

Я як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

20.12.2022.

(дата, підпис, прізвище студента/-ки)



Бондар Є.А.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено «20» грудня 2022 р.

Керівник кваліфікаційної роботи



проф. Саваневич В.Є.

Харківський національний університет радіоелектроніки
 Факультет _____ Комп'ютерних наук
 Кафедра _____ Системотехніки
 Рівень вищої освіти _____ другий (магістерський)
 Спеціальність _____ 122 – Комп'ютерні науки
 _____ (код і повна назва)
 Тип програми _____ освітньо-професійна
 Освітньо-професійна програма _____ Комп'ютерні науки
 _____ (повна назва)

ЗАТВЕРДЖУЮ:

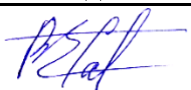
Зав. кафедри _____
 _____ (підпис)

« ____ » _____ 20 ____ р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____
 Бондарю Єгору Анатолійовичу
 _____ (прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка та дослідження ігрового сервісу у жанрі «Hyper-Casual»»
 затверджена наказом по університету від " 21 " листопада 2022 р. № 1504СТ
2. Термін подання студентом роботи до екзаменаційної комісії 20 грудня 2022 р.
3. Вихідні дані до роботи: Розробити ігровий додаток в гіпер казуальному жанрі . Ігровий додаток розробити за допомогою Unity 2020.1.1. Операційна система – MacOS. Інтегроване середовище розробки Unity 2020.3.0f1 , мова програмування – C#.NET. Для розробки програми використовувати методології і технології проектування програмних засобів.
4. Перелік питань, що потрібно опрацювати в роботі: 4.1 Вступ. 4.2 Аналіз предметної області. 4.2.1 Розвиток мобільних ігор. 4.2.2 Сучасна гіпер казуальна ігрова індустрія 4.2.3 Гіпер казуальні ігри, як казуальний піджанр. 4.2.4 Патерни проектування. 4.3 Розробка вимог до ігрового додатку. 4.3.1 Патерн проектування Singleton. 4.3.2 Патерн проектування Abstract Factory. 4.3.3 Функціональні вимоги до процесу розробки ГКІ. 4.3.4 Розробка вимог до інтерфейсу (діаграма класів, станів, прецедентів, діяльності). 4.4 Вибір та обґрунтування проектних рішень 4.4.1 Опис механіки реалізації гіпер казуальної гри 4.4.2 Вибір операційної системи. 4.4.3 Вибір середовища розробки ігрового додатку. 4.4.4 Обґрунтування вибору середовища розробки програмного коду. 4.5 Процес розробки ігрового додатку 4.5.1 Розробка ігрового циклу. 4.5.2 Розробка системи меню. 4.5.3 Інтеграція абстрактної фабрики. 4.6 Тестування ігрового додатку 4.6.1 Тестування в середовищі розробки ігрового додатку. 4.6.2 Аналіз контрольного прикладу 4.7 Висновки. 4.8 Перелік джерел посилання.
5. Перелік графічного матеріалу із зазначенням креслень, схем, плакатів, комп'ютерних ілюстрацій: 5.1 Вікова статистика (1 аркуш формату А4). 5.2 Найпопулярніші жанри в регіонах (1 аркуш формату А4). 5.3 Кількість млрд доларів прибутку від платформи (1 аркуш формату А4). 5.4 Прибуток по регіонам (1 аркуш формату А4) 5.5 Прогноз прибутку на 2025 (1 аркуш формату А4). 5.6 Короткий цикл зворотного зв'язку — гравець покращує навички, якщо програє (1 аркуш формату А4)). 5.7 Синглтон в UML (1 аркуш формату А4)). 5.8 Абстрактна фабрика в UML (1 аркуш формату А4) . 5.9 Контекстна діаграма IDEF0 (1 аркуш формату А4) 5.10 Перша декомпозиція діаграми IDEF0 (1 аркуш формату А4) 5.11 Декомпозиція «Створення ігрових рівнів» (1 аркуш формату А4) 5.12 Декомпозиція «Реалізація елементів малювання» (1 аркуш формату А4) 5.13 Декомпозиція «Створення файлів зворотнього зв'язку» (1 аркуш формату А4) 5.14 Декомпозиція «Тестування ігрового додатку» (1 аркуш формату А4) 5.15 Діаграма прецедентів (1 аркуш формату А4) 5.13 Діаграма класів (1 аркуш формату А4) 5.16 Діаграма станів (1 аркуш формату А4) 5.17 Діаграма діяльності (1 аркуш формату А4).

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спеціальна частина	Професор Саваневич В.Є.		20.12.2022

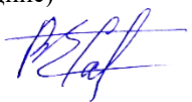
КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання кваліфікаційної роботи	12.09.2022	Виконано
2.	Аналіз завдання, літератури та аналогів з теми кваліфікаційної роботи	12.09. — 23.09.2022	Виконано
3.	Аналіз та дослідження патерні проектування	23.09. — 03.10.2022	Виконано
4.	Вибір засобів для розробки технічних вимог до програми	03.10. — 05.10.2022	Виконано
5.	Розробка та уточнення вимог методами структурного та об'єктно-орієнтованого проектування	05.10. — 18.10.2022	Виконано
6.	Вибір середовища розробки програми	18.10. — 21.10.2022	Виконано
7.	Розробка програмного забезпечення	21.10. — 24.11.2022	Виконано
8.	Аналіз даних контрольного прикладу та тестування програми, впровадження системи	24.11. — 26.11.2022	Виконано
9.	Розробка «Керівництва користувача»	26.11. — 16.12.2022	Виконано
10.	Оформлення пояснювальної записки та програмної документації	16.12. — 17.12.2022	Виконано
11.	Представлення роботи на плагіат	18.12.2022	Виконано
11.	Оформлення графічної частини та презентаційних матеріалів комп'ютерного захисту	18.12.22	Виконано
12.	Представлення на рецензування	19.12.22	Виконано
13.	Проведення попереднього захисту	20.12.22	Виконано
14.	Представлення кваліфікаційної роботи в ДЕК	21.12.22	Виконано

Дата видачі завдання «12» вересня 2022 р.

Студент 
(підпис)

Бондар Є.А.

Керівник роботи 
(підпис)

проф. Саваневич В.Є.
(посада, прізвище, ініціали)

РЕФЕРАТ

Кваліфікаційна робота : 117 с., 28 рис., 20 джерел, 4 додатки.

ГІПЕР КАЗУАЛЬНІ ІГРИ, C# .NET, UNITY, STOWAWAY, GAMEDEV.

Об'єктом розробки є процес дослідження та розробка інструментального засобу в жанрі «Hyper-Casual».

Предметом розробки є інструментальний засіб в жанрі гіпер казуальних ігор.

Метою роботи є дослідження сучасних ігор та розробка ігрового додатку в жанрі гіпер казуальних ігор.

Результатами роботи є розроблений ігровий додаток, який відповідає жанру гіпер казуальних ігор. Для визначення та уточнення функціональних вимог та інтерфейсу ігрового додатку було проведено функціональне та об'єктно-орієнтоване моделювання. Проведено дослідницьку роботу по вивченню сучасних трендів гіпер казуальних ігор. Об'єктно-орієнтоване моделювання представлено такими діаграмами: класів, прецедентів, активності та послідовностей. Проведено реалізацію ігрового додатку на обраних операційних системах. За результатами тестування проведено аналіз та апробація відповідності розробленого ігрового додатку з висунутим вимогам.

ABSTRACT

Qualification work: 117 pages, 28 images, 20 sources, 4 applications.

HYPER CASUAL GAMES, C# .NET, UNITY, STOWAWAY, GAMEDEV.

The object of development is the process of research and development of an instrumental tool in the "Hyper-Casual" genre.

The subject of development is an instrumental tool in the genre of hyper casual games.

The purpose of the work is the research of modern games and the development of a game application in the genre of hyper casual games.

The results of the work are a developed game application that corresponds to the genre of hyper casual games. Functional and object-oriented modeling was carried out to define and specify the functional requirements and interface of the game application.

Research work was carried out on the study of modern trends of hyper casual games. Object-oriented modeling is represented by the following diagrams: classes, precedents, activities, and sequences. The game application has been implemented on selected operating systems. Based on the results of the testing, an analysis and approval of the compliance of the developed game application with the proposed requirements was carried out.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК.....	8
ВСТУП.....	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	11
1.1 Розвиток мобільних ігор	11
1.2 Сучасна гіпер казуальна ігрова мобільна індустрія	13
1.3 Гіпер казуальні ігри, як казуальний піджанр.....	18
1.4 Патерни проектування	21
2 РОЗРОБКА ВИМОГ ДО ІГРОВОГО ДОДАТКУ.....	24
2.1 Патерн проектування Singleton	24
2.2 Патерн проектування Abstract Factory.....	25
2.3 Функціональні вимоги до процесу розробки ГКІ.....	26
2.3 Розробка вимог до інтерфейсу(діаграма класів, станів, прецедентів, діяльності)	32
3 ВИБІР ТА ОБГРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ	37
3.1 Опис механіки реалізації гіпер казуальної гри	37
3.2 Вибір операційної системи	38
3.3 Вибір середовища розробки ігрового додатку.....	39
3.4 Обґрунтування вибору середовища розробки програмного коду	41
4 ПРОЦЕС РОЗРОБКИ ІГРОВОГО ДОДАТКУ	44
4.1 Розробка ігрового циклу	44
4.2 Розробка системи меню	48
4.3 Інтеграція абстрактної фабрики	52
5 ТЕСТУВАННЯ ІГРОВОГО ДОДАТКУ	56
5.1 Тестування в середовищі розробки ігрового додатку.....	56
5.2 Аналіз контрольного прикладу	57
ВИСНОВКИ	64
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ	65
ДОДАТОК А	67
ДОДАТОК Б.....	85
ДОДАТОК В	91
ДОДАТОК Г.....	21

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК

ГКІ – гіпер казуальні ігри.

ГК – гіпер казуальний.

ШІ – штучний інтелект.

ПК – персональний комп'ютер.

IDE – інтегроване середовище розробки.

ОС – операційна система.

Тренд – зміна, що визначає загальний напрямок розвитку, зміну якого-небудь процесу або явища.

Скрипт – це програма або сценарій, написаний певною мовою, який автоматизує рішення деякої задачі.

Ігровий движок (Engine) – це набір систем, інструментів які спрощують роботу розробнику ігор при створенні гри.

Ігровий Ассет (від англ. Game asset) – ресурс, що складається з однотипних даних.

Сцена – сукупність графічних моделей розташованих в певному порядку.

