клавіш.

Головною частиною є збереження змін у об'єкт сценарію про рівень. Варто підмітити, що потрібно використовувати перевірку чи являється це юніті редактором, адже при спробі зробити збірку гри можуть виникнути проблеми: ці методи не доступні за межами юніті редактора. Лістинг коду методу збереження сітки рівня:

```
#if UNITY_EDITOR
    public void Save()
    {
        lvl.cells = new CellLvlInfo[49];
        for (int x = 0; x < 49; x++)
        {
            lvl.cells[x] = visualCell[x].GetCellInfo();
        };
        EditorUtility.SetDirty(lvl);
        AssetDatabase.SaveAssets();
    }
#endif
```

З лістингом коду менеджера редактора сітки можна ознайомитись у додатку Б.

Розширимо менеджер сітки, додаючи метод налаштування, що отримуватиме масив інформації про клітинки й передаватиме його у відповідні клітинки ігрового поля. Також додамо відповідний метод до клітинки сітки, що прийматиме у себе інформацію й додаватиме об'єкти гри, стінки, або підлоги, якщо існує така інформація.

Реалізуємо систему завдань. Почнемо з створення сценарію завдання. Метод налаштування отримує тип завдання, кількість, тип гему, якщо завданням є об'єднати якусь кількість гемів. Після здійснюється підпис на відповідні глобальні події. Лістинг коду методу підписання на події:

```
void Subscribe()
{
    switch (type)
    {
        case GoalType.gem:
            EventManager.instance.OnGemDestroy += (Gem gem) =>
            {
                if (gem.GetGemType() == gemType) ChangeState();
            };
            break;
        case GoalType.feed:
            EventManager.instance.OnCustomerSatisfied += () => {
ChangeState(); };
            break;
        case GoalType.clean:
            EventManager.instance.OnFloorCleaned += () => {
ChangeState(); };
            break;
        case GoalType.destroy:
            EventManager.instance.OnDestroyableDestroy += () =>
{ ChangeState(); };
            break;
    }
}
```

Метод зміни стану обробляє ці події та зменшує кількість, що лишилась до повного завершення завдання. При його завершенні викликається відповідна подія, що інформує про це.

Менеджер завдань приймає у себе масив завдань на рівень та створює екземпляри сценаріїв завдань й підписується на подію їх завершення. Якщо кількість завершених завдань рівна кількості завдань викликається глобальна подія, що інформує про це.

Реалізуємо сценарій відслідковування ходів гравця. Для цього передаватимемо у метод налаштування відповідні дані. Також у цьому методі здійснюється підпис на глобальну подію завершення ходу, після чого кількість дій зменшується. Також додамо метод, що повертатиме кількість отриманих зірок за кількість потрачених на рівень ходів. Також додамо подію, що інформуватиме про завершення ходів.

Створимо сценарій гри, що міститиме у собі усі менеджери, що потрібні для гри. При завантаженні сцени, передаватиметься рівень й розділяє потрібну інформацію серед менеджерів. Після чого йде підпис на закінчення ходів у менеджері ходів та підпис на завершення усіх завдань.

При успішному завершенні рівня, формується дані про пройдений рівень, а саме:

- рівень, що був пройдений;
- кількість отриманих зірок за рівень;
- отримана ігрова валюта за рівень, якщо ще не було отримано за це кошти, та за обслуговування клієнтів;
- позначка, що валюта була отримана за проходження рівня.

На початку рівня також здійснюється налаштування підказки й підпис на подію її завершення. Передається відповідна інформація у вигляді відеоматеріалу та супроводжуючого тексту. Якщо підказки на рівень не передбачено, або підказка уже була показана, або гравець підтвердив, що ознайомився з інформацією викликається метод завершення підказки, на який сценарій гри реагує початком рівня.

Ознайомитись з лістингом коду сценарію гри можна ознайомитись у додатку В.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		54

2.2.5. Розробка системи даних гравця та їх збереження

Серед ігрових ресурсів є ігрова валюта, зірки та енергія.

Ігрова валюта отримуватиметься за проходження рівня та за обслуговування клієнтів. Вона використовується для покращення шефів.

Зірки отримуватимуться за проходження рівня відповідно до потрачених ходів для цього. Вони використовуються для покращення рівня елементу інтер'єру.

Енергія отримується через 15 хвилин, якщо не досягнуто її максимум. Вона використовується за кожную спробу пройти рівень, вартість кожного дві енергії.

Для ігрової валюти й зірок реалізуємо узагальнюючий клас Resource. Він включатиме у себе поле, що зберігатиме інформацію про наявну кількість ресурсу, два методи для додання й його витрати, а також події, що інформуватиме, що значення змінено.

Перед наступним налаштуванням варто створити базовий клас для об'єктів сценарію, що будуть зберігати й завантажувати інформацію. У ньому присутні наступні методи:

- збереження;
- завантаження;
- налаштування;
- отримання інформації після проходження рівня;
- очищення.

Лістинг коду базового класу:

```
using UnityEngine;

public class ISaveLoadSO : ScriptableObject
{
    virtual public void Save() { }
    virtual public void Load() { }
    virtual public void Setup() { }
    virtual public void SaveGameStat(GameStat stat)
    {
        PlayerPrefs.Save();
    }
    virtual public void Clear() { }
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		55

Перейдемо до реалізації об'єкту сценарію, що зберігатиме в собі інформацію ресурсів гравця. Зберігання у грі здійснюватиметься шляхом перетворення екземпляру класу у формат JSON та зберігання його за ключем у PlayerPrefs. Отже, об'єкт матиме поля зірок й ігрової валюти, що є екземплярами класу. При налаштуванні здійснюється підпис на подію зміни у ресурсах, метод який це оброблятиме є методом збереження, тому перейдемо до його реалізації.

Використовуючи JsonUtility перетворюємо сереалізуємий клас, що зберігає лише значення зірок та ігрової валюти, у JSON, далі передаємо це за ключем у PlayerPrefs та зберігаємо.

Метод завантаження отримуватиме рядок за ключем, далі, за допомогою утиліти JSON, він перетворюється у клас, що, якщо отримана інформація не порожня, надає інформацію про стан зірок й ігрової валюти гравця. Якщо інформація порожня, створюється нові екземпляри класів. Лістинг коду завантаження:

```

override public void Load()
{
    string strData = PlayerPrefs.GetString("PlayerWallet");
    PlayerWalletData data =
    JsonUtility.FromJson<PlayerWalletData>(strData);
    if (data == null)
    {
        Stars = new Resource();
        Money = new Resource();
        return;
    }
    Stars = data.Stars ?? new Resource();
    Money = data.Money ?? new Resource();
}

```

Метод очищення видаляє усі збережені дані за ключем.

Метод збереження даних з рівня прийматиме у себе інформацію про його проходження та збільшуватиме значення зірок й ігрової валюти відповідно до неї. Після чого йде виклик методу збереження.

Перейдемо до реалізації поведінки енергії. Це буде об'єкт сценарію, що наслідуватиме поведінку базового класу збереження та завантаження. Він матиме наступні поля:

- максимальна енергія;

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		56

- час відновлення однієї одиниці;
- час, що лишився до відновлення однієї одиниці;
- поточна енергія;
- дата й час останнього збереження;
- подія, що інформує про завершення налаштування;
- подія, що інформує про зміну енергії.

Окрім перевизначених методів збереження, завантаження тощо, потрібні методи для додання, витрати та збільшення максимального значення енергії. Кожен з цих методів викликатиме відповідну подію.

Дані, що зберігатимуться у серіалізованому класі: максимальна енергія, поточна енергія, час збереження та час, що лишився до відновлення однієї одиниці енергії.

Метод збереження отримуватиме поточну дату, перетворивши її у рядок, та створюватиме екземпляр класу для збереження. Далі зберігання відбувається шляхом перетворенням у формат JSON та збереженням за ключем у PlayerPrefs.

Метод завантаження отримуватиме значення за ключем.

Варто зупинитись на методі налаштування. Якщо отримана інформація порожня – відповідні значення полів зазначаються базовою інформацією. Проте, якщо ні, спочатку зазначаються відповідні поля значеннями з отриманих даних, далі рядок інформації про збереження перетворюється у формат дати. Знаходиться різниця у секундах між поточною датою й датою останнього збереження. Якщо різниця більша за час, що лишився до відновлення однієї одиниці – додається енергія. Якщо лишається ще час, це значення ділиться на час відновлення однієї одиниці енергії, ціле число додається до усієї енергії. Якщо є залишок від ділення, а енергія не досягла максимуму, то час, що лишився до відновлення однієї одиниці є різницею між усім потрібним для цього часу й остатком від ділення. Після цього викликається подія про завершення налаштування. Лістинг методу налаштування:

```
public override void Setup()
{
    if (loadedData != null)
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		57

```

{
    maxEnergy = loadedData.maxEnergy;
    energy = loadedData.energy;
    lastSavedTime = loadedData.lastSavedTime;
    DateTime savedTime = DateTime.Parse(lastSavedTime);
    timeLeftToRecharge = loadedData.timeLeftToRecharge;
    int timeSinceLastSave = (int)(DateTime.Now -
savedTime).TotalSeconds;
    if (timeSinceLastSave >= timeLeftToRecharge)
    {
        timeSinceLastSave -= timeLeftToRecharge;
        timeLeftToRecharge = energyRecoveryTime;
        energy = Mathf.Clamp(energy + 1, 0, maxEnergy);
        if (timeSinceLastSave > 0)
        {
            int energyGained = (timeSinceLastSave /
energyRecoveryTime);
            timeLeftToRecharge = (timeSinceLastSave %
energyRecoveryTime);
            energy = Mathf.Clamp(energy + energyGained, 0,
maxEnergy);
        }
    }
    else timeLeftToRecharge -= timeSinceLastSave;
}
else
{
    energy = 15;
    maxEnergy = 15;
    timeLeftToRecharge = energyRecoveryTime;
}

OnSetup?.Invoke();
}

```

Метод очищення видаляє усі збережені дані за ключем.

Реалізуємо менеджер енергії. Він міститиме у собі поле, що зберігатиме об'єкт сценарію та виконуватиме відновлення енергії за час, що зазначений у об'єкті, використовуючи щосекундного виклику методу оновлення за допомогою `InvokeRepeating`. Також додано методи витрачання енергії. Ознайомитись з лістингом кодом менеджера можна у додатку Г.

Реалізуємо збереження розблокованих гравцем шефів. Це також буде об'єкт сценарію, що наслідуватиме поведінку базового класу збереження та завантаження. Він одночасно зберігатиме масив усіх доступних шефів у грі та

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		58

масив розблокованих шефів. Розблоковані шефи міститимуть додаткову інформацію, а саме їх рівень й позначку про те, чи він розблокований.

Методи збереження й завантаження мають подібну поведінку, що й у вище описаних об'єктах сценарію.

Метод налаштування перевіряє, чи завантажена інформація є порожньою, якщо так, створюється новий масив, куди передаються усі шефи з позначкою «не розблоковано» й нульовим рівнем.

Метод збереження після проходження рівня перевірятиме на наявність у рівня доступного шефа, і якщо такий є, передає його у метод розблокування. Перейдемо до його реалізації.