Текстура – растрове цифрове зображення, що відтворює візуальні властивості будь-яких поверхонь або об'єктів.

Canvas – об'єкт, який відображає елементи інтерфейсу.

Spline – об'єкт, який можна змінювати та вигибати в будь-якому напрямку.

Мультиплеєр – режим гри з іншими гравцями в онлайн режимі.

Фейк – від англ. фальш, підробка.

ВСТУП

Актуальність даної теми зумовлена тим, що у зв'язку з технологічним прогресом ігрова індустрія все більше і більше заоплює світ. Ігри стали невід'ємною частиною життя сучасних людей, і це не обов'язково підлітки та діти. Старше покоління теж починає захоплюватись сучасними іграми. Проте для успішної гри, яка буде легкою у використанні та цікавою, незалежно від аудиторії, потрібно слідкувати за напрямом розвитку відеоігор та кваліфіковані спеціалісти, які за короткий проміжок часу або створять гру за трендом, або зроблять новий тренд, що призведе до успіху проекту. Розробка ігор в сучасному світі відіграє дуже важливу роль, адже, навіть, провідні психологи радять у терапії та релаксації сучасні ігри, особливо в гіпер казуальному жанрі, для відновлення або відволікання.

В кваліфікаційній роботі досліджуються методи розробки гіперказуальних ігор та дослідження сучасних трендів гіпер казуальних ігор. Розглянути питання створення провідних функціональних вимог до процесу розробки ігрового додатку як інструментального засобу, обґрунтування вибору середовища розробки програмного коду та ігрового движка. Розроблений додаток, у вигляді гіпер казуальної гри, був протестований.

Метою даної роботи є аналіз, проектування, розробка та тестування гіпер казуальної гри, як сучасний напрям у розробці мобільних ігор.

Необхідно розробити ігровий додаток в жанрі гіпер казуальних ігор. Гра повинна мати декілька рівнів складності. Рівень складності реалізувати за допомогою збільшення перепон та довжиною ігрового рівня.

Розроблювальна гіпер казуальний додаток повинен мати можливість виконуватись в середовищі операційної системі Android та IOS. Інтерфейс повинен бути зрозумілим для гравця, мати мінімально необхідне для виконання гри функціональне наповнення. Ігрова сесія обмежується проходженням одного рівня певної складності.

В рамках виконання кваліфікаційної роботи необхідно:

- формалізувати та визначити основні вимоги до розробки гіпер казуальної гри;

- за допомогою об'єктно-орієнтованого проектування визначити вимоги до інтерфейсу та проектуванню ігрового додатку;

–обґрунтувати вибір, програмного середовища, вибір операційної системи реалізації ігрового додатку та середовища розробки, що будуть представлені в основу розробки ігрового додатку;

–виконати кодування, провести тестування працездатності гіпер казуальної гри;

У програмному додатку має бути реалізовано контролювання системи інтерфейсу; ігрового наповнення; системи, яка реалізує проходження з одного рівня на інший; базового розробленого ігрового процесу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Розвиток мобільних ігор

Популярність мобільних ігор в останні десятки років постійно збільшується, а це означає, що навіть люди, які не є геймерами, на своїх телефонах мають хоча б одну гру. З появою мобільних ігрових магазинів 10 років тому ми спостерігали як казуальні, мідкорні та хардкорні ігри захоплювали мобільну екосистему. Розвиток мобільної індустрії буде продовжуватися, але в міру зростання та зміни ринку відеоігор, розробникам доведеться змінювати концепції. Тож вони мають постійно приймати рішення, які повинні одночасно задовольняти класичні характеристики ігрового процесу, і унікальність та простоту з цікавістю.

Якщо розглядати ігрову індустрію як глобальну промисловість, можна відзначити швидкий розвиток цього ринку, оцінкова вартість якого – понад 160 мільярдів доларів у доходах на 2020 рік. Більша частина такого здобутку, зазвичай, складається від продажу мобільних ігор, навіть не враховуючи комп'ютерні розробки, оскільки вони вважаються найшвидшим сектором. Великі цифри доходів – це показник для розробників ігор, тому вони і надалі зацікавлені у постійному створенні нових додатків, використанні нових «матеріалів» [2].

Першу мобільну гру розробили у 1994 році для Hagenuk MT-2000 – це був варіант версії однієї з найпопулярнішої гри «Tetris». Саме ця гра відзначила початок того, що стане згодом основою успішною та вигідною промисловістю [3, 4]. У грудні 1997 року компанія Nokia почала встановлення у своїх девайсах гри «Snake». Ця гра першопрохідець отримала величезну аудиторію на той час, а її варіанти та копії встановлені і знаходяться на майже 500 мільйонах пристроїв. У роках з 1999 по 2005 створився пік масштабно поширених ігор, у багато з яких грають, навіть, до тепер. Приклад такої гри – «Space Invaders». Одним з найважливіших моментів у історії розвитку мобільних ігор стала розробка N-Gage Phone у жовтні 2003 року компанією Nokia. Пристрій поєднав ігри з телефонними функціями. N-Gage дозволяв грати у безліч додатків, навіть 3D.

Кульмінаційним проривом у 2007 році на ринку стала поява iPhone. Смартфони мали покращені програмні та апаратні функції з кожним новим

поколінням. Поступово компанії та розробники віддаляються від ігрових консолей до розвитку мобільних ігор та ПК. Мобільні платформи не мають обмежень, пов'язаних із розробкою на консолі девайси. Вони більш дешевші та можуть бути розроблені без дозволу компанії. Із покращеннями Android та iPhone, почалось зменшення відставання якості розробки мобільних ігор від консольних. Коли «магазини» були вперше запуснені, вони пропонували до «500» мобільних додатків та ігор, найбільш відомими з яких були «Super Monkey Ball» та «Texas Hold'em». На початку 2010-х років було створено одні з найуспішніших і надзвичайно захопливих мобільних ігор ігор, зокрема «Angry Birds» у 2009 році, «Cut the Rope» у 2010 році, «Fruit Ninja» у 2010 році, «Plants vs. Zombies» у 2010 році, «Temple Run» у 2011 році, «Flappy Bird» у 2013 році, «2048» у 2014 році та багато іншого. У 2012 році була створена одна з найпопулярніших мобільних ігор, легендарна відеогра «З в'язи», у світі – «Candy Crush». Вона здатна утримувати увагу деякий час, при цьому розвиваючи реакцію, увагу та спостережливість. Окрім цього яскравість кольорів та графіка гри додають ефекту задоволення від гри. Зі збільшенням кількості випусків, багато розробників звертаються до нових технологій, щоб виділитися на фоні інших проектів. Наприклад, Pokemon Go — це мобільний додаток, основна функція якого — використовувати симулятор для відстеження головних героїв у реальному світі та в реальному часі. Іншими словами, гра проектує зображення мультиплікаційного героя через камеру пристрою. Наразі в магазині програм доступно багато типів мобільних ігор і різні способи опису таксономії мобільних ігор (іншими словами, групування). Завдяки унікальній розробці додатків з'явилася аудиторія, яка включає не тільки геймерів, а й звичайних споживачів. Більшість «мобільних» гравців є частиною цільової групи, яка не ідентифікує себе як професіонали, а належить до гіпер казуального жанру мобільних ігор. Зараз існує понад 6 мільйонів додатків. Це число включає: магазин Google Play – 2,57 мільйона додатків, App Store – 1,84 мільйона додатків, вікно Windows – 669 000 додатків, а також Amazon, у 2019 році пропозиція становила понад 489 000 додатків [5]. Стрімкий розвиток магазинів програмного забезпечення відбувається паралельно зі зростанням популярності мобільних ігор на світовому ринку. У зв'язку з постійним кількісним зростанням додатків і їх оновленнями більшість з перерахованого наразі не затребувані. Пошук правильної стратегії монетизації часто є основною метою для засвоєння гри користувачем [6, 7]. Ігри можна завантажити з магазину програм Google play або App Store і попередньо

завантажити на пристрій за замовчуванням. Ви також можете грати в свої ігри з хмари (віртуального сховища).

Згідно з дослідженнями [8], жінкам особливо подобаються логічні ігри та проходження рівнів, а серед чоловіків такі ігри цікавлять лише 30%. При цьому для чоловіків пріоритетним є підбір рівнів різної складності. Також існує умовна класифікація гравців за рівнем інтересу на казуальних (новачки), хардкорних (про-геймери) і середніх гравців (гравці, які не підходять до перших двох категорій і грають у кардинально різні ігри). Також в іграх існує поділ споживачів за платоспроможністю - тих, хто готовий платити, і тих, хто не готовий. Такий розвиток геймерської субкультури потребує розвитку самого ринку. Щоб охопити цю аудиторію, потрібно не лише розширити стратегії на набагато ширше населення, але й створити більше підгруп споживачів.

Глобальна епідемія коронавірусу стала однією з головних причин посилення розвитку гіпер казуальних ігор в першому кварталі 2020 року. Не секрет, що пандемія завдала шкоди як світовій економіці, так і суспільству – багато компаній навіть збанкрутували. Але розробники і видавці ігор від цього розбіжності тільки виграли - їхні прибутки значно зросли. Ізоляція змусила користувачів встановити рекорди завантажень для мобільних додатків. Більше того, виявилось, що навіть вдома люди не звертали уваги на доступ до ноутбуків, комп'ютерів чи ігрових консолей — вони все ще здебільшого грали на своїх телефонах.

1.2 Сучасна гіпер казуальна ігрова мобільна індустрія

У 2019 році на ринок гіперказуальних ігор припадало понад 45% усіх встановлень мобільних ігор із приблизно 7,8 мільярда завантажень. До 2020 року кількість завантажень гіперказуальних ігор зросла до 11,8 мільярда. У перерахунку на суму в доларах, за деякими оцінками, дохід від гіперказуальних ігор становить понад 3 мільярди доларів на рік.

Зростання категорії до домінування відбулося відносно швидко; ще в 2017 році гіперказуальні ігри становили 17% завантажень мобільних ігор, забезпечивши їм третє місце після головоломок (15%) і аркадних ігор (19%). Отже, як гіперказуальні ігри так швидко стали такими популярними?

Зміна демографічних показників має багато спільного з цим. Сучасні геймери не відповідають молодим чоловічим стереотипам, які колись домінували на сцені: одна третина гравців мобільних ігор старше 45 років, а більше половини — жінки. Зі зміною демографічних показників геймерів змінювалися і моделі поведінки геймерів; замість того, щоб сидіти перед екраном, щоб грати в гру цілий день, більшість мобільних геймерів сьогодні вписують свій час у багатозадачність або, наприклад, в очікуванні початку іншої діяльності.

Гіперказуальні ігри добре піддаються цим коротким сплескам активності, оскільки їх розроблено так, щоб їх було якомога легше підібрати та одразу грати. Для розробників гіперказуальні ігри також легко створювати та запускати завдяки їх легкому дизайну. Студії мобільних ігор часто співпрацюють із видавцями програм, щоб вивести на ринок свої гіперказуальні ігри. Серед провідних видавців у просторі гіперказуальних ігор:

- Voodoo: понад 5 мільярдів завантажень
- Crazy Labs: понад 4,5 мільярда завантажень
- SayGames: понад 2 мільярди завантажень
- Ketchapp: понад 800 000 ігор
- Lion Studios: 13 ігор №1
- Green Panda: понад 50 мільйонів користувачів

Маємо наступну вікову статистику за 2022 рік:

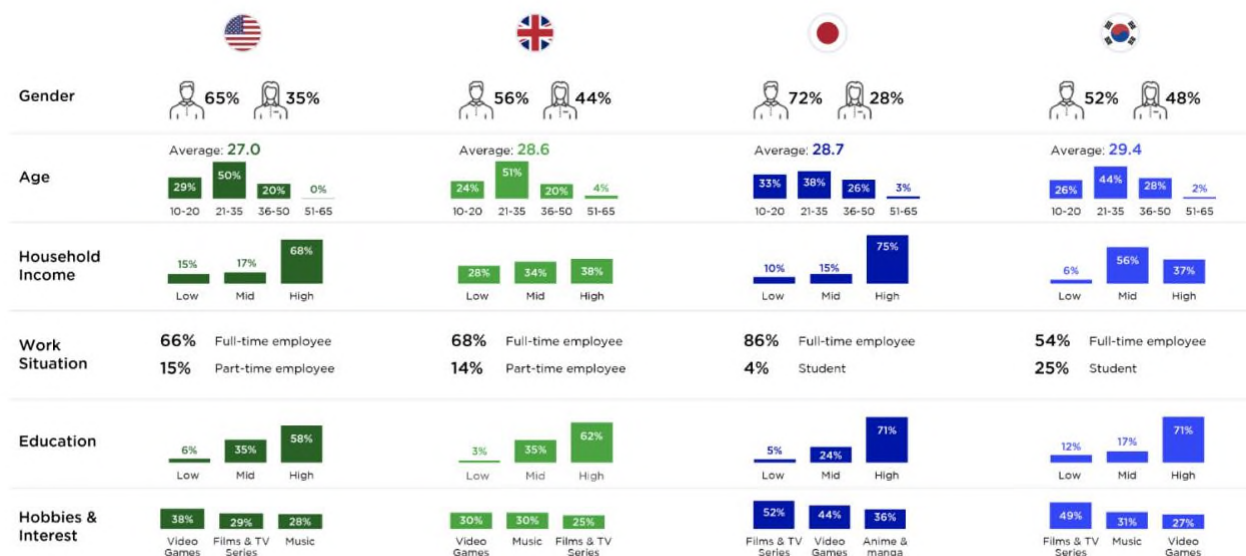


Рисунок 1.1. – Вікова статистика

			
1 Battle Royale	1 Arcade	1 Role-Playing	1 Role-Playing
2 Puzzle	2 Simulation	2 Strategy	2 Arcade
3 Sports	3 Strategy	3 Arcade	3 Simulation
4 Shooter	4 Fighting	4 Simulation	4 Strategy
5 Simulation	5 Sports	5 Battle Royale	5 Adventure

Рисунок 1.2. – Найпопулярніші жанри в регіонах

Гіперказуальні гравці на всіх чотирьох ринках молодші за всіх мобільних геймерів. У той час як середній вік мобільних геймерів на цих чотирьох ринках становить близько 33 років, середній вік гіперказуальних геймерів нижче 30 років.

Наведена сучасна статистика ігрової індустрії:

–За даними Sensor Tower, в третьому кварталі 2019 року кількість видавців ігрових проектів в Apple Store і Google Play склала 108 000 з 793 000, що становить ні багато ні мало 15 відсотків.

–App Annie прогнозує, що в 2022 році споживачі будуть разом витратити 674 млрд. годин на мобільні пристрої в порівнянні з 558 млрд. в 2019 р

–Сьогодні на мобільні ігри припадає 33% всіх завантажень програм, 74% витрат, пов'язаних з ігровими процесами і 10% всього часу, проведеного в додатку.

–Більше 50% користувачів мобільних додатків грають в ігри, що робить цю категорію додатків такою ж популярною, як музичні програми, такі як Spotify і Apple Music, і поступається тільки додаткам для соціальних мереж і комунікацій з точки зору витраченого часу.

Консолідація завантажень і тиранія старих ігор триватимуть: оскільки витрати на залучення користувачів, здається, не зменшаться жодним чином, успішні гіперказуальні ігри з уже сформованою базою користувачів продовжуватимуть змінювати масштаб і захоплювати більшість завантажень протягом усього періоду. 2022 рік.

Короткострокові ігри, які слідують тенденціям поп-культури, можна буде бачити частіше: все більша частина нових гіперказуальних ігор, які досягають певного рівня успіху, завдячують своїм успіхом дотриманню тенденцій у соціальних мережах і телешоу, а не оригінальній механіці чи інноваціям у ігровому процесі.

Розмах злиттів і поглинань збільшиться: оскільки все більше видавців приєднуються до тенденції придбання розробників, ті, які ще не зробили жодного кроку, братимуть участь у цьому, і ми побачимо, що більше розробників Hypercasual будуть частиною видавців, які колись були їхніми партнерами.

Диверсифікація портфоліо буде важливою частиною майбутньої стратегії, і буде опубліковано більше казуальних ігор-головоломок: Zen Match є хорошим прикладом для решти видавців Hypercasual, які продемонструють свій ефект із збільшенням кількості ігор у 2022 році.

Newzoo очікує, що світовий ринок ігор у 2023 році принесе 204,5 мільярда доларів, що на 4,3% менше, ніж у минулому році, і вперше прогнозує зменшення за всю історію охоплення ринку.

Раніше в цьому році компанія з обробки даних оцінила, що ринок досягне \$196,8 млрд у 2022 році, тому цей новий прогноз був різко знижений.

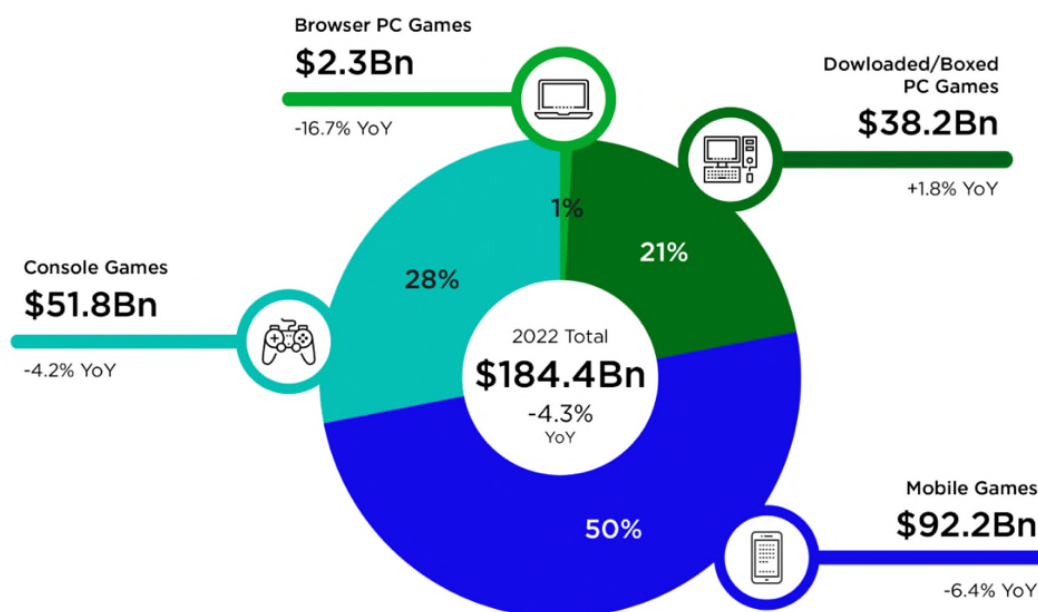


Рисунок 1.3. – Кількість млрд доларів прибутку від платформи

У Newzoo заявили, що падіння буде пов'язане з низкою факторів, зокрема «повільнішим, ніж очікувалося, роком для консолі» та тим, що 2022 рік став «виправним» після двох років зростання через пандемію COVID-19.

У компанії додали, що її прогноз все ще вищий, ніж очікувалося до пандемії, і наголошують на тому, що лише 2022 рік буде коригуючим.

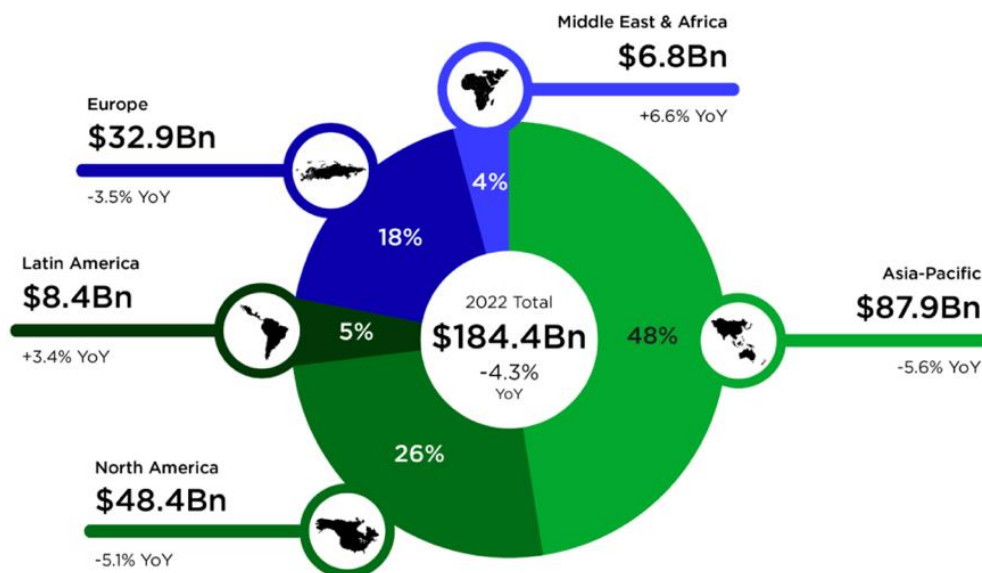


Рисунок 1.4. – Прибуток по регіонах

«Наш довгостроковий прогноз щодо ігрового ринку залишається позитивним», — йдеться у звіті. «Хоча це може здатися невдачею для ринку ігор, ми зазначаємо, що сума доходів, отриманих з 2020 по 2022 роки, майже на 43 мільярди доларів перевищує наше початкове прогнозування до пандемії».