Метод розблокування перевірятиме, чи існує такий шеф у масиві і чи він ще не розблокований, якщо так – позначається, що він розблокований. Лістинг коду методу:

```
public void UnlockChief(ChiefSO chief)
{
    ChiefPlayerData chiefPlayer = playerChief.Find(x => x.chief
== chief);
    if (chiefPlayer == null) return;
    if (chiefPlayer.unlocked) return;
    chiefPlayer.unlocked = true;
    chiefPlayer.lvl = 0;
    Save();
}
```

Метод оновлення даних про шефа шукає відповідного шефа у масиві і, якщо він розблокований і існує, збільшує його рівень.

Додано також методи для зручності, а саме: отримання усіх та лише розблокованих шефів.

Ознайомитись з лістингом коду об'єкту сценарію можна у додатку Д.

Перейдемо до реалізації збереження рівнів. Це також буде об'єкт сценарію, що наслідуватиме поведінку базового класу збереження та завантаження. Міститиме у собі масив усіх рівнів та інформацію про їх проходження гравцем. Інформація ця є наступною:

- рівень;
- отримано зірок;

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		59

- чи розблокований він;
- чи він є поточним;
- чи отримано кошти за його проходження;
- чи показано підказку.

Методи збереження й завантаження мають схожі дії, що й описані вище об'єкти сценаріїв.

Метод налаштування, якщо завантажена інформація порожня, створює новий масив інформації гравця та робить перший рівень розблокованим і поточним, якщо ні – усі значення передаються відповідно до рівня, який зазначений у інформації. Лістинг коду методу налаштування:

```

override public void Setup()
{
    if (playerLvlData.Count == 0)
    {
        for (int i = 0; i < lvlSO.Length; i++)
        {
            playerLvlData.Add(new() { lvl = lvlSO[i], stars = 0,
unlocked = i == 0, curent = i == 0 });
        }
    }
    if (loadedData == null) return;
    foreach (var lvl in loadedData)
    {
        if (playerLvlData.Find(x=>x.lvl == lvl.lvl) != null)
        {
            LvlPlayerData data = playerLvlData.Find(x => x.lvl
== lvl.lvl);
            data.lvl = lvl.lvl;
            data.unlocked = lvl.unlocked;
            data.stars = lvl.stars;
            data.curent = lvl.curent;
            data.moneyReceived = lvl.moneyReceived;
            data.tipsShowed = lvl.tipsShowed;
        }
    }
}

```

Метод збереження інформації після проходження рівня оновлює дані рівня, що був пройдений, після чого йде збереження. Лістинг коду методу оновлення даних:

```

public void UpdateLvlData(LvlPlayerData lvl)
{
    int dataID = playerLvlData.FindIndex(x => x.lvl == lvl.lvl);

```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		60

```

playerLvlData[dataID].stars = lvl.stars;
playerLvlData[dataID].moneyReceived = true;
playerLvlData[dataID].tipsShowed = true;
if (dataID + 1 == playerLvlData.Count) return;
if (playerLvlData[dataID].curent)
{
    playerLvlData[dataID].curent = false;
    playerLvlData[dataID + 1].curent = true;
}
playerLvlData[dataID + 1].unlocked = true;

Save();
}

```

Перейдемо до реалізації системи інтер'єру та його бонусів. Одним з складовим його є об'єкт сценарію елементу інтер'єру, що вміщує у себе інформацію про кожен рівень, а саме:

- вартість;
- бонус;
- варіації елементу на цьому рівні.

Іншим складовим є об'єкт сценарію бонусів, що буде використовуватись там у місцях, де потрібно. Він містить в собі інформацію про бонус до максимальної кількості енергії, до задоволення клієнтів та до отриманих коштів за гру. Додамо його також до сценаріїв гри та клієнтів відповідно.

Також об'єкт сценарію, що виконуватиме збереження й завантаження інформації. Він має у собі такі елементи як об'єкт сценарію бонусів та енергії, масив усіх елементів інтер'єру, даних гравця й подія, що інформує про зміни у даних. Дані гравця – масив інформації кожного типу елементу інтер'єру, що зберігає його рівень та вибрану варіацію.

Метод налаштування при відсутності збереженої інформації створює новий масив даних гравця, а бонуси повертаються у базове положення. При наявності збережень – бонуси змінюються відповідно рівнів гравця.

Метод покращення рівня приймає у себе тип елементу інтер'єру, знаходить відповідний елемент та підвищує його рівень на 1. Далі відповідно типу здійснюється наступне:

- якщо це стіл – надається бонус до задоволення клієнтам;

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		61

- якщо це стільчик – надається бонус до заробітку від обслуговування клієнтів;
- якщо це світильник – збільшується максимальна кількість енергії, шляхом виклику відповідного методу у об’єкті сценарію енергії.

Метод зміни варіанту елемента змінює відповідне поле у даних гравця по типу інтер’єру.

Після кожної зміни у інтер’єрі здійснюється збереження та виклик події, що інформує про це.

Сценарій елемента інтер’єру здійснює підпис на подію зміни у даних та змінює візуальну частину відповідно отриманим даним.

Контролюватиме усе це менеджер завантаження й збереження, що міститиме у собі масив усіх об’єктів, що наслідують базовий клас. При активації здійснюється завантаження ігрових даних та подальше їх налаштування, за наявності результатів рівня, викликається відповідний метод. При деактивації здійснюється збереження. Лістинг методу завантаження:

```
private void Start()
{
    foreach (var s in saveData)
    {
        s.Load();
        s.Setup();
    }
    if (Game.gameStat != null)
    {
        foreach (var s in saveData)
        {
            s.SaveGameStat(Game.gameStat);
        }
        Game.gameStat = null;
    }
    isFirstStart = false;
}
```

Лістинг методу збереження:

```
public void Save()
{
    foreach (var s in saveData)
    {
        s.Save();
    }
}
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		62

2.2.6. Інтеграція реклами у гру

Для проекту буде використано SDK Unity ads. Для цього спочатку потрібно зареєструвати проєкт на сервісі юніті, після чого завантажити пакет Advertisement Legacy [16].

Сервіс надає ідентифікатори відповідно до платформи й типу реклами, яка буде використана. У грі буде використано два типи реклами, а саме: інтерстиціальна та реклама з винагородою. Так як гра розроблюється для платформи андроїд звертаємо увагу на відповідні ідентифікатори.

Перейдемо до реалізації класу поведінки реклами. Метод налаштування вказуватиме ідентифікатор реклами. Додамо метод завантаження реклами, для цього використаємо відповідний метод у підключеній бібліотеці. Проте для правильної роботи методу потрібен відповідний слухач, тому підключимо відповідний інтерфейс, й передамо цей клас як слухач.

Метод показу реклами прийматиме дві події, що спрацьовуватимуть під час успішного й ні перегляду реклами. Після показу реклами варто завантажувати наступну, задля уникання затримки при показі реклами. Проте й для показу реклами потрібен відповідний слухач, що також буде додано підключенням відповідного інтерфейсу бібліотеки.

Із всіх обробників події реалізуємо OnUnityAdsShowComplete, що буде спрацьовувати під час завершення перегляду реклами. Якщо ідентифікатор відповідає ідентифікатору реклами й реклама була успішно переглянута, викликаємо метод успішного перегляду. Якщо реклама була пропущена, або виникла якась інша помилка – метод не успішного перегляду. Лістинг коду методу обробнику події перегляду:

```
public void OnUnityAdsShowComplete(string placementId,
UnityAdsShowCompletionState showCompletionState)
{
    if (placementId == adUnitId && showCompletionState ==
UnityAdsShowCompletionState.COMPLETED)
    {
        successAction?.Invoke();
    }
    if (placementId == adUnitId
```

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		63

```

        && (showCompletionState ==
UnityAdsShowCompletionState.SKIPPED
        || showCompletionState ==
UnityAdsShowCompletionState.UNKNOWN))
    {
        failureAction?.Invoke();
    }
}

```

Реалізуємо менеджер реклами. Використаємо патерн одинак для цього та не позначимо цей елемент який не руйнується на сцені. Для налаштування роботи реклами наведемо відповідні поля: ідентифікатор гри для платформи андроїд, ідентифікатори відповідних типів реклами, позначка про те, що це тестування (його варто використовувати, щоб проект не був позначений як шахрайство). У методі Awake реалізуємо патерн одинак та ініціалізуємо рекламу, перевіряючи, чи вона до цього не була ініціалізована та чи реклама підтримується на пристрої. Після чого створимо екземпляри класів для двох типів реклами, передавши відповідні ідентифікатори, й викликавши у них методи завантаження реклами. Лістинг коду методу:

```

private void Awake()
{
    if(Instance != null && Instance != this)
    {
        Destroy(gameObject);
        return;
    }
    Instance = this;
    DontDestroyOnLoad(gameObject);

    if (!Advertisement.isInitialized &&
Advertisement.isSupported)
    {
        Advertisement.Initialize(androidGameID, isTesting,
this);
    }
    rewarded = new AdvertisementBase();
    rewarded.Setup(rewardedID);
    rewarded.LoadAd();

    interstitial = new AdvertisementBase();
    interstitial.Setup(interstitialID);
    interstitial.LoadAd();
}

```

Інтерстиціальна реклама у грі буде показуватись перед кожною третьою спробою пройти рівень. Лістинг виклику показу реклами:

```
public void StartLvl()
{
    LvlSelector.PlayedCount++;
    if (LvlSelector.PlayedCount % 3 == 0)
    {
        AdManager.Instance.interstitial.ShowAd(() =>
        {
            LvlSelector.LvL = lvlData;
            energy.SpendEnergy(2);
            SceneManager.LoadScene(1);
        }, () => { });
        return;
    }
    LvlSelector.LvL = lvlData;
    energy.SpendEnergy(2);
    SceneManager.LoadScene(1);
}
```

Отримати додаткові два ходи під час невдалого проходження рівня, додаткову енергію для можливості пройти рівень, додаткові 150% ігрової валюти заробленої за рівень можна за перегляд реклами з винагородою. Лістинг коду методу отримання додаткової енергії:

```
public void GetExtraTurns()
{
    AdManager.Instance.rewarded.ShowAd(() =>
    {
        turnManager.curentTurn -= 2;
        goalManager.UpdateTurns(turnManager);
        menuFailure.gameObject.SetActive(false);
    },
    () =>
    { });
}
```

Інші винагороди підключено схожим чином.

2.3. Створення графічних елементів

Графічні елементи будуть створено власноруч для збереження одного стилю за допомогою SAI Tools. Почнемо з ігрової сітки. Для цього нам спочатку потрібно зобразити кожен вид продукту (рис.2.5).

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		65



Рисунок 2.5 – П'ять видів продуктів

Також потрібно додати елементи клітинки сітки, як сама клітинка, її бортик та стрілка для вказівки руху лінії (рис. 2.6).



Рисунок 2.6 – Елементи клітинки

У грі головними допоміжниками є шефи, кожного типу по 3 шефи, а також табличка, що інформуватиме за тип гему у відповідному місці (рис. 2.7).



Рисунок 2.7 – Декілька видів шефів та табличка

Кожен шеф має свою властивість (бонусний гем), яких є чотири види: горизонталь, вертикаль, каструлька й ящик (рис. 2.8).

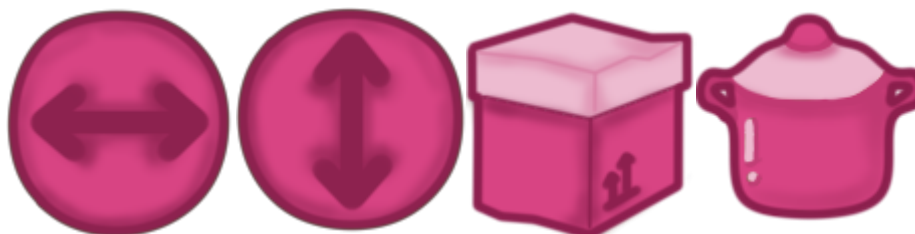


Рисунок 2.8 – Бонусні гему для напоїв

Так як гра про кафе, у ньому повинні бути відвідувачі. Для цього виберемо найкращі варіанти, що підійдуть до стилістики гри, а для того, щоб додати різноманітності, додамо аксесуари для них (рис. 2.9).

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		66



Рисунок 2.9 – Відвідувач і його аксесуари

Зобразимо місце для шефів (рис. 2.10) та фон ігрового поля (рис. 2.11).

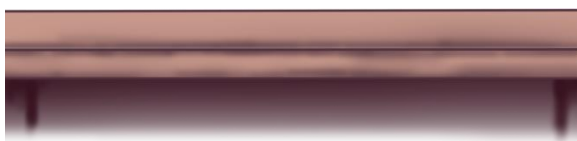


Рисунок 2.10 – Місце для шефів.

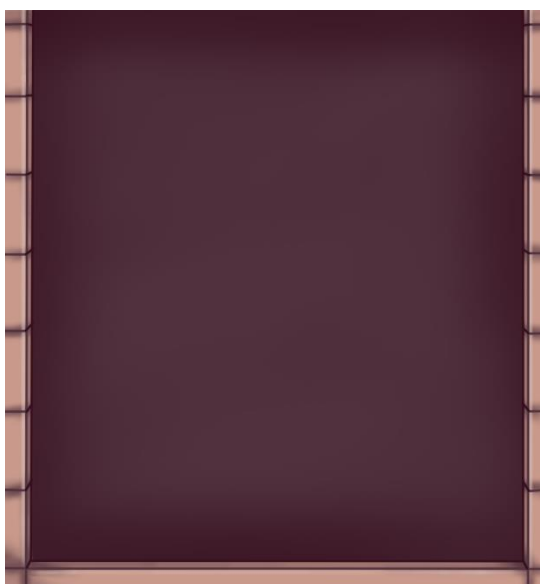


Рисунок 2.11 – Фон ігрового поля

Однією з ігрових механік є вподобання клієнтів щодо продуктів, тому для зрозумілості додамо відповідні значки (рис. 2.12).



Рисунок 2.12 – Емоції відвідувачів

Для ігрового різнобарв'я у грі додані наступні елементи:

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		67

- стінка клітки (рис. 2.13);
- елемент, що руйнується й зникає (рис. 2.14);
- елемент, що руйнується але залишається (використовується для завдань) (рис. 2.15);
- елементи, що активуються й об'єднують лінію у напрямку їх напрямлення (рис. 2.16);



Рисунок 2.13 – Стінка для клітинки



Рисунок 2.14 – Елемент, що руйнується й зникає



Рисунок 2.15 – Елемент, що руйнується але залишається

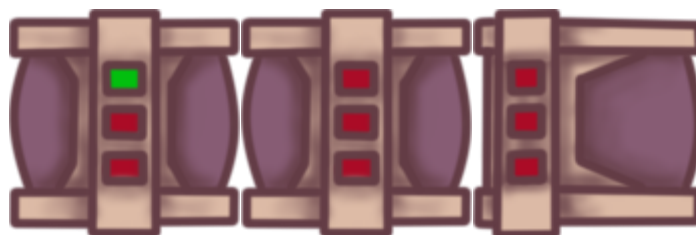


Рисунок 2.16 – Елементи, що активуються й об'єднують лінію у напрямку їх напрямлення

Для зрозумілості завдань на рівень (рис. 2.17) та обмеження у кроках на рівень (рис. 2.18) додамо відповідні значки.



Рисунок 2.17 – Значки завдання на рівень



Рисунок 2.18 – Обмеження у кроках на рівень

У грі також присутня можливість змінювати й покращувати інтер'єр. Рівнів покращення для гри три. Тому для цього потрібні по три на кожен рівень наступних елементів:

- стіл (рис. 2.19);
- стільчик (рис. 2.20);
- світильник (рис 2.21).

Кожен з рівнів відповідає за свій стиль. До прикладу, перший рівень світильника це свічка, другий – керосинові лампи, третій – електричний світильник.



Рисунок 2.19 – Різні варіанти столу другого рівня



Рисунок 2.20 – Один варіант кожного рівня стільця



Рисунок 2.21 – Один варіант кожного рівня світильника

Для сторінки інтер'єру окрім меблів, потрібен ще фон у вигляді кафе. Зобразимо це місце біля стінки (рис. 2.22).

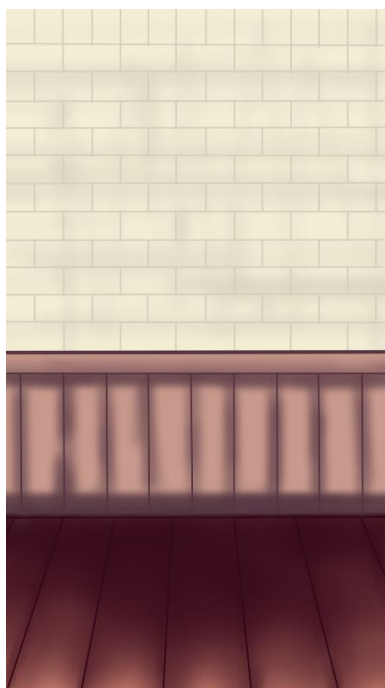


Рисунок 2.22 – Місце у кафе

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				70
Змн.	Арк.	№ докум.	Підпис	Дата		

Наступним потрібно зобразити іконки для кнопок меню (рис. 2.23), а саме:

- перехід у меню шефів;
- перехід у список рівнів;
- перехід на сторінку інтер'єру;
- реклама;
- меню.



Рисунок 2.23 – Іконки для кнопок меню

Наступним потрібно зобразити іконки для ресурсів гравця (рис. 2.24).



Рисунок 2.24 – Іконки ресурсів

Зобразимо список шефів у вигляді записної книжки (рис. 2.25). Самі шефи будь відображатись у рамці (рис. 2.26).



Рисунок 2.25 – Книга для шефів

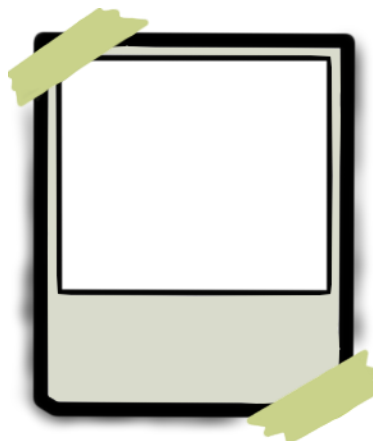


Рисунок 2.26 – Книга для шефів

А список рівнів буде зображено у вигляді календаря (рис. 2.27).

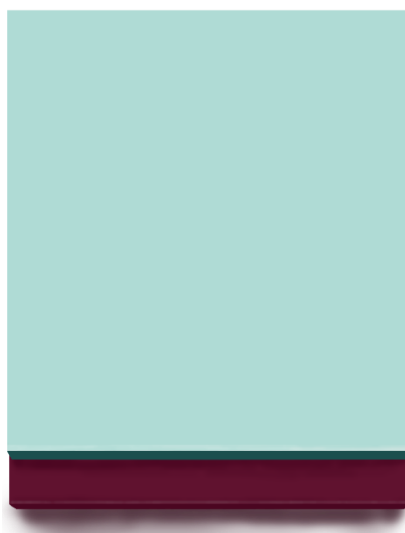


Рисунок 2.27 – Календар рівнів

2.4. Створення користувацького інтерфейсу

Створення користувацького інтерфейсу розпочнемо з навігаційного меню, для переходу між сторінками шефів, рівнів та інтер'єру. Для цього реалізуємо сценарій навігаційного меню, що зберігатиме в собі кнопку та сторінку, до якої вона прив'язана. При активації однієї із кнопок, здійснюється перехід на відповідну сторінку, інші кнопки й вікна стають неактивними (рис. 2.28).



Рисунок 2.28 – Навігаційне меню (активна сторінка рівнів)

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		72

При завантаженні головної сцени для зручності викликається подія активації сторінки рівнів.

Реалізуємо відображення панелі ресурсів гравця (рис. 2.29). Для створимо відповідний сценарій відображення, що буде підписаний на події зміни значень ресурсів.



Рисунок 2.29 – Інформаційна панель ресурсів гравця

Розглянемо детальніше метод виведення інформації щодо наявної енергії у гравця. Таймер обов'язковий для того, щоб гравець знав скільки йому ще чекати. Додано кнопку «+», що викликає меню пропозиції (рис. 2.30).

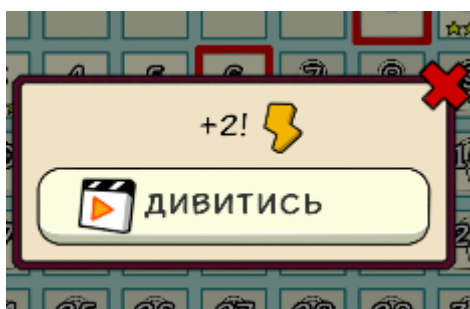


Рисунок 2.30 – Меню пропозиції

Відображення енергії відслідковує події зміни у об'єкті сценарію енергії. Лістинг коду відображення інформації щодо енергії:

```
private void ChangeTime()
{
    if (energyManager.GetEnergy() >=
energyManager.energySO.maxEnergy)
    {
        timer.gameObject.SetActive(false);
        adBtn.gameObject.SetActive(false);
        return;
    }
    if (!timer.gameObject.activeSelf)
    {
        timer.gameObject.SetActive(true);
        adBtn.gameObject.SetActive(true);
    }
    int timeLeft = energyManager.energySO.timeLeftToRecharge;
    int minutes = Mathf.FloorToInt(timeLeft / 60);
    int seconds = Mathf.FloorToInt(timeLeft % 60);
}
```

```
timer.text = "${minutes:00}:{seconds:00}";
}
```

Перейдемо до відображення рівнів (рис. 2.31). Так як відображення здійснюється у вигляді календаря, то потрібні наступні елементи: картинка місяця, назва місяця, перелік днів, клітинки днів та можливість переключитись між місяцями.

Перейдемо до реалізації клітинки днів. День може не належати до поточного місяця, тому відображатись повинен як неактивний елемент. Інші дні пронумеровані у відповідному порядку. Не розблоковані дні відображаються з зображенням, що вказує на це. Розблоковані й пройдені мають кількість зірок, які вони отримали за проходження рівня, а поточний не пройдений рівень позначається червоним контуром, як ніби це поточний день.



Рисунок 2.31 – Відображення рівнів

Для цього додано інформацію про місяці, що мають назву місяця, кількість днів у ньому, початковий день та картинка місяця. Лістинг коду відображення інформації про місяць:

```
private void ChangeMonthInfo()
{
    curMonth = monthTypes[curMonthID];
```

```

List<LvlPlayerData> lvlPlayerDatas =
db.GetLvlsByMonth(curMonth);
MonthInfo curInfo = monthInfo.Find(x => x.month ==
curMonth);
monthImg.sprite = curInfo.sprite;
monthTxt.SetEntry(curInfo.month.ToString());
for (int i = 0, j = 0; i < 35; i++)
{
    if (i < curInfo.startDay || j >= curInfo.days)
    {
        lvls[i].Setup(null);
        lvls[i].OnLvlClick -= OnLvlClick;
        continue;
    }
    lvls[i].Setup(lvlPlayerDatas[j]);
    j++;
}
}

```

Кожній клітинці надається відповідний рівень. Натискаючи на неї, відкривається відповідне меню (рис. 2.32), що має у собі усю інформацію про рівень, та робить цей рівень активним.



Рисунок 2.32 – Інформація про рівень

Сценарій цього меню отримує у методі налаштування інформацію про проходження рівня гравцем. Лістинг коду методу налаштування візуального налаштування:

```
public void Setup(LvlPlayerData data, bool isActive = true)
{
    if (data == null) return;
    lvlData = data;
    day.text = data.lvl.Day.ToString();
    month.SetEntry(data.lvl.Month.ToString());
    goals.Setup(data.lvl.goals);

    for (int i = 0; i < 3; i++)
    {
        starsTxt[i].text =
data.lvl.turns.turnForStar[i].ToString();
        stars[i].sprite = i + 1 <= data.stars? Completed:None;
    }

    if (energy.GetEnergy() >= 2)
    {
        start.interactable = true;
        ad.gameObject.SetActive(false);
    }
    else
    {
        start.interactable = false;
        ad.gameObject.SetActive(true);
    }
    if (data.lvl.unlockChief != null)
    {
        chief.gameObject.SetActive(true);
        chiefs.SetEntry(data.lvl.unlockChief.name);
    }
    else
    {
        chief.gameObject.SetActive(false);
    }
    money.text = data.lvl.moneyFromLvl.ToString();

    gameObject.SetActive(isActive);
    LayoutRebuilder.ForceRebuildLayoutImmediate(info.transform
as RectTransform);
    LayoutRebuilder.ForceRebuildLayoutImmediate(all.transform as
RectTransform);
    Canvas.ForceUpdateCanvases();
}
```

При недостатній кількості енергії для проходження рівня, активною стає кнопка реклами, що після перегляду завантажує сцену з рівнем.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		76

Перейдемо до реалізації списку шефів. Кожен шеф відповідає конкретному продукту, тому зручним буде відсортовувати їх за типом. Для цього зробимо відповідні вкладки з зображенням типу продукту на них (рис. 2.33).



Рисунок 2.33 – Інформація про рівень

Реалізуємо сценарій лінії вкладок, що відслідковуватиме натискання на них й передаватиме сценарію відображення шефів відповідний тип.

Шефи відображенні у виді карточок (рис. 2.34), де у середині знаходиться вони, а під ними написаний їх рівень. Не розблоковані шефи відображаються карточкою «?».



Рисунок 2.34 – Відображення доступних шефів

Натискаючи на карточку передається відповідному меню (рис. 2.35) інформація щодо шефа гравця.



Рисунок 2.35 – Відображення інформації шефа

На цьому вікні виводиться уся потрібна інформація: рівень, назва шефа, бонус до якого гему, властивість, зображення шефа, опис властивості, вивід інформації щодо наступного рівня й його вартість. Лістинг коду покращення рівня шефу:

```
public void LvlUp()
{
    ChiefSO chief = data.chief;
    ChiefLvlInfo lvlInfo = chief.GetLvlInfo(data.lvl+1);
    if (lvlInfo == null)
    {
        return;
    }
    if (wallet.Money.Amount >= lvlInfo.lvlCost)
    {
        wallet.Money.Spend(lvlInfo.lvlCost);
        db.UpdateChiefData(data.chief, data.lvl + 1);
        ChangeVisual();
    }
}
```

З повним лістингом коду сценарію відображення інформації шефа можна ознайомитись у додатку Е.

Наступним реалізуємо сторінку інтер'єру (рис. 2.36). Розмістимо спрайт столу поцентру, а спрайти стільчиків поруч по обидва боки. Також розмістимо світильники на стіні.



Рисунок 2.36 – Сторінка інтер'єру

До кожного елемента додамо сценарій, що відслідковуватиме події пов'язані зі зміною інформацію щодо даних гравця. Також відслідковуватиметься натискання на ці елементи, що викликатимуть інформаційне меню (рис. 2.37) про відповідний тип елемента інтер'єру. Це меню дає можливість користувачу змінити варіант елемента або покращити його рівень. Лістинг коду цих методів:

```
public void ChangeCurr(int curr)
{
    db.ChangeCurrSpriteByType(data.type, curr);
    Setup(db.GetPlayerDataByType(data.type), currInt);
}
public void Upgrade()
{
    wallet.Stars.Spend(cost);
    db.UpgradeType(data.type);
    Setup(db.GetPlayerDataByType(data.type), currInt);
}
```



Рисунок 2.37 – Інформаційне меню елемента інтер'єру

У ході гри потрібно інформувати гравця про розблокування ним шефів (рис. 2.38). Сценарій щодо відображення інформації схожий на відображення шефів на їх сторінці.



Рисунок 2.38 – Інформація про розблокованого шефа

При переході до самого рівня, гравця зустрічатиме вікно з підказкою (рис. 2.39). Лістинг коду його налаштування:

```
public void Setup(LvlPlayerData data)
{
    plData = data;
    if (data.lvl.tips.video == null || data.tipsShowed)
    {
        OnTipsEnded?.Invoke();
        gameObject.SetActive(false);
        return;
    }
    video.clip = data.lvl.tips.video;
    tips.SetEntry(data.lvl.tips.text);
    LayoutRebuilder.ForceRebuildLayoutImmediate(tips.transform
as RectTransform);
    LayoutRebuilder.ForceRebuildLayoutImmediate(transform as
RectTransform);
    Canvas.ForceUpdateCanvases();
}
```



Рисунок 2.39 – Меню підказки

У сцені проходження рівня є 5 вікон:

- вікно, що інформує про завдання на рівень і зникає (рис. 2.40);
- вікно, що інформує про завдання на рівень, викликається натисканням кнопки меню, та надає можливість перезавантажити рівень;
- вікно поразки, з кнопками повернення у кафе, спробувати знову та додаткові ходи за перегляд реклами (рис. 2.41);
- вікно успіху, що відображає отримані зірки, ігрову валюту та, за наявності, розблокованого шефа.

Методи перезапуску та отримання додаткових ходів реалізуємо у сценарії гри.



Рисунок 2.40 – Інформація перед початком рівня

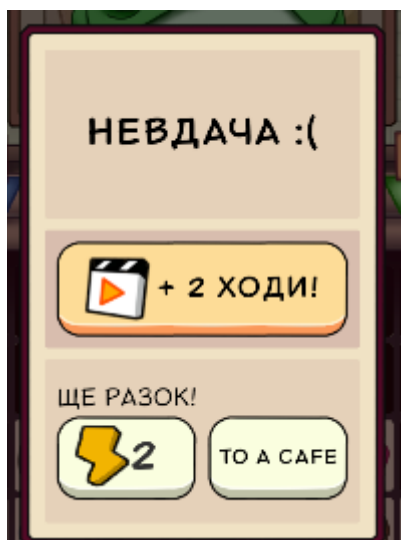


Рисунок 2.41 – Меню невдачі.

Інформаційну панель рівня розмістимо згори (рис. 2.42). Тут відображатиметься інформація про стан задоволення клієнта, завдання на рівень та кнопка меню.



Рисунок 2.42 – Інформаційна панель

Реалізуємо також перед початком рівня вибір шефів. Натискаючи на відповідні робочі місця, меню відображатиме доступних шефів (рис. 2.43).

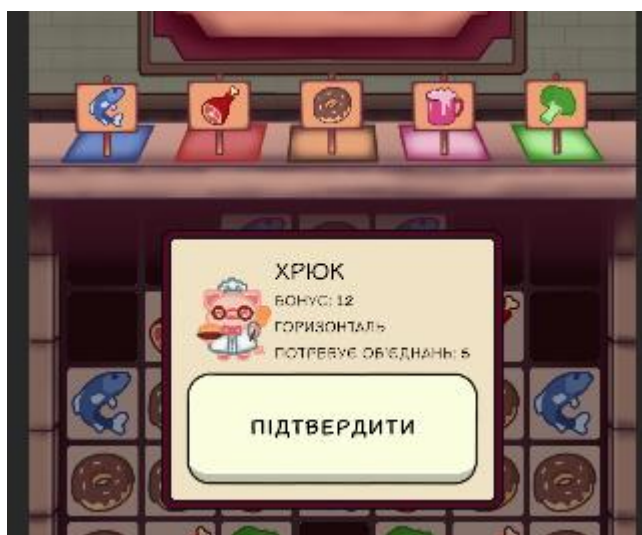


Рисунок 2.43 – Меню вибору шефа

Так як у грі мають бути дві мови: українська та англійська, використаємо пакет Localization [17], що надає інструменти для додавання підтримки декількох мов і регіональних варіантів до програми (наприклад, підтримка тексту кількома мовами). Отож, налаштуємо цей пакет, додавши у відповідні вікна мови, що будуть у грі (рис. 2.44). По замовчуванню виберемо мову англійську. Та приступимо до реалізації можливості переключатись між ними. Для цього створимо сценарій локалізації. При запуску перевірятиметься встановлена мова системи й мінятиме, якщо це українська – на українську, якщо інша – англійська. Або мінятиме мову на ту, яка була обрана користувачем уже безпосередньо в грі. Цю інформацію також зберігатимемо у PlayerPrefs за відповідним ключем.

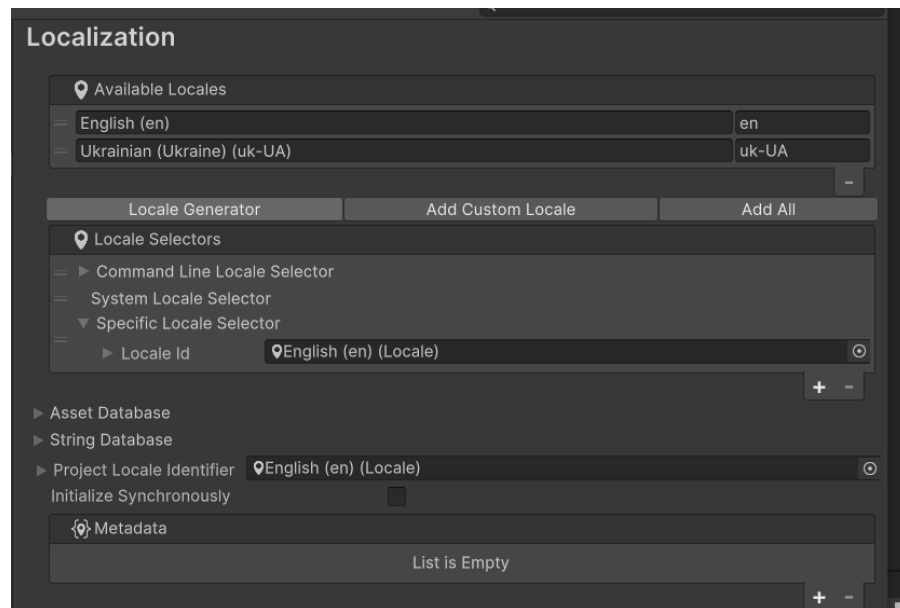


Рисунок 2.44 – Налаштування локалізатора

Лістинг коду методу зміни мови:

```
IEnumerator SetLocate(int localeID)
{
    active = true;
    yield return LocalizationSettings.InitializationOperation;
    LocalizationSettings.SelectedLocale =
LocalizationSettings.AvailableLocales.Locales[localeID];
    data = localeID;
    PlayerPrefs.SetInt("Language", data);
    PlayerPrefs.Save();
    active = false;
}
```

Використаємо таблиці ресурсів локалізації (рис. 2.45), що відповідає за повернення ресурсу на запит за ключем (ідентифікаційне ім'я або унікальний ідентифікатор). У таблиці ресурсів один стовпчик містить ключ, а інший стовпчик або стовпчики містять ресурс (або посилання на ресурс, наприклад, GUID) для вибраної локалі.

Створимо кілька таблиць для кожного випадку. Краще використовувати кілька таблиць, так як потрібно локалізувати велику кількість ресурсів, але не потрібно звертатися до них усіх одночасно. Використання менших таблиць може допомогти зменшити використання пам'яті, особливо під час попереднього завантаження таблиць.

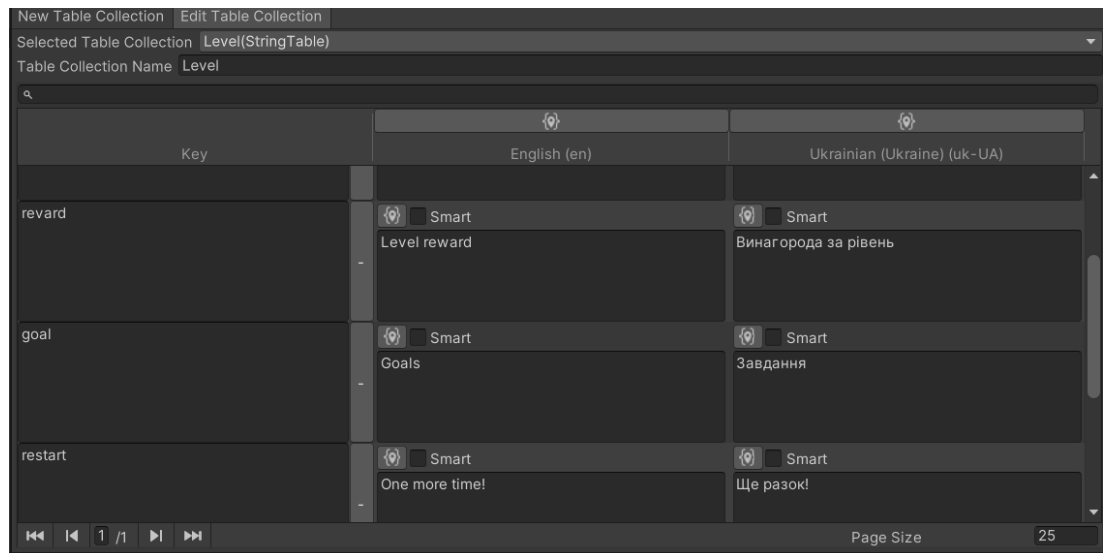


Рисунок 2.45 – Таблиця ресурсів для рівнів

Після чого додамо на відповідні елементи компонент Localize String Event та налаштуємо відповідно до ключів та підпишемо компонент TextMesh на подію зміни стану (рис. 2.46).

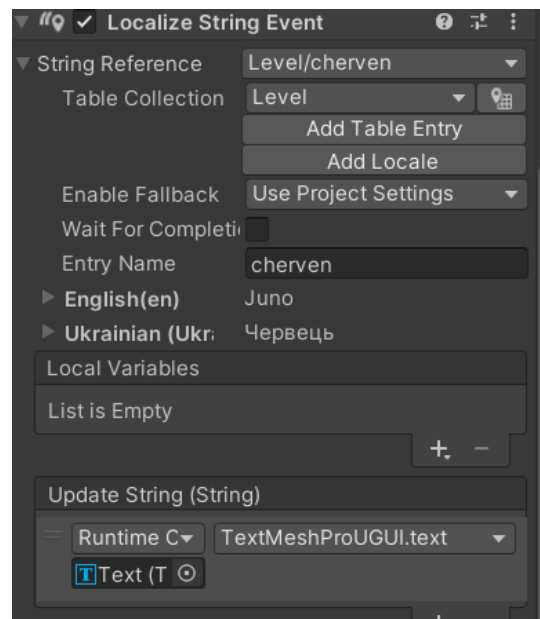


Рисунок 2.46 – Компонент Localize String Event

Для місць, де динамічно змінюється текст, налаштуємо код. До прикладу, зміна назви місяця під час його зміни: отримуємо посилання на компонент у елементі та методом SetEntry змінюємо значення за ключем. Лістинг частини коду зміни тексту:

```
month.SetEntry(data.lv1.Month.ToString());
```

У місцях, де у тексті присутні цифри, використаємо розумні ресурси. Для цього у таблицях ставимо відповідну позначку та «{ }» й ідентифікатором у середині вказуємо місце для аргументів (рис. 2.47)

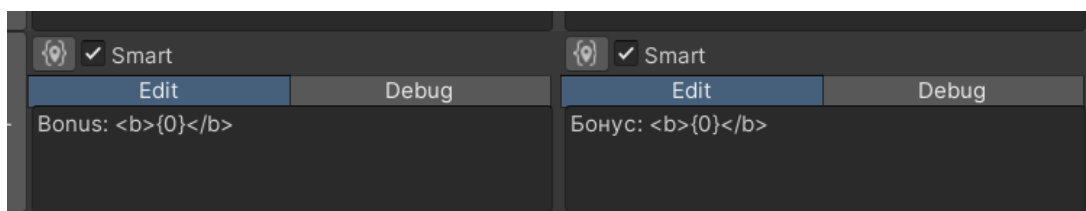


Рисунок 2.47 – Розумні ресурси

Після чого налаштуємо код для цих компонентів. Для цього у компонент передаємо аргументи й викликаємо подію оновлення тексту. Варто перед цим підписати компонент TextMesh на відповідну подію. Лістинг фрагменту коду з аргументами:

```
newLvlTxt.StringReference.StringChanged += (string t) =>
newLvl.text = t;
newLvlTxt.StringReference.Arguments = new object[] {
nextlvlInfo.yummyBonus };
newLvlTxt.RefreshString();
```

Висновки до другого розділу

У ході виконання другого розділу виконано процес проєктування та розробки програмного забезпечення. Спочатку було визначено алгоритм та стани програмної системи. Серед ігрових механік було реалізовано ключові системи: ігрової сітки, шефів, клієнтів, рівнів та їх побудови, а також систему збереження даних гравця. Додатково інтегровано рекламу у гру, що забезпечує її монетизацію. Було створено графічні елементи, які забезпечують привабливий візуальний стиль, розроблено користувацький інтерфейс для зручності взаємодії гравців з грою.

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ЗАСТОСУНКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1. Порядок встановлення та налаштування системи

Протягом розробки проекту гри було присвоєно назву My Juno Cafe, що відображає його концепцію та атмосферу. Після побудови релізної версії, гру було викладено на платформі Google Play, де гравці зможуть завантажити файл та встановити на свій пристрій. Гру можна встановити за посиланням: <https://play.google.com/store/apps/details?id=com.ThatOneFriend.MyJunoCafe>.

Збірку виконано для операційної системи Android.

Щоб застосунок коректно працював та зберігав достатню частоту кадрів у секунду, є такі мінімальні вимоги до системи:

- операційна система: Android 13 (API рівень 33)
- процесор (CPU): ARMv7 або новіший з підтримкою NEON, або x86
- оперативна пам'ять (RAM): 2 ГБ
- вільне місце на диску: 100 МБ (включаючи 39.8 МБ для інсталяції гри та додатковий простір для даних)
- графічний процесор (GPU): підтримка OpenGL ES 3.0 або новіше
- екран: Роздільна здатність 720p (1280 x 720) або вище
- інтернет-з'єднання: необхідне для завантаження оновлень та показу реклами (опціонально для гри)

3.2. Інтерфейс та порядок роботи

При першому завантаженні, гравця зустріне початковий екран, що інформуватиме про передісторію самої гри. Після чого гравцеві буде проведено короткий екскурс по сторінкам гри, розповідаючи про можливості на них (рис. 3.1). Це початок гри.

Після чого гравець може перейти до безпосереднього проходження рівнів на відповідній сторінці. Вибравши рівень, гравець бачить інформацію щодо

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		86

нього, після чого натискаючи на кнопку «почати» переходить на сцену рівня (рис. 3.2).



Рисунок 3.1 – Перше повідомлення та підказки



Рисунок 3.2 – Вибір рівня

На рівні, гравця зустрічає підказка, що допомагає гравцеві легше увійти у гру. Для наглядного прикладу текст супроводжується відео. Після чого спливаюче вікно показує знову інформацію про рівень (рис. 3.3).

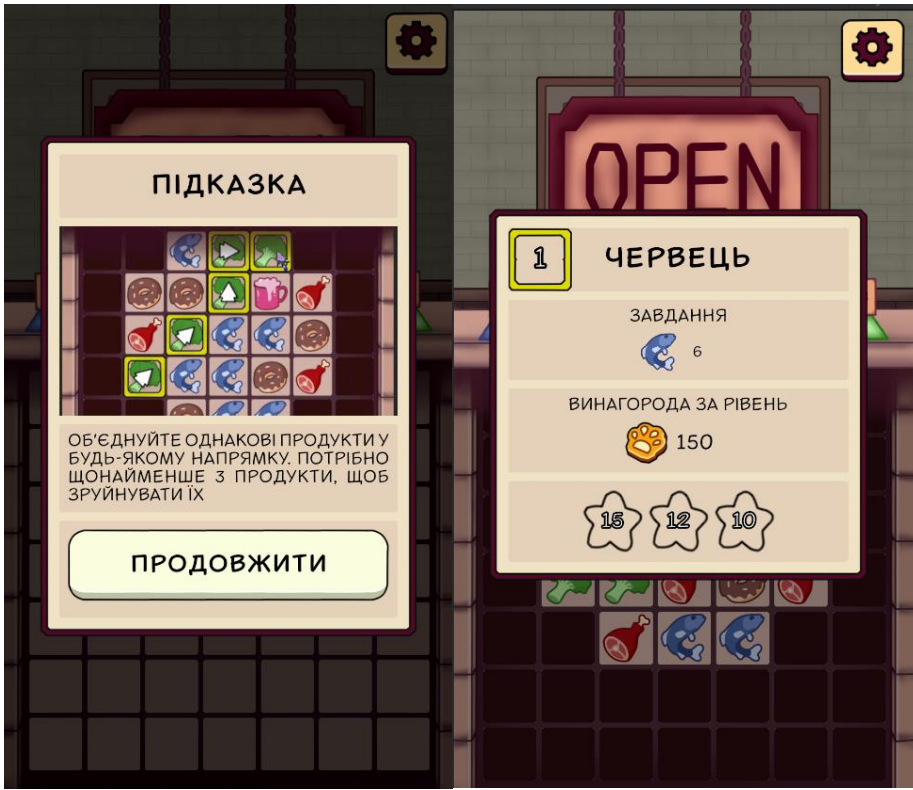


Рисунок 3.3 – Початок рівня

Об’єднуючи геми, гравець виконує завдання. При успішному виконанні гравець бачить відповідне вікно (рис. 3.4).

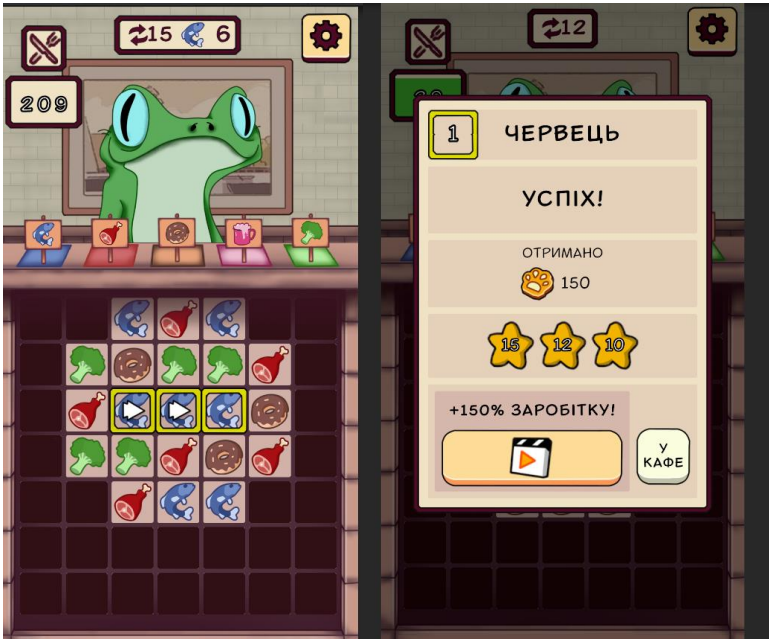


Рисунок 3.4 – Початок рівня

Гравець має можливість збільшити прибуток, переглянувши реклами. Після її перегляду, гравець отримує винагороду (рис. 3.5).

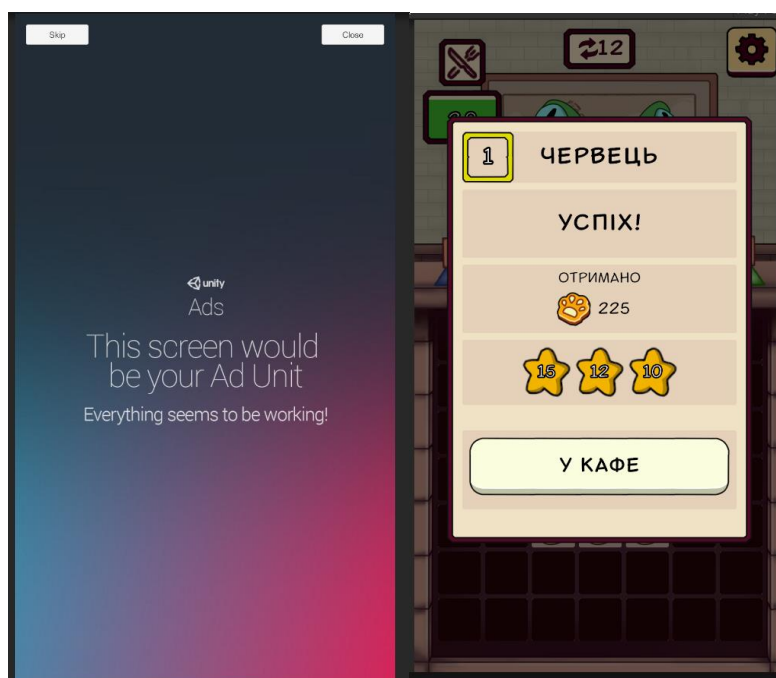


Рисунок 3.5 – Перегляд реклами за винагороду

Після чого гравець повертається до головного меню, де відображається, якщо у рівні це передбачено, розблокуємий шеф та видозмінений календар, а саме відображається отримані зірки за пройдений і позначається розблокований новий рівень (рис. 3.6).



Рисунок 3.6 – Вид після проходження рівня

Гравець може переглянути наявних шефів у відповідному вікні, а також переглянути детальну інформацію про них (рис. 3.7).



Рисунок 3.7 – Перегляд шефів і їх детальної інформації

Гравець покращує шефа, інформація оновлюється (рис. 3.8).



Рисунок 3.8 – Вікно після покращення шефа

Тепер гравець може вибрати шефа перед початком рівня (рис. 3.9).

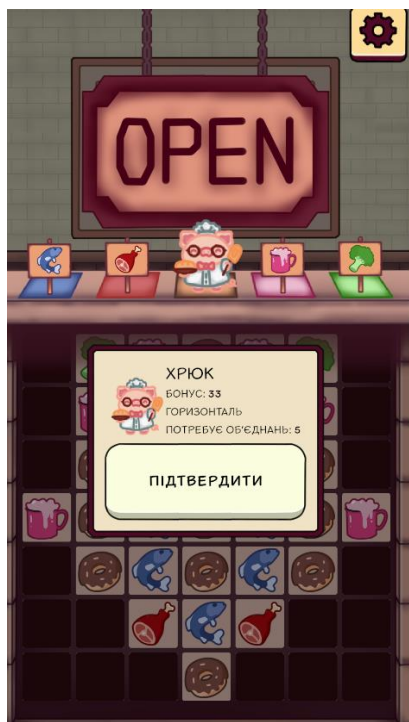


Рисунок 3.9 – Вибір шефа перед початком рівня

Гравець має можливість спробувати знову пройти рівень, або повернутись до кафе (рис. 3.10).



Рисунок 3.10 – Меню рівня

При поразці спливає відповідне вікно, що додатково пропонує отримати 2 ходи за перегляд реклами. Після перегляду її, гравець отримує винагороду й продовжує грати (рис. 3.11).

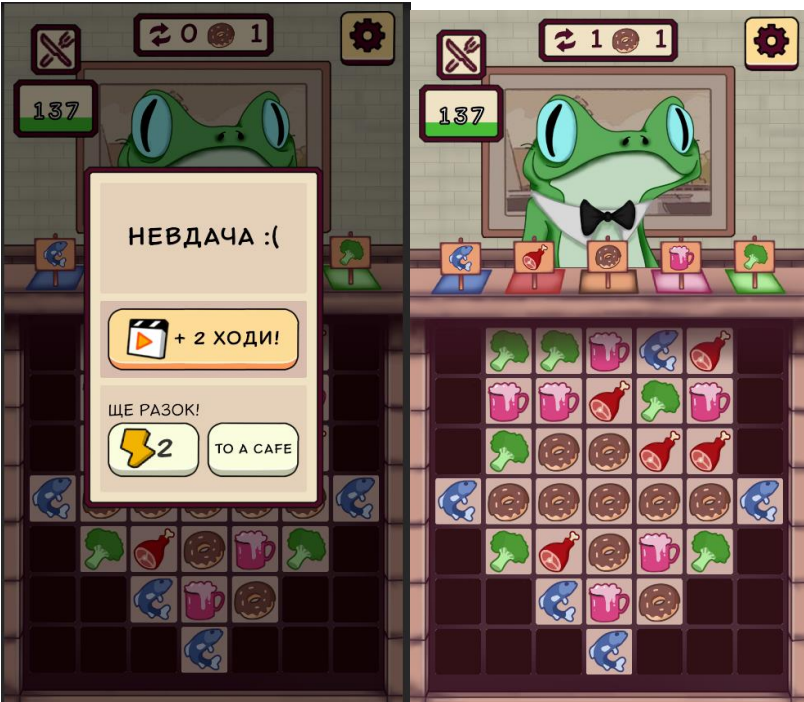


Рисунок 3.11 – Меню поразки та отримання додаткових ходів

Гравець також у головному меню може змінити мову або вимкнути звук (рис. 3.12).



Рисунок 3.12 – Налаштування

3.3. Тестування роботи застосунку

Проведено ретельне тестування користувацького інтерфейсу, включаючи навігаційні кнопки, відповідність тексту та дії, а також роботу вікон налаштувань та вибору рівнів. Результати цього тестування представлені у таблиці 3.1.

Також було здійснено перевірку на коректність оновлення візуальної інформації під час ігрового процесу, а також на правильність роботи вікон під час проходження рівня і плавність анімації. Ці аспекти детально описані у таблиці 3.2.

У ході іншого тестування перевірялася правильність спрацювання реклами, показ реклами та отримання за неї винагороди. Ці результати наведено у таблиці 3.3.

Окрім цього, було проведено тестування основних механік гри, зокрема поведінки гемів, бонусів шефів, а також правильності об'єднання та зберігання прогресу гравця. Ці дані представлені у таблиці 3.4.

Також проведено тестування процесу створення нових рівнів та їх збереження, що відображено у таблиці 3.5.

Таблиця 3.1

Тестування на правильність роботи користувацького інтерфейсу

No	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Навігація кнопок	Перехід до вікна, про яке повідомляє кнопка	Правильна навігація	Пройдено
2	Перевірка налаштувань	Кнопки вимикають звук, міняють розширення	Правильна робота кнопок налаштувань	Пройдено
3	Вибір рівня	Показ інформації про рівень та можливість почати його	Інформація відображається	Пройдено
4	Вивід шефів	Показ інформації про наявних шефів та можливість відкрити детальну інформацію	Правильна навігація	Пройдено
5	Можливість покращити шефа	Витрачається ігрова валюта та рівень шефа покращується	Витрачається ігрова валюта та рівень шефа покращується	Пройдено

Таблиця 3.2

Тестування на коректність оновлення візуальної інформації

No	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Перевірка оновлення рахунку по ходу гри	Оновлення рахунку по ходу гри	Рахунок оновлюється коректно	пройдено
2	Перевірка зміна кількості ходів	Змінюється кількість доступних ходів	Ходи коректно змінюються	пройдено
3	Перевірка оновлення завдань на рівень	Завдання оновлюються коректно	Завдання оновлюються коректно	пройдено
4	Перевірка вікна програшу та виграшу	У кінці рівня появляється відповідне вікно	У кінці рівня появляється відповідне вікно	пройдено

Таблиця 3.3

Тестування на коректну поведінку реклами

No	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Перевірка пропонування під час програшу	Під час програшу виникає вікно з пропозицією додати пару кроків за перегляд реклами	Під час програшу виникає вікно з пропозицією додати пару кроків за перегляд реклами	пройдено
2	Перегляд реклами	Реклама відображається коректно	Реклама відображається коректно	пройдено
3	Перевірка коректного отримання винагороди	Після завершення перегляду отримано додаткові кроки	Після завершення перегляду гравець отримує додаткові кроки	пройдено

Таблиця 3.4

Тестування основних механік гри

No	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Перевірка падіння гемів	Геми падають і заповнюють ігрове поле вертикально униз. Після по можливості падають діагонально	Геми падають і заповнюють ігрове поле вертикально униз. Після по можливості падають діагонально	пройдено
2	Перевірка з'єднання гемів	Правильне руйнування обраного ряду	Правильне руйнування обраного ряду	пройдено

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		94

Продовження таблиці 3.4

3	Перевірка активних об'єктів	Активация сусідніх від ряду об'єднаних об'єктів	Активация сусідніх від ряду об'єднаних об'єктів	пройдено
4	Перевірка бонусних гемів	Коректна робота бонусних гемів	Коректна робота бонусних гемів	пройдено
5	Перевірка зберігання прогресу	Прогрес зберігається	Прогрес зберігається	пройдено

Таблиця 3.5

Тестування редактора рівнів

No	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Вибір об'єктів для заповнення ігрового поля	Ігрові об'єкти обираються коректно	Ігрові об'єкти обираються коректно	пройдено
2	Призначення об'єкта до клітинки	Ігрові об'єкти прив'язані до клітинки	Ігрові об'єкти прив'язані до клітинки	пройдено
3	Збереження рівня	Новостворені рівні зберігаються	Новостворені рівні зберігаються	пройдено

У деяких рівнях були завеликі потреби у рині задоволення клієнтів, що ускладнює, а іноді унеможлиблює, проходження рівнів. Для вирішення проблеми було змінено баланс деяких рівнів.

Іноді у рівнях зустрічались місця, у які геми не можуть потрапити, або дуже рідко формують лінію для об'єднання, через що виконання завдань було не можливим. Було змінено побудову відповідних рівнів, або змінено завдання для нього.

При зміні фокусу гри (згортанні) відновлення енергії призупинялось. Ця проблема вирішена доданням функції OnFocus, де при значенні false йде збереження інформації, а при true – її завантаження й налаштування.

Після цього було проведено фінальне тестування, яке не виявило проблем або недоліків.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		95

Висновки до третього розділу

У ході виконання третього розділу було визначено порядок встановлення, платформу та мінімальні системні вимоги для гри. Було розглянуто порядок дій користувача впродовж гри від її початку, демонструючи весь функціонал гри. Було успішно проведене тестування усіх аспектів гри та вирішено знайдені проблеми, після чого проведено фінальне тестування для затвердження, що гра відповідає заданим вимогам.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		96

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено ігровий застосунок та втілено у ньому ігрову механіку жанру tile-matching.

На початку проведено аналіз задачі та існуючих популярних ігор у жанрі запланованого проєкту, що дозволило точно визначити функціональні вимоги до гри. Було створено концепт-документ, який включає жанрову спрямованість, цільову аудиторію, особливості геймплею та платформу, що дозволило чітко окреслити рамки проєкту.

На етапі проєктування було розроблено алгоритм роботи програмної системи і деталізовано різні стани за допомогою схем варіантів використання, активності, класів. У процесі розробки були створено й описано основні ігрові механіки: система ігрової сітки, шефів, клієнтів, рівнів, а також системи збереження даних гравця. Інтеграція реклами забезпечила додаткові можливості монетизації гри. Значна увага була приділена розробці графічних елементів, які підвищують візуальну привабливість гри, і створенню користувацького інтерфейсу для забезпечення зручності взаємодії гравця з грою.

Завершальним етапом було визначення порядку встановлення, платформу та мінімальні системні вимоги для гри. Було розглянуто порядок дій користувача впродовж гри від її початку, демонструючи весь функціонал гри. Проведено тестування програми, де визначено та виправлено помилки, а також вдосконалено різні аспекти гри для забезпечення стабільної роботи та високої якості геймплею, після чого проведено фінальне тестування для затвердження, що гра відповідає заданим вимогам. В додатках роботи приведено лістинг важливих функцій.

В результаті роботи отримано гру, яка коректно працює та реагує на усі команди гравця.

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		97

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WHAT IS A MATCH 3 GAME? – 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://retrostylegames.com/blog/what-is-match-3/>
2. Best Fiends - Match 3 Puzzles – 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.Seriously.BestFiends>
3. Homescapes – 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.playrix.homescapes>
4. Candy Crush Saga – 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://play.google.com/store/apps/details?id=com.king.candycrushsaga>
5. Welcome to Unity Ads – 2024. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity.com/ads/en-us/manual/UnityAdsHome>
6. PaintTool SAI – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://systemax.jp/en/sai/>
7. Варіанти використання та сценарії (Use Cases and Scenarios) – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.maxzosim.com/use-cases-and-scenarios/>
8. Дізнайтеся все про діаграму активності UML – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mindonmap.com/uk/blog/uml-activity-diagram/>
9. Before anything else (DOTween) – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://dotween.demigiant.com/getstarted.php>
10. There are 3 ways to create a Tweener – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://dotween.demigiant.com/documentation.php>
11. Namespace UnityEngine.EventSystems – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/api/UnityEngine.EventSystems.html>
12. Singleton Design Pattern – 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://sourcemaking.com/design_patterns/singleton

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		98

13. Observer Design Pattern – 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://sourcemaking.com/design_patterns/observer

14. Object Pool Design Pattern – 2023. [Електронний ресурс] – Режим доступу до ресурсу: https://sourcemaking.com/design_patterns/object_pool

15. Діаграма класів UML – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/>

16. Unity developer integration guides – 2023. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity.com/ads/en/us/manual/UnityDeveloperIntegrations>

17. Using Localization – 2019. [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.unity3d.com/Packages/com.unity.localization@0.4/manual/index.html>

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		99

ДОДАТКИ

		Б.С. ДЗЮБАНЧУК			КН.КР.Б – 122 – 24 – ПЗ	Арк.
		В.Л. ЛЕВКІВСЬКИЙ				
Змн.	Арк.	№ докум.	Підпис	Дата		100

ЛІСТИНГ КОДУ МЕНЕДЖЕРА КЛІЄНТІВ