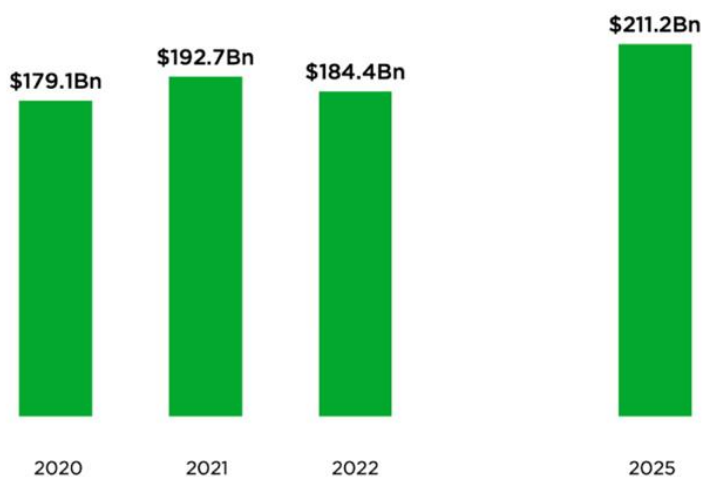


Рисунок 1.5. – Прогноз прибутку на 2025 рік

1.3 Гіпер казуальні ігри, як казуальний піджанр

Гіпер-казуальні ігри – відносно, молодий жанр в іграх, який виділяється своєю елементарною механікою, яка проста, і зрозуміла, і не потребує первинного навчання. Гіпер казуальний жанр має дуже широку аудиторію, більшість гравців мають вікову категорію від 8 до 50 років. Цей піджанр казуальних ігор виділяється своєю зрозумілістю, адже більшість людей завантажують ці ігри через рекламу, так як бачать весь цілковитий та зрозумілий їм геймплей [1]. З плином часу, як казуальні ігри використовують глибші та мідкорні функції, вони стають цікавішими та набирають аудиторію, але їх складність проходження тільки збільшується. У свою чергу, такий шлях казуальних ігор дає можливість гіпер казуальним іграм своєю простотою збільшувати свою ігрову аудиторію.

Із-за мінімалізму і простоти розробники витрачають на первинну розробку від кількох тижнів до кількох місяців, замість декількох років розробки AAA-проектів. Це один з елементів стратегії, яка дає можливість гіпер-казуальним іграм бути успішними та швидко оновлюватись. Основні видавці гіпер-казуальних ігор, такі як: Voodoo, Appsolute Games, Ketchapp – випускають, приблизно, одну гру на один-два тижні. Успіх гіпер казуальних ігор більшу частину залежить від кампаній по залученню користувачів (User Acquisition) в соцмережах. Найбільше весь трафік або лідів (користувачів соціальних мереж) припадає на відеоролики, довжиною до 30 секунд, між різними постами у соціальних мережах.

Одна з головних цілей – швидко зацікавити та пояснити потенційному гравцеві чому потрібно встановити гру. Більшість користувачів використовують смартфони, приблизно 97%, тому потрібен маленький розмір файлу, задля того, щоб користувач швидко скачав та встановив гру.

Після встановлення використовується поняття онбордингу. Онбординг — це перше враження гравця і його реакція. Потенційному гравцеві у гру потрібно приблизно 7 секунд, щоб вирішити чи зацікавила гра і чи варто її залишити. Такий короткий відрізок часу означає, що не потрібні довгий тьюторіал і багатоекранний UI, який займає цей час. В гіперказуалках важливо, щоб гравець одразу приступив до ігрового процесу за для задоволення мозку та економії часу. Через швидкий запуск та короткі ігрові сесії, гіпераказуальні ігри є ідеальними тайм-кіллерами в повсякденному житті. Ігрова сесія або рівень не повинна

тривати більше 3-5 хвилин. Цього проміжку достатньо, щоб зайняти гравця між його рутинними справами. Швидкий вихід і вхід в ігровий сеанс — це один з важливих аспектів, який допомагає досягти успіху.

Мозок гравця потребує швидкого реагування на сам процес гри. В гіперказуальних іграх обов'язковий короткий цикл зворотнього зв'язку — гравець грає коротку сесію або рівень, має можливість як перемогти і поставити рекорд, або програти і отримати ще один шанс і покращити свої навички, щоб пройти ще раз рівень, а в кінці збирати нагороди. Про успішні та невдалі дії повідомляється миттєво.

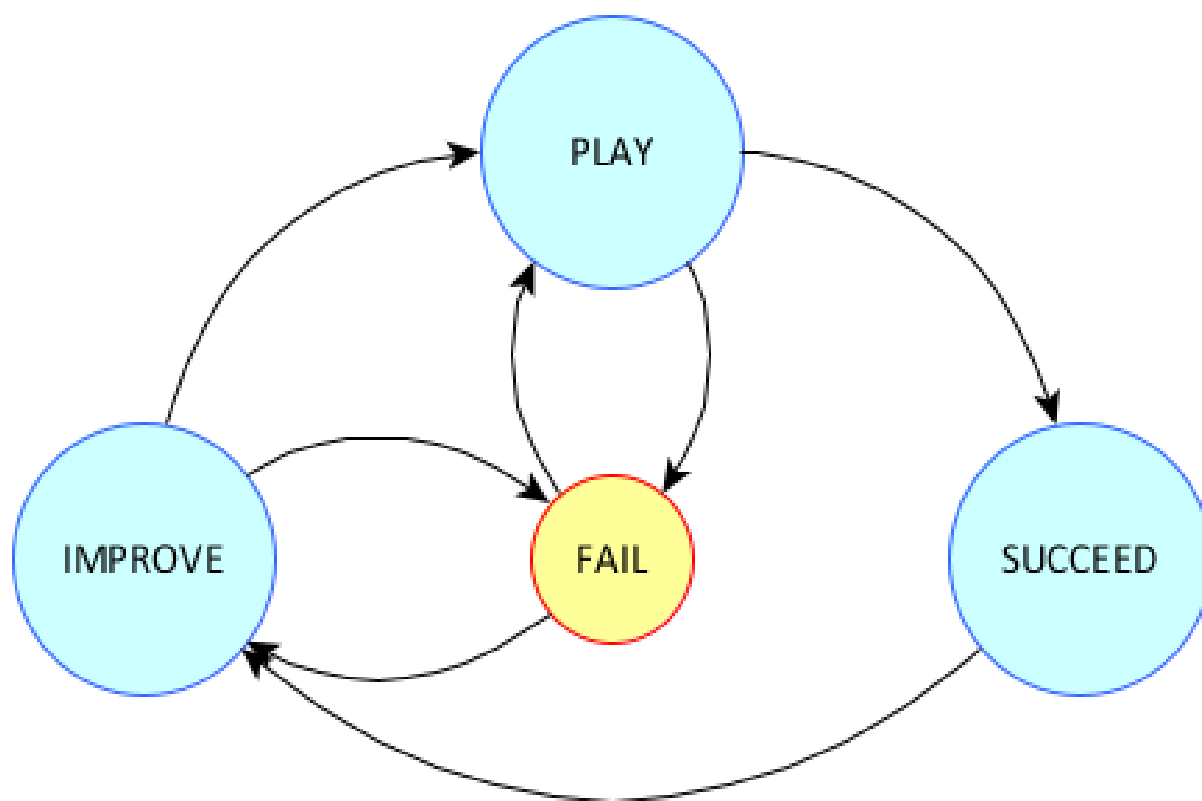


Рисунок 1.6. Короткий цикл зворотного зв'язку — гравець покращує навички, якщо програє

Кор-геймплей — це основна механіка, яка задіяна та представлена в грі. Він простий, викликає звикання та бажання повернутися в гру — це основна задача кор-геймплею. Існує безліч різних реалізацій кор-геймплею.

Також, існує мета-геймплей. Мета-геймплей — це додатковий закритий функціонал, який затримує та змушує гравця відкрити його з проходженням гри. Це різні варіанти кастомізації та покращень.

Якщо підсумувати ключові аспекти жанру:

- Механіка, яка буде одразу зрозуміла гравцю.
- Задоволення гравця, яке він отримує під час ігрових сесій.
- Короткі цикли зворотного зв'язку.
- Легкість входу і виходу.
- Відчуття потужності/покращень у формі прогресії.
- Простий мета-геймплей для підтримки основного ігрового процесу.

У період карантинного «застою» гіпер казуальні ігри користувалися попитом на ринку, тому що вони були настільки прості, що їх було легко освоїти і користувачі поверталися до гри знову і знову. Вони не мають сюжетного навантаження, що дозволяє багаторазові проходи. Гіперказуальні ігри стали популярними після початку карантину в березні 2020 року: завантаження зросли на 72% у всьому світі та на 300% у Китаї. У квітні та травні вони ще знаходились на піку мейнстріму, але до кінця літа темпи зростання вже починають сповільнюватися.

Гравці бувають різних категорій: хардкорні, середні, звичайні та неігрові. Запеклі геймери знають про гру все. Більшу частину вільного часу вони проводять за іграми, досконально вивчаючи всю інформацію про гру або обрану гру. Вони часто є кіберспортсменами. Гравці середнього рівня проводять у грі менше часу, ніж хардкорні гравці, але вони все одно знають більшість ігрових механік. Казуальним гравцям потрібна лише година на день, щоб грати в гру. Вони не вдаються в подробиці гри. При цьому ці категорії становлять лише невеликий відсоток населення всієї планети. Гіпер казуальні проекти почали націлюватися на нон-геймерів, переважно на людей, які ніколи не грали в ігри. Ми знаємо, що стандартні категорії гравців мають вбудовані моделі поведінки, які проектується на базові механіки, які вони використовують. У той же час у нон-геймерів абсолютно відсутні. В той же проект намагаються перенести шаблони з відомих додатків, наприклад WhatsApp, Tinder або Instagram. Таким чином, одне із завдань розробників ГКІ — створити ігровий процес, який відповідає шаблонам, зрозумілим нон-геймерам. Важливо відзначити, що аудиторія нон-геймерів є найбільшою аудиторією, тому саме вони популяризують продукт.

Відео процесу гри має дати нон-геймерам повне розуміння того, що відбувається в грі, і привернути його чи її увагу. У випадку ГКІ кожен може створити аудиторію для різних проектів, і це не помилка позиціонування.

Простий та інтуїтивно зрозумілий інтерфейс дозволяє ГКІ зосередити увагу на розв'язанні конкретного завдання. Складання ГКІ займає від одного до двох місяців і не є дуже складним.

1.4 Патерни проектування

Патерн проектування — це загальне рішення конкретної проблеми, яка часто зустрічається при проектуванні архітектури програмного забезпечення.

На відміну від готових функцій або бібліотек, патерни не можна просто взяти та скопіювати в програму. Патерн — це не конкретний код, а загальний спосіб вирішення проблеми, який майже завжди потрібно налаштувати відповідно до потреб конкретної програми.

Патерни часто плутають з алгоритмами, оскільки обидві концепції описують загальні рішення відомих проблем. Але якщо алгоритм — це чіткий набір дій, то шаблон — це високорівневий опис рішення, реалізація якого може відрізнитися у двох різних програмах.

Опис шаблону зазвичай формальний і часто складається з наступних тверджень.

- завдання, яке вирішує патерн;
- мотивація розв'язувати задачі за патерном;
- структура класів, складові рішення;
- приклад мовою програмування;
- особливості реалізації в різних областях;
- відношення до інших патернів.

Цей описовий формалізм дозволяє накопичити великий каталог патернів, а також перевірити, чи є кожен патерн дієвим.

Поняття патернів вперше було описано Крістофером Александром у книзі «Мова шаблонів. Міста. Будівлі. Будівництво». У цій книзі йдеться про «мову» дизайну навколишнього середовища, одиницями якої є шаблони (або патерни, ближче до оригінального терміну патерни), які відповідають на архітектурні запитання: наскільки вікна повинні бути високими, скільки поверхів повинна мати будівля, яку частину мікрорайону слід відвести під склад.

Ідею придумали чотири письменники: Еріх Гаммі, Річард Хелм, Ральф Джонсон, Джон Вліссайдс. У 1994 році вони опублікували книгу під назвою *Design Patterns: Reusable Elements of Object-Oriented Software Architecture*, яка містила 23 патерни для вирішення проблем об'єктно-орієнтованого

проектування. Назва книги була надто довгою, щоб її можна було запам'ятати. Тому незабаром усі стали називати її «book by the gang of four», тобто «книга від банди чотирьох», а потім і зовсім «GoF book».

Відтоді було виявлено багато інших моделей патернів. Підхід «патерни» також став популярним у програмуванні, тож тепер ви можете бачити патерни за межами об'єктно-орієнтованого проектування.

Патерни відрізняються за рівнем складності, деталізації та охоплення розробленої системи. Використовуючи аналогію з будівництвом, можна підвищити безпеку, встановивши світлофор на перехресті вулиці, або замінити перехрестя з цілою розв'язкою на підземний перехід.

Найнижчими і найпростішими зразками є ідіоми. Вони не дуже універсальні, оскільки мають сенс лише в межах мови програмування.

Найбільш універсальними є архітектурні патерни, які можна використовувати будь-якою мовою. Вони потрібні для оформлення всієї програми, а не окремих елементів.

Причому патерни відрізняються за призначенням. Патерни зазвичай ділять на три категорії:

- Породжуючі патерни - відображають гнучке створення об'єктів без внесення в програму зайвих залежностей.

- Структурні патерни - відображають різні способи побудови зв'язків між об'єктами.

- Поведінкові патерни - відображають ефективну комунікацію між об'єктами.

Патерни проектування — це незалежні від мови програмування стратегії вирішення типових проблем об'єктно-орієнтованого проектування. Це означає, що патерн проектування представляє ідею, а не конкретну реалізацію.

Використовуючи патерни проектування, код створюється та реалізується більш гнучким, зручним для багаторазового використання та обслуговуванням. Це найважливіша частина, оскільки розробка програм внутрішньо слідує патернам проектування.

Щоб стати професійним розробником програмного забезпечення, рекомендують знати принаймні деякі популярні рішення (тобто патерни проектування) проблем кодування.

Перевага патернів проектування:

- Їх можна повторно використовувати в кількох проектах.

–Вони пропонують рішення, які допомагають визначити архітектуру системи.

–Вони фіксують досвід розробки програмного забезпечення.

–Вони забезпечують прозорість дизайну програми.

–Це добре перевірені та підтверджені рішення, оскільки вони створені на основі знань і досвіду експертів-розробників програмного забезпечення.

Патерни проектування не гарантують абсолютного вирішення проблеми. Вони забезпечують ясність архітектури системи та можливість побудови кращої системи.

2 РОЗРОБКА ВИМОГ ДО ІГРОВОГО ДОДАТКУ

2.1 Патерн проектування Singleton

Singleton(Синглтон) — патерн, що породжує, який гарантує, що для певного класу буде створено тільки один об'єкт, а також надасть до цього об'єкта точку доступу.

Синглтон дозволяє створити об'єкт лише за його необхідності. Якщо об'єкт не потрібен, він не буде створений. У цьому відмінність синглтона від глобальних змінних.

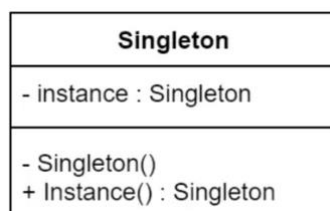


Рисунок 2.1 – Синглтон в UML

У класі визначається статична змінна - посилання на конкретний екземпляр даного об'єкта та приватний конструктор. У статичному методі `getInstance()` цей конструктор викликається для створення об'єкта, якщо, звичайно, об'єкт відсутній і дорівнює `null`.

Визначення об'єкта синглтона у вигляді статичного поля класу відкриває нам шлях до створення Lazy-реалізації патерна Синглтон, тобто такої реалізації, де дані ініціалізуюватимуться лише перед безпосереднім використанням. Оскільки статичні поля ініціалізуються перед першим доступом до статичного членів класу та перед викликом статичного конструктора (за його наявності). Але при такій реалізації є дві проблеми.

По-перше, клас синглтона може мати безліч статичних змінних.

По-друге, статичні поля ініціалізуються перед першим доступом до статичного членів класу та перед викликом статичного конструктора (за його наявності). Якщо клас містить статичні поля, не містить статичного конструктора, час ініціалізації статичних полів залежить від реалізації платформи. Нерідко це безпосередньо перед першим використанням, проте момент точно не визначений - це може бути відбуватися і трохи раніше. Однак якщо клас містить статичний

конструктор, то статичні поля ініціалізуватимуться безпосередньо або при створенні першого екземпляра класу, або при першому зверненні до статичних членів класу.

2.2 Патерн проектування Abstract Factory

Патерн «Абстрактна фабрика» (Abstract Factory) надає інтерфейс створення сімейств взаємозалежних об'єктів з певними інтерфейсами без зазначення конкретних типів даних об'єктів.

Коли використовувати абстрактну фабрику:

- Коли система не повинна залежати від способу створення та компонування нових об'єктів
- Коли об'єкти, що створюються, повинні використовуватися разом і є взаємопов'язаними

За допомогою UML абстрактну фабрику можна представити так:

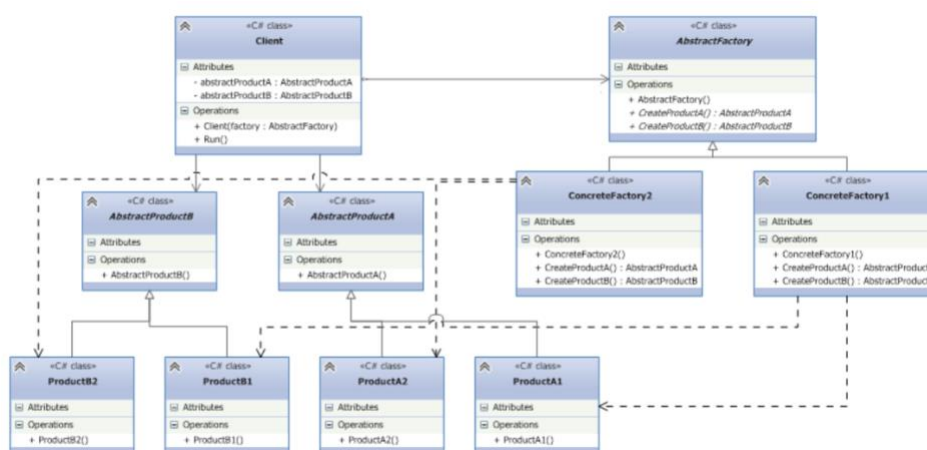


Рисунок 2.2 – Абстрактна фабрика в UML

Паттерн визначає наступних учасників:

- Абстрактні класи AbstractProductA та AbstractProductB визначають інтерфейс для класів, об'єкти яких створюватимуться у програмі.
- Конкретні класи ProductA1/ProductA2 та ProductB1/ProductB2 представляють конкретну реалізацію абстрактних класів.

- Абстрактний клас фабрики `AbstractFactory` визначає методи створення об'єктів. Причому методи повертають абстрактні продукти, а чи не їх конкретні реалізації.
- Конкретні класи фабрик `ConcreteFactory1` та `ConcreteFactory2` реалізують абстрактні методи базового класу та безпосередньо визначають які конкретні продукти використовувати.
- Клас клієнта `Client` використовує клас заводу для створення об'єктів. При цьому він використовує виключно абстрактний клас фабрики `AbstractFactory` та абстрактні класи продуктів `AbstractProductA` та `AbstractProductB` і ніяк не залежить від їх конкретних реалізацій.

Таким чином, створення продукту абстрагується від класу продукту. У той же час не можна не відзначити недоліки шаблону. Зокрема, якщо нам захочеться додати в конфігурацію продукту новий об'єкт, наприклад, тип одягу, доведеться переробляти класи фабрик і клас продукту. Тому можливості розширення в даному патерні мають деякі обмеження.

2.3 Функціональні вимоги до процесу розробки ГКІ

Щоб визначити функціональні вимоги до ІС, було вирішено розробити функціональну модель з використанням стандарту IDEF0. Метою цього моделювання є визначення та уточнення функціональних вимог до розробки ігрового додатку в жанрі гіпер казуальних ігор.

Процес проектування ігрового додатку доцільно розпочати з побудови контекстної діаграми A-0, яка відображена на рисунку 2.1. Така діаграма зможе дати стисле, а також доцільне пояснення кожного кроку реалізації, що дає змогу оцінити правильність обраного шляху виконання. Головними елементами даної діаграми є: вхідні дані, вихідні дані, управління та механізми. Зазвичай вказують точку зору (виконавця робіт), предметну область та ціль виконання.