```
using UnityEngine;
public class CustomerManager : MonoBehaviour
{
    [SerializeField] Transform poolPos;
    [SerializeField] Customer prefab;
    [SerializeField] UIFast fast;
    [SerializeField] StatusBar bar;

    private CustomerPool pool;
    private int satPool;
    private Customer currentCus;
    int moneyFromCustomer;
    [SerializeField] CustomerInfo[] customerInfo;
    [SerializeField] db_CustomerSO customersSO;
    public int totalMoney = 0;
    void Start()
    {
        pool = new CustomerPool(poolPos, prefab, 10);
        EventManager.instance.OnChiefBonus += AddSat;
        EventManager.instance.OnGameStarted += OnGameStartHandler;
    }
    private void OnGameStartHandler()
    {
        SpawnNewCustomer();
    }
    public void Setup(CustomerInfo[] customer, int moneyFromCustomer)
    {
        this.customerInfo = customer;
        this.moneyFromCustomer = moneyFromCustomer;
    }
    private void OnCusReadyHandler(Customer customer)
    {
        currentCus = customer;
        if (satPool > 0)
        {
            satPool = currentCus.AddSatisfaction(satPool);
        }
    }
    private void OnCusSatHandler(Customer customer)
    {
        int money = moneyFromCustomer + (moneyFromCustomer * customer.bonus / 100);
        totalMoney += money;
        EventManager.instance.OnMoneyEarned?.Invoke(money);
        customer.OnCustomerSatisfied -= OnCusSatHandler;
        customer.OnCustomerReady -= OnCusReadyHandler;
        currentCus = null;
        SpawnNewCustomer();
    }
    public void SpawnNewCustomer()
    {
        Customer customer = pool.Get();
        customer.transform.SetParent(transform);
        customer.SetupVisual(customersSO.GetRandomCustomer());
        customer.Setup(GetCustomerInfo(), bar, fast);
        customer.OnCustomerSatisfied += OnCusSatHandler;
        customer.OnCustomerReady += OnCusReadyHandler;
        customer.Spawn();
    }
    private CustomerInfo GetCustomerInfo()
    {
        int totalChance = 0;
        foreach (var info in customerInfo)
        {
```

```

        totalChance += info.chancePercent;
    }
    int randomNum = Random.Range(0, totalChance);
    int cumulativeChance = 0;
    foreach (var info in customerInfo)
    {
        cumulativeChance += info.chancePercent;
        if (randomNum < cumulativeChance)
        {
            return info;
        }
    }
    return customerInfo[0];
}
private void AddSat(GemType type, int value)
{
    satPool += value;
    if (currentCus != null)
    {
        satPool = currentCus.SatWithFast(type, value);
    }
}
}

```

Лістинг коду менеджера редактора сітки

```

using UnityEngine;
#if UNITY_EDITOR
using UnityEditor;
#endif
public class BuilderManager : MonoBehaviour
{
    public LvlSO lvl;
    int width = 7;
    int height = 7;
    public FloorType floorType;
    public BorderType borderType;
    public GemType gemType;
    private GridObject currentPref;
    private BGridCell activeCell;
    BGridCell[] visualCell;
    BGridCell[,] gridCells;
    private void Start()
    {
        visualCell = GetComponentsInChildren<BGridCell>();
        gridCells = new BGridCell[width, height];
        int index = 0;
        for (int y = height - 1; y >= 0; y--)
        {
            for (int x = 0; x < width; x++)
            {
                gridCells[x, y] = visualCell[index];
                gridCells[x, y].OnBGCClick += SetActiveCell;
                gridCells[x, y].Setup(lvl.cells[index]);
                index++;
            }
        }
    }

    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            SetObjectToCell();
        }
        if (Input.GetKeyDown(KeyCode.W))
        {
            AddFloorToCell();
        }
        if (Input.GetKeyDown(KeyCode.Q))
        {
            RemoveFloorInCell();
        }
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            AddBorderToCell(Direction.Left);
        }
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            AddBorderToCell(Direction.Top);
        }
        if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            AddBorderToCell(Direction.Right);
        }
        if (Input.GetKeyDown(KeyCode.Alpha4))
        {
            AddBorderToCell(Direction.Bottom);
        }
    }
}

```

```

        if (Input.GetKeyDown(KeyCode.Alpha5))
        {
            RemoveBorderInCell(Direction.Left);
        }
        if (Input.GetKeyDown(KeyCode.Alpha6))
        {
            RemoveBorderInCell(Direction.Top);
        }
        if (Input.GetKeyDown(KeyCode.Alpha7))
        {
            RemoveBorderInCell(Direction.Right);
        }
        if (Input.GetKeyDown(KeyCode.Alpha8))
        {
            RemoveBorderInCell(Direction.Bottom);
        }
    }
#if UNITY_EDITOR
        if (Input.GetKeyDown(KeyCode.S))
        {
            Save();
        }
#endif
    }
    private void SetObjectToCell()
    {
        activeCell.SetGridObject(currentPref, gemType);
    }
    private void AddBorderToCell(Direction dir)
    {
        activeCell.AddBorder(dir, borderType);
    }
    private void RemoveBorderInCell(Direction dir)
    {
        activeCell.RemoveBorder(dir);
    }
    private void AddFloorToCell()
    {
        activeCell.AddFloor(floorType);
    }
    private void RemoveFloorInCell()
    {
        activeCell.RemoveFloor();
    }
    private void SetActiveCell(BGridCell cell)
    {
        activeCell = cell;
    }
    public void Clear()
    {
        currentPref = null;
    }

    public void ChoosePref(GridObject pref)
    {
        currentPref = pref;
    }
#if UNITY_EDITOR
    public void Save()
    {
        lvl.cells = new CellLvlInfo[49];
    }

```

```
    for (int x = 0; x < 49; x++)
    {
        lvl.cells[x] = visualCell[x].GetCellInfo();
    };
    EditorUtility.SetDirty(lvl);
    AssetDatabase.SaveAssets();
}
#endif
}
```

Лістинг коду сценарію гри

```

using DG.Tweening;
using UnityEngine;
using UnityEngine.SceneManagement;
using static UnityEngine.EventSystems.EventTrigger;

public class Game : MonoBehaviour
{
    [SerializeField] GridManager gridManager;
    [SerializeField] CustomerManager customerManager;
    [SerializeField] GoalManager goalManager;
    [SerializeField] LvlPlayerData playerLvlData;
    [SerializeField] LvlPlayerData testData;
    [SerializeField] db_Chief chiefs;
    public TurnManager turnManager;
    public EnergyManager energyManager;
    public static GameStat gameStat;