Після проведеного аналізу відносно головних елементів контекстної діаграми «Створення ігрового додатка», було сформовано перелік даних:

- вхідні дані: завдання на створення ігрового додатку;
- вихідні дані: готовий ігровий додаток на різних ОС;
- управління: планування розробки, вимоги до гри;
- механізми: програмні засоби, розробник, апаратне забезпечення.

Контекстна діаграма А-0, що була розроблена за допомогою програмного забезпечення AllFusion Process Modeler .

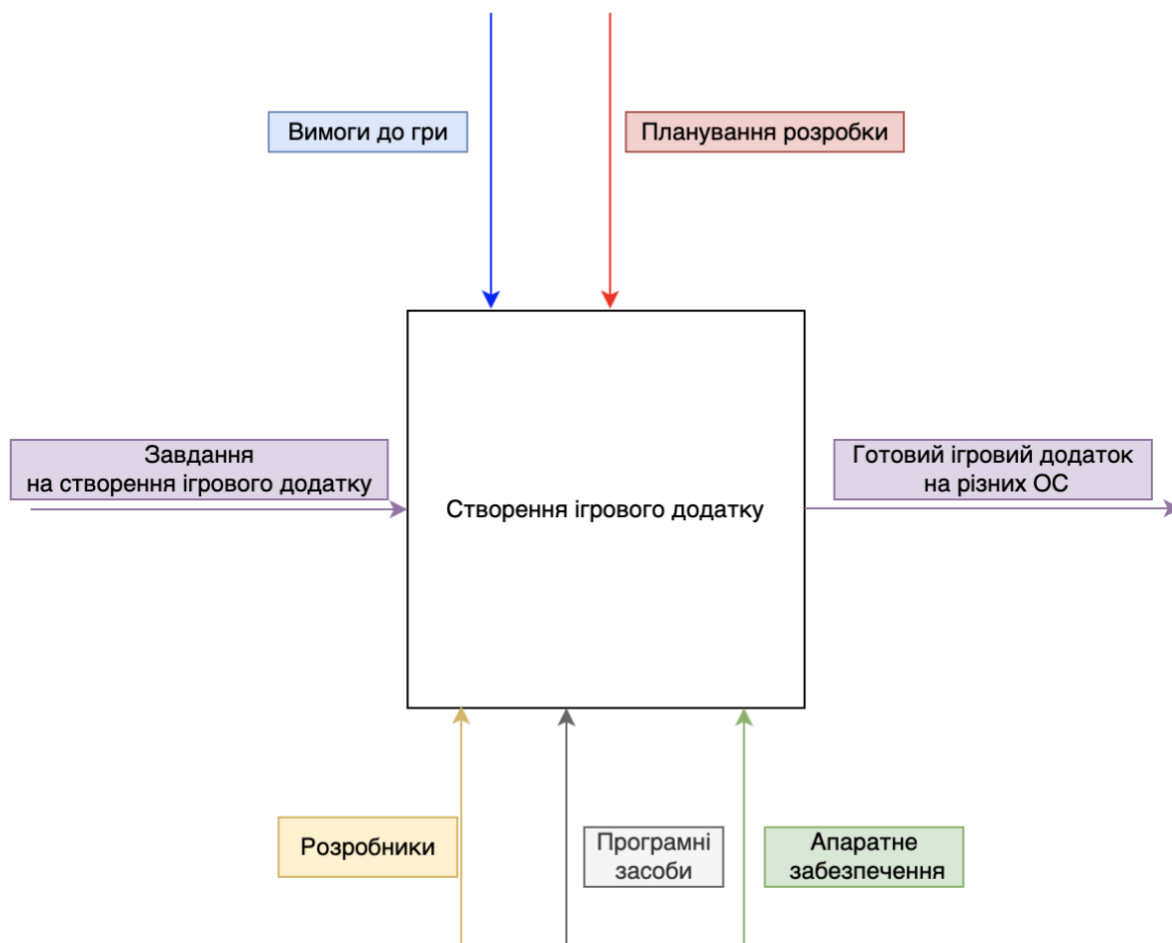


Рисунок 2.3 – Контекстна діаграма А-0

Оскільки така діаграма містить у собі лише загальну інформацію щодо системи, то виникає необхідність у виконанні декомпозиції, тобто розподілу цього об'єкту, системи чи задачі на складові елементи або частини.

Даний процес дозволить детальніше розглянути логіку послідовності виконання робіт для реалізації запланованого продукту.

Для уточнення структури та послідовності дій, зробимо декомпозицію функції «Створення ігрового додатку».

Діаграма А-0 була декомпована на чотири функції, а саме: створення головних скриптів ігрового процесу, налаштування візуальної частини, створення рівнів гри і тестування ігрового додатку. Це обумовлено тим, що процес створення ігрового додатку поділяється на чотири етапи:

- перший етап: створення головних скриптів ігрового процесу;
- другий етап: налаштування візуальної частини;

- третій етап: створення та заготовка рівнів гри;
- четвертий етап: процес тестування гри та процес відладки, якщо були знайдені якісь похибки в розробці та баги.

Всі чотири елементи діаграми А0 керуються «Вимогами до гри» та «Плануванням розробки», а також мають такі механізми виконання, як: «Програмні засоби», «Розробник» та «Апаратне забезпечення».

Вигляд після декомпозиції діаграми А-0 на 4 етапи наведений на рисунку 2.4

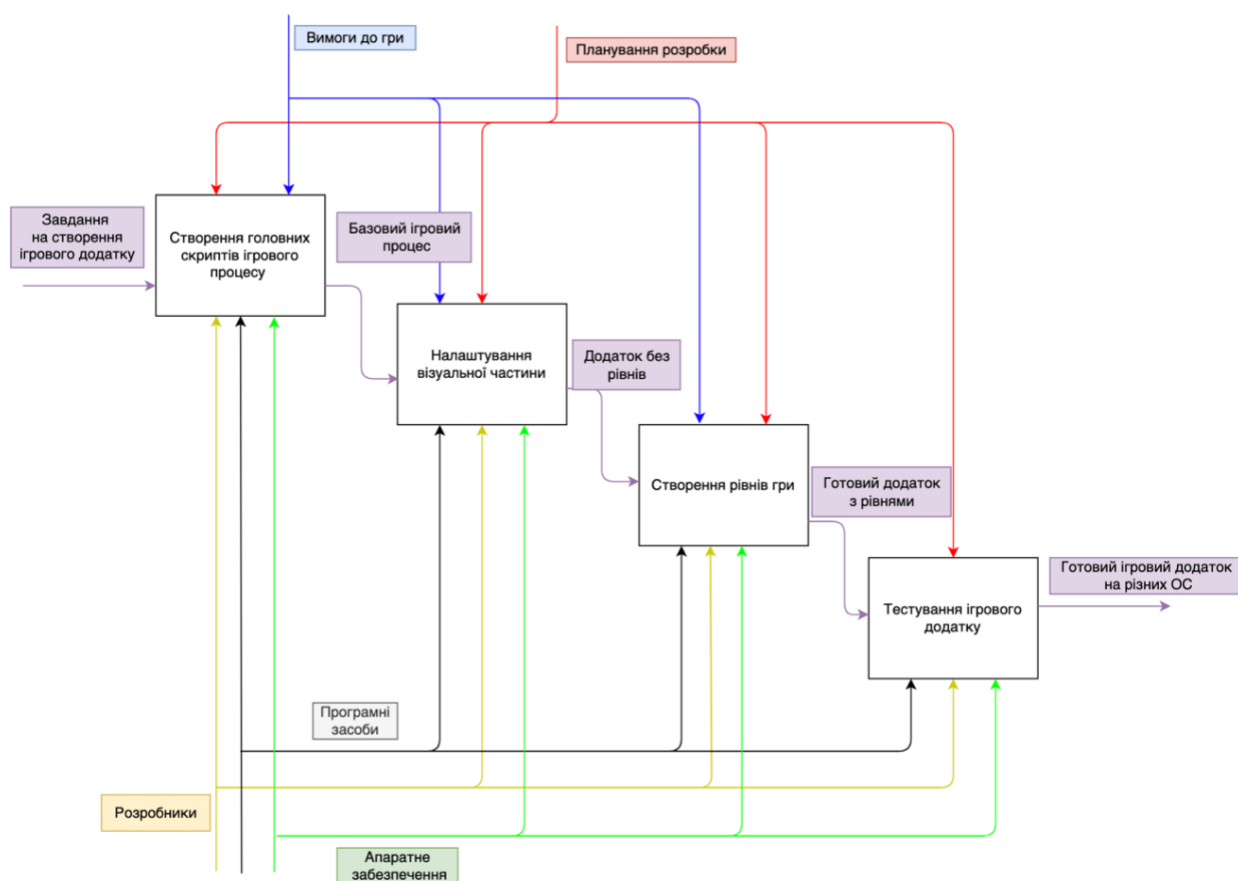


Рисунок 2.4 – Декомпозиція контекстної діаграми А-0

Кожен з рівнів діаграми А0 був декомпозований ще на декілька підрівнів.

Бізнес функція починається з функції «Створення головних скриптів ігрового процесу», адже для початку нам потрібно розробити скрипти, які будуть відповідати за сам процес. Після всього процесу розробки відбувається тестування, щоб була можливість видалити або виправити баги, створенні під час розробки в ігровому додатку.

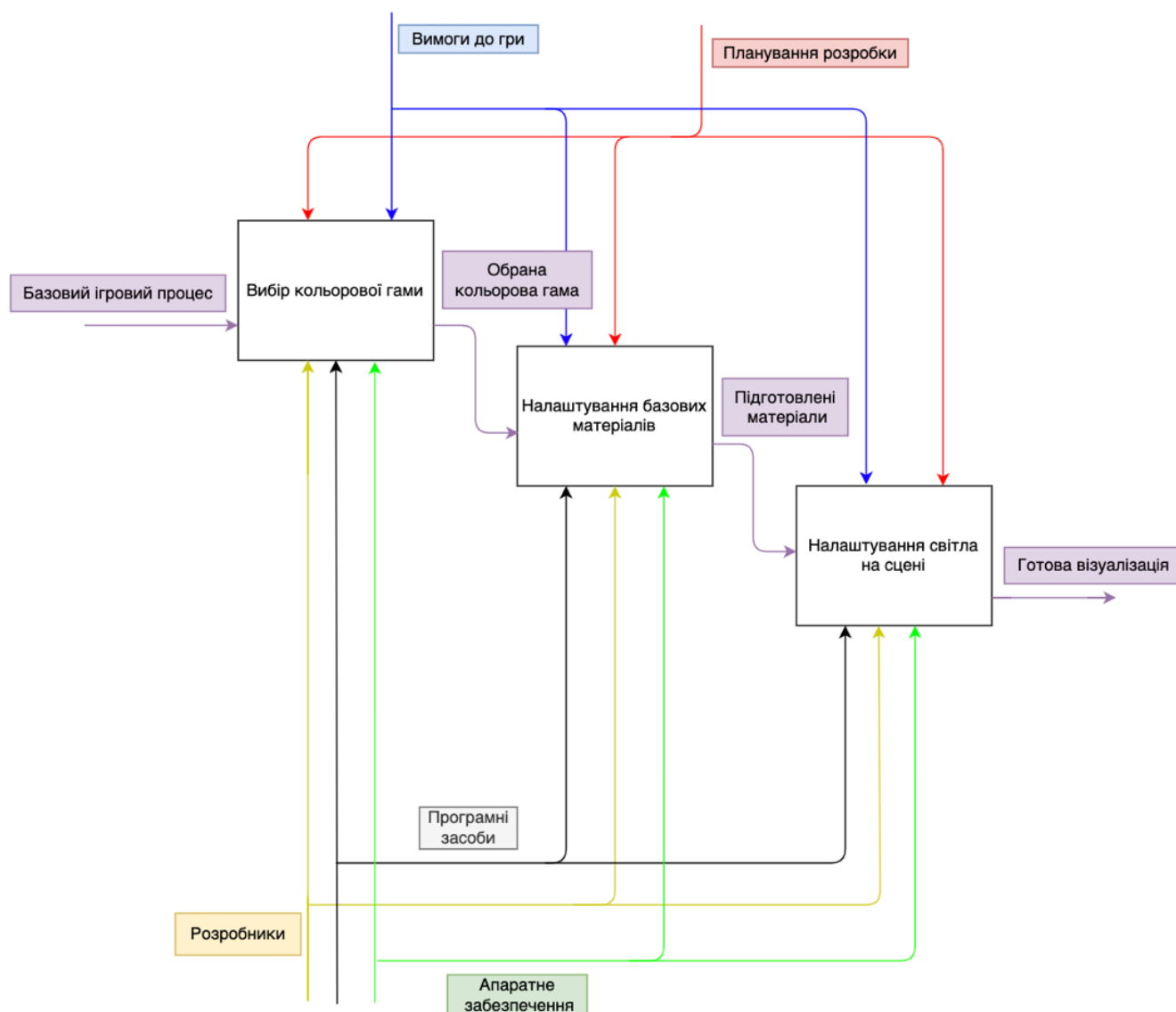


Рисунок 2.6 – Декомпозиція контекстної діаграми A2

Потім починається «Створення рівнів гри». Вхід отримує готову візуалізацію. Готова візуалізація дає змогу швидко створювати подібні рівні. Створивши рівні, ми потім заготуємо префаби для того, щоб використовувати їх в розробці.

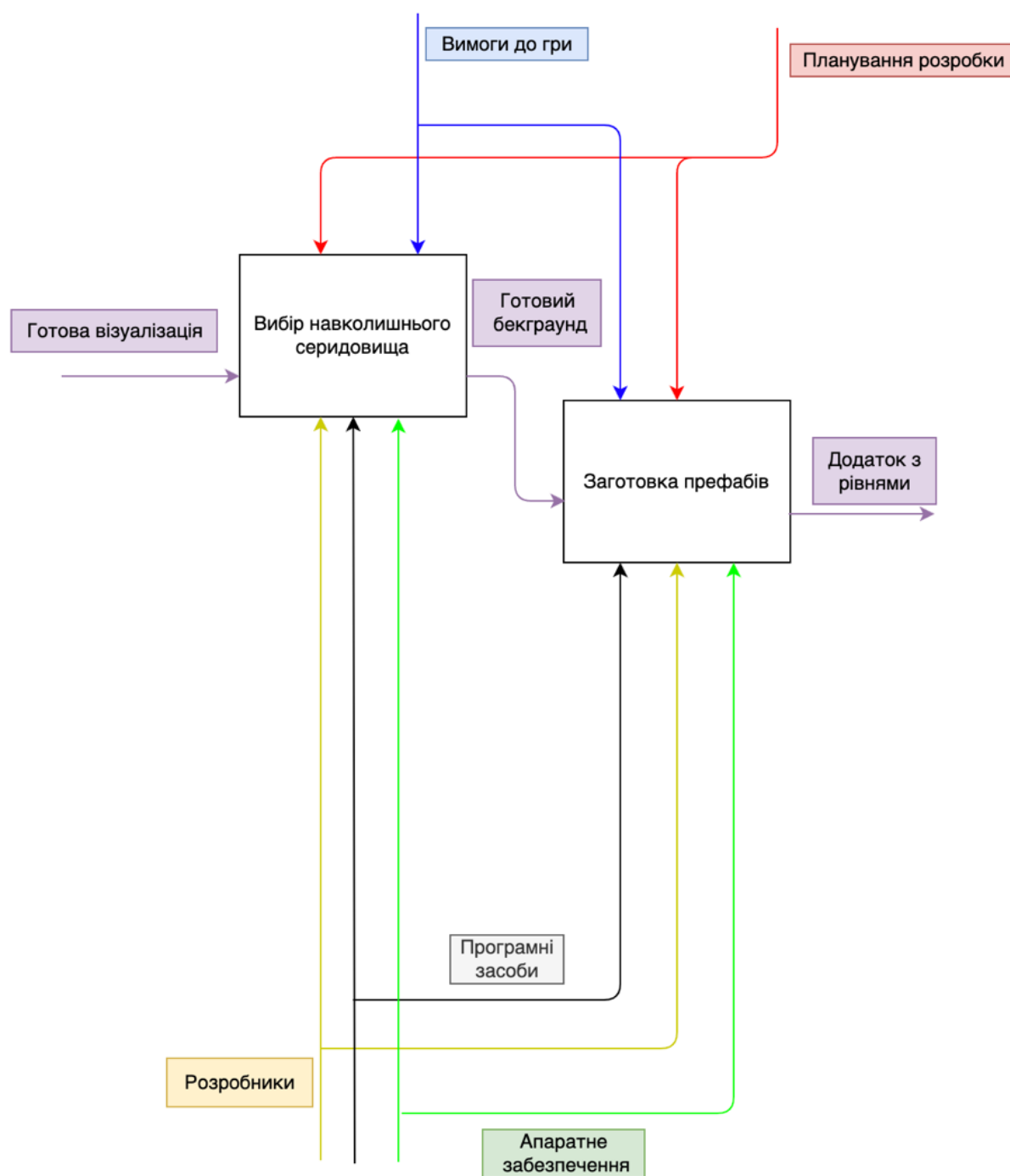


Рисунок 2.7 – Декомпозиція контекстної діаграми АЗ

Далі починається «Тестування ігрового додатку», що отримує на вхід додаток з рівнями, а на виході видає готовий ігровий додаток на різних операційних системах. Спочатку потрібно провести первинне тестування розробником, потім йде виправлення багаїв і вже повторне тестування.

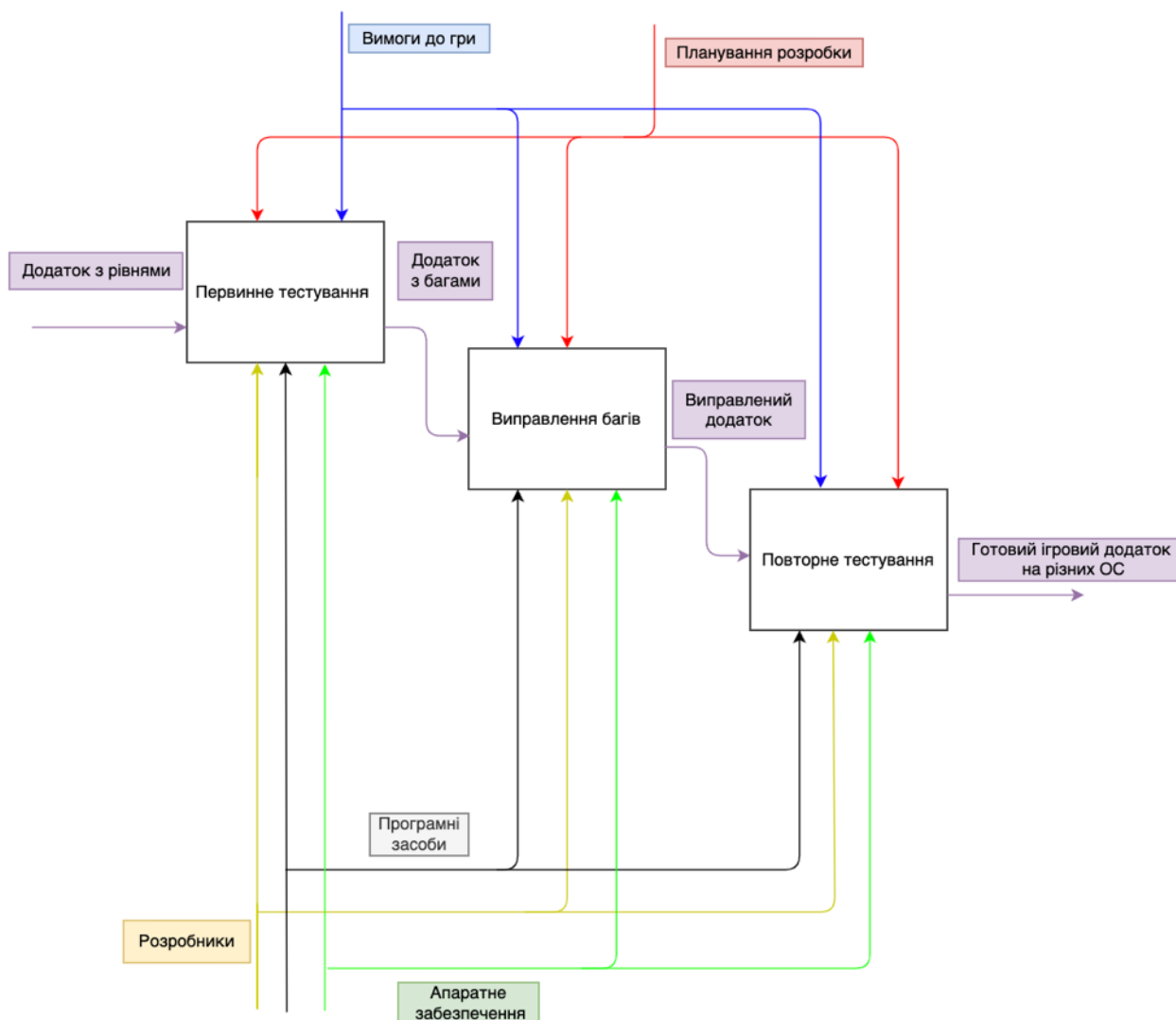


Рисунок 2.8 – Декомпозиція контекстної діаграми А4

Таким чином була створена функціональна модель з використанням стандарту IDEF0. Дана модель визначає функціональні вимоги до створення ігрового додатку в жанрі гіпер казуальних ігор.