    [Header("Menu bars")]
    [SerializeField] UIStatStart menuStart;
    [SerializeField] UIStatLvl menuPause;
    [SerializeField] UISuccess menuSuccess;
    [SerializeField] UIFailure menuFailure;
    [SerializeField] UITips tips;

    public void Start()
    {
        gameStat = null;
        playerLvlData = LvlSelector.LvL;
        if (playerLvlData == null)
            playerLvlData = testData;
        tips.OnTipsEnded += OnTipsEnded;
        tips.Setup(playerLvlData);
        EventManager.instance.OnAllGoalAchived += OnGameSuccess;
    }
    private void OnTipsEnded()
    {
        LvlSO lvl = playerLvlData?.lvl;
        turnManager = new TurnManager(lvl.turns);
        turnManager.OnTurnsEnded += OnGameFailed;
        customerManager.Setup(lvl.customers, lvl.moneyFromCustomer);
        goalManager.Setup(lvl.goals);
        goalManager.SetupTurns(turnManager);
        gridManager.Setup(lvl);
        menuStart.Setup(playerLvlData);
        menuPause.Setup(playerLvlData, false);
        energyManager.Setup();
    }
    public void RestartGame()
    {
        DOTween.KillAll();
        LvlSelector.PlayedCount++;
        if (LvlSelector.PlayedCount % 3 == 0)
        {
            AdManager.Instance.interstitial.ShowAd(() =>
            {
                energyManager.SpendEnergy(2);
                LvlSelector.LvL = playerLvlData;
                SceneManager.LoadScene(1);
            },
            () =>
            { });
        }
    }
}

```

```

        return;
    }
    energyManager.SpendEnergy(2);
    LvlSelector.LvL = playerLvldata;
    SceneManager.LoadScene(1);
}

public void BackToMenu()
{
    DOTween.KillAll();
    SceneManager.LoadScene(0);
}

public void GetBonusMoney()
{
    AdManager.Instance.rewarded.ShowAd(() =>
    {
        menuSuccess.ShowAd();
        gameStat.money += gameStat.money * 50 / 100;
        menuSuccess.Setup(gameStat);
    }, () =>
    {
        menuSuccess.Setup(gameStat);
    });
}

public void GetExtraTurns()
{
    AdManager.Instance.rewarded.ShowAd(() =>
    {
        turnManager.curentTurn -= 2;
        goalManager.UpdateTurns(turnManager);
        menuFailure.gameObject.SetActive(false);
    },
    () =>
    { });
}

public void GetExtraEnergyAndRestart()
{
    AdManager.Instance.rewarded.ShowAd(() =>
    {
        LvlSelector.LvL = playerLvldata;
        SceneManager.LoadScene(1);
    }, () => { });
}

private void OnGameSuccess()
{
    menuSuccess.gameObject.SetActive(true);
    int stars = Mathf.Clamp(turnManager.GetStars() - playerLvldata.stars, 0, 3);
    gameStat = new()
    {
        lvl = playerLvldata.lvl,
        lvlStars = turnManager.GetStars() <= playerLvldata.stars ?
playerLvldata.stars : turnManager.GetStars(),
        stars = stars,
        money = playerLvldata.moneyReceived ? customerManager.totalMoney :
customerManager.totalMoney + playerLvldata.lvl.moneyFromLvl,
        moneyFromLvlRecived = true
    };
    menuSuccess.Setup(gameStat);
}

private void OnGameFailed()
{
    menuFailure.gameObject.SetActive(true);
}
}

public class GameStat

```

```
{
    public LvlSO lvl;
    public int stars;
    public int lvlStars;
    public int money;
    public bool moneyFromLvlRecived;
}
```

Лістинг коду менеджера енергії

```

using UnityEngine;
using System;
using UnityEngine.SceneManagement;

public class EnergyManager : MonoBehaviour
{
    public event Action OnTimeChanged;
    public event Action OnEnergyCharged;
    public EnergySO energySO;
    private bool isCharging;
    private bool delete;

    // Залишений час до наступного поповнення енергії

    private void Awake()
    {
        energySO.OnSetup += Setup;
        energySO.OnUpdate += EnergySOChanged;
    }

    public void Setup()
    {
        if (energySO.energy < energySO.maxEnergy)
        {
            isCharging = true;
            InvokeRepeating(nameof(UpdateEnergy), 0f, 1f); // Виклик методу кожну
секунду
        }
        else
        {
            OnEnergyCharged?.Invoke();
        }
    }

    private void UpdateEnergy()
    {
        energySO.timeLeftToRecharge -= 1; // Зменшуємо час до поповнення енергії на
одну секунду
        OnTimeChanged?.Invoke();

        if (energySO.timeLeftToRecharge <= 0)
        {
            energySO.AddEnergy(1);
            energySO.timeLeftToRecharge = energySO.energyRecoveryTime; //
Встановлюємо час до наступного поповнення енергії
            if (energySO.energy >= energySO.maxEnergy)
            {
                isCharging = false;
                CancelInvoke(nameof(UpdateEnergy)); // Скасувати виклик методу, коли
енергія досягнута максимуму
                OnEnergyCharged?.Invoke();
            }
        }
    }

    public void SpendEnergy(int value)
    {
        if (energySO.SpendEnergy(value))
        {
            if (!isCharging)
            {
                isCharging = true;
            }
        }
    }
}

```

```

        energyS0.timeLeftToRecharge = energyS0.energyRecoveryTime; //
Встановлюємо час до наступного поповнення енергії
        InvokeRepeating(nameof(UpdateEnergy), 0f, 1f); // Почати відлік, якщо
енергія була витрачена і вона не досягла максимуму
    }
}
public void EnergyS0Changed()
{
    if (energyS0.energy < energyS0.maxEnergy && !isCharging)
    {
        isCharging = true;
        energyS0.timeLeftToRecharge = energyS0.energyRecoveryTime; //
Встановлюємо час до наступного поповнення енергії
        InvokeRepeating(nameof(UpdateEnergy), 0f, 1f); // Виклик методу кожному
секунду
    }
}
private void OnApplicationFocus(bool focus)
{
    if (!focus)
    {
        CancelInvoke(nameof(UpdateEnergy));
    }
}
private void OnDisable()
{
    energyS0.OnSetup -= Setup;
    energyS0.OnUpdate -= EnergyS0Changed;
    CancelInvoke(nameof(UpdateEnergy));
}
public int GetEnergy() => energyS0.energy;
}

```

Лістинг коду об'єкту сценарію шефів

```

using System;
using System.Collections.Generic;
using UnityEngine;
[CreateAssetMenu()]
public class db_Chief : ISaveLoadSO
{
    public List<ChiefSO> all;
    public List<ChiefPlayerData> playerChief;
    private List<ChiefPlayerData> loadedData;
    public void UnlockChief(ChiefSO chief)
    {
        ChiefPlayerData chiefPlayer = playerChief.Find(x => x.chief == chief);
        if (chiefPlayer == null) return;
        if (chiefPlayer.unlocked) return;
        chiefPlayer.unlocked = true;
        chiefPlayer.lvl = 0;
        Save();
    }
    public void UpdateChiefData(ChiefSO chief, int lvl)
    {
        if (playerChief.Find(x => x.chief == chief) == null) return;
        ChiefPlayerData data = playerChief.Find(x => x.chief == chief);
        data.lvl = lvl;
        Save();
    }
    public List<ChiefPlayerData> GetUnlockedChiefsByType(GemType gem)
    {
        return playerChief.FindAll(x => x.chief.gemType == gem && x.unlocked);
    }
    public List<ChiefPlayerData> GetChiefsByType(GemType gem)
    {
        return playerChief.FindAll(x => x.chief.gemType == gem);
    }
    private List<ChiefPlayerData> GetAllNotUnlocked(GemType gem)
    {
        return playerChief.FindAll(x => x.chief.gemType == gem && !x.unlocked);
    }
    override public void Setup()
    {
        playerChief = new();
        for (int i = 0; i < all.Count; i++)
        {
            playerChief.Add(new() { chief = all[i], lvl = 0, unlocked = false });
        }
        if (loadedData == null) return;
        foreach (var load in loadedData)
        {
            if (playerChief.Find(x => x.chief == load.chief) != null)
            {
                ChiefPlayerData data = playerChief.Find(x => x.chief == load.chief);
                data.lvl = load.lvl;
                data.unlocked = load.unlocked;
            }
        }
    }
    override public void Save()
    {
        string chiefs = JsonUtility.ToJson(new ChiefsPlayerDataSave() { chiefs =
playerChief });
        PlayerPrefs.SetString("UnlockChief", chiefs);
        PlayerPrefs.Save();
    }
}

```

```

    }
    override public void Load()
    {
        loadedData = null;
        string chiefs = PlayerPrefs.GetString("UnlockChief");
        ChiefsPlayerDataSave data =
JsonUtility.FromJson<ChiefsPlayerDataSave>(chiefs);
        if (data == null) return;
        loadedData = data.chiefs;
    }
    public override void SaveGameStat(GameStat stat)
    {
        UnlockChief(stat.lvl.unlockChief);
        base.SaveGameStat(stat);
    }
    public override void Clear()
    {
        loadedData = null;
        playerChief = new();
        PlayerPrefs.DeleteKey("UnlockChief");
    }
}
public class ChiefsPlayerDataSave
{
    public List<ChiefPlayerData> chiefs;
}
[Serializable]
public class ChiefPlayerData
{
    public ChiefSO chief;
    public int lvl;
    public bool unlocked;
}

```

Лістинг коду сценарію відображення інформації шефа

```

public void ChangeVisual()
{
    ChiefSO chief = data.chief;
    ChiefLvlInfo currlvlInfo = chief.GetLvlInfo(data.lvl);
    lvlTxt.text = data.lvl.ToString();
    nameTxt.SetEntry(chief.name);
    bonusTxt.text = currlvlInfo.yummyBonus.ToString();
    ultimateNameInfoTxt.SetEntry(chief.bgType.ToString());
    ultimateInfoTxt.SetEntry(chief.bgType switch {
        BGType.saucepan=> "saucepanInfo",
        BGType.box => "boxInfo",
        BGType.vertical => "verticalInfo",
        BGType.horizontal => "horizontalInfo"
    });

    bonusImg.sprite = gemSO.GetGemSOByType(chief.gemType).sprite;
    if (chief.bgType == BGType.horizontal)
    {
        ultimateImg.transform.DORotate(new Vector3(0, 0, 90),0);
    }
    else
    {
        ultimateImg.transform.rotation = Quaternion.identity;
    }
    ultimateImg.sprite = bGSO.GetBGByType(chief.bgType).GetSprite(chief.gemType);
    foto.sprite = data.chief.sprite;
    if (chief.GetLvlInfo(data.lvl + 1) == null)
    {
        btn.interactable = false;
        costTxt.text = "---";
        newLvlTxt.SetEntry("maxLvl");
    }
    else
    {
        ChiefLvlInfo nextlvlInfo = chief.GetLvlInfo(data.lvl + 1);
        costTxt.text = nextlvlInfo.lvlCost.ToString();
        newLvlTxt.StringReference.Arguments = new object[] {
            nextlvlInfo.yummyBonus };
        newLvlTxt.RefreshString();
        if (wallet.Money.Amount < nextlvlInfo.lvlCost)
        {
            btn.interactable = false;
        }
        else
        {
            btn.interactable= true;
        }
    }
}

LayoutRebuilder.ForceRebuildLayoutImmediate(newLvlTxt.GetComponent<RectTransform>());

LayoutRebuilder.ForceRebuildLayoutImmediate(transform.GetComponent<RectTransform>());
Canvas.ForceUpdateCanvases();
}

```