2.3 Розробка вимог до інтерфейсу(діаграма класів, станів, прецедентів, діяльності)

Для побудови ігрового додатку до нього формуються вимоги щодо умов виконання функцій і обробки даних.

Найпоширенішим інструментом розробки вимог є UML, у якому вимоги визначаються та специфікуються через представлення можливих варіантів

використання програмної системи (use case), які трансформуються в проектні рішення та архітектуру системи.

UML представляє стандарти, що використовуються в інших галузях, і підтримує декілька типів діаграм. Діаграми UML описують межі, структуру та поведінку як усієї системи, так і окремих об'єктів у її складі.

UML не є мовою програмування, але діаграми UML можна використовувати для створення коду на різних мовах, і для цього існує ряд спеціальних інструментів. Однак уніфікована мова моделювання безпосередньо пов'язана з об'єктно-орієнтованим аналізом і проектуванням.

Діаграма прецедентів (випадків використання) — це граф, що складається з набору акторів, випадків, асоціацій між акторами та випадками, зв'язків між випадками та зв'язків узагальнення між акторами.

За допомогою прецедентів описується функціонування ігрового додатку з точки зору користувача, якого в UML називають актором. Актор представляє собою будь-яку сутність, зовнішню по відношенню до модельованої системи (особу, програмну систему, пристрій), яка взаємодіє з системою та використовує її функціональність для досягнення певної мети або вирішення приватних завдань [6].

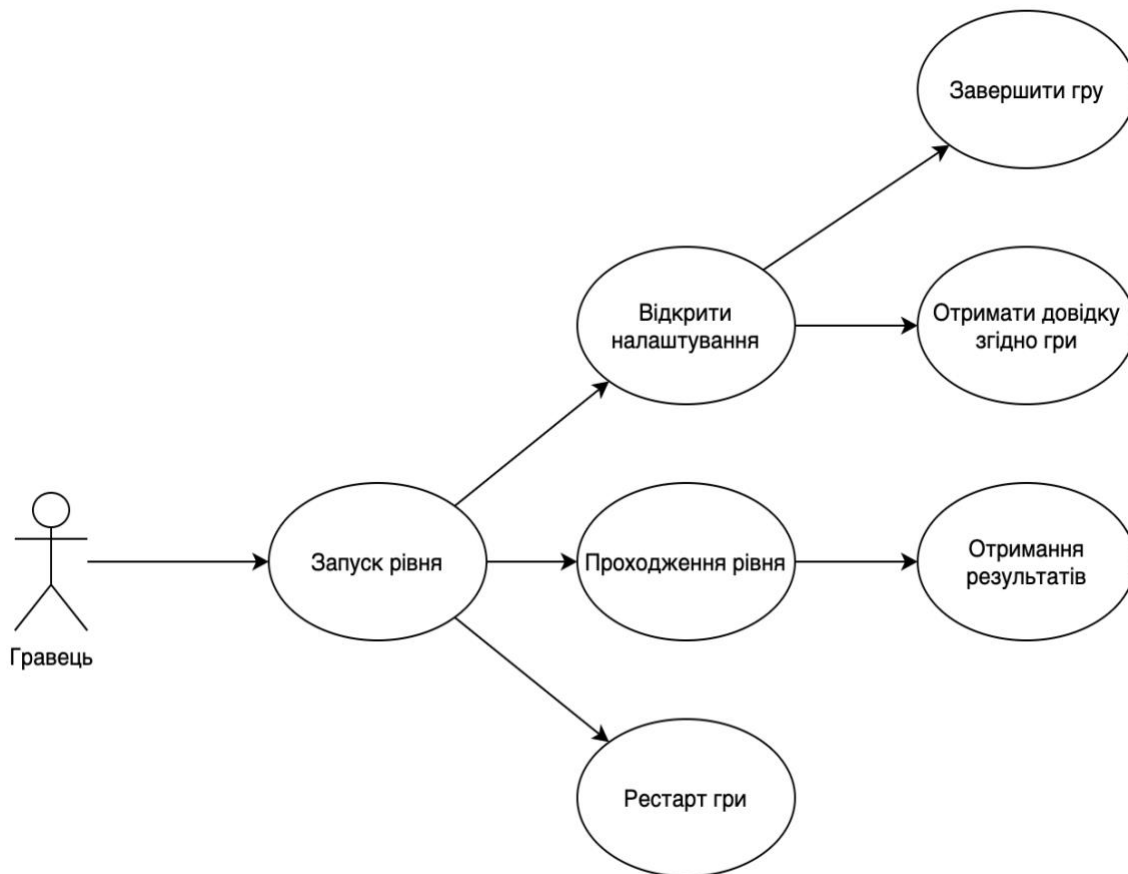


Рисунок 2.9 – Діаграма прецедентів

Діаграма класів є найпоширенішим типом діаграми UML і фундаментальною основою будь-якого об'єктно-орієнтованого рішення. Діаграма відображає класи в системі, а також атрибути, операції та зв'язки між класами.

Структура системи визначається діаграмою класів, що показує її атрибути, зв'язки між об'єктами і т. д. Діаграма є основою об'єктно-орієнтованого моделювання, а також може використовуватися для моделювання даних.

Сутність діаграми класів слід розділити на такі складові. Верхній розділ, який складається з імені класу і є обов'язковим компонентом. Середній розділ описує атрибути класу, які використовуються при описі конкретного екземпляра класу. У наступному розділі описано, як клас взаємодіє з даними, тобто методи класу.

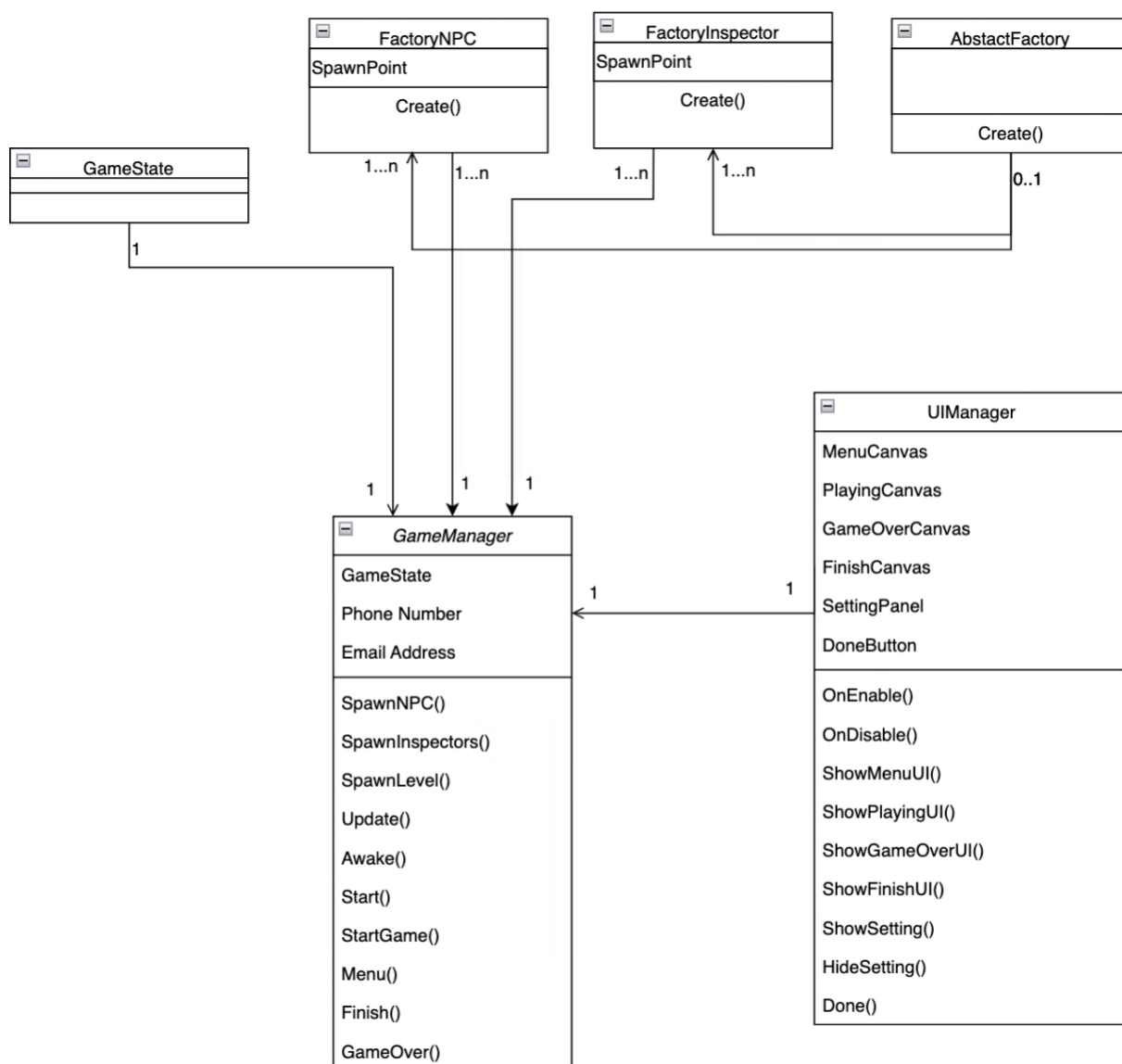


Рисунок 2.10 – Діаграма класів

Діаграми стану необхідними для пояснення роботи складних об'єктів. Діаграма станів показує, як об'єкт переходить з одного стану в інший. Очевидно, що діаграми будуть використовуватися для моделювання динамічних аспектів систем.

Діаграма стану в нагоді для моделювання життєвого циклу об'єкта. Діаграма відрізняється від інших тим, що описує процес зміни лише одного екземпляра певного класу – об'єкта, поведінка якого характеризує реакцію на зовнішні події. Поняття життєвого циклу використовуються саме для таких об'єктів, стан яких визначається минулим станом. Але діаграми стануть важливими не тільки для опису динаміки окремих об'єктів. Деякі діаграми використовуються для опису поведінки окремих об'єктів, але також можуть використовуватися для визначення функціональності інших компонентів моделі, таких як варіанти використання, актори, підсистеми, операції та методи.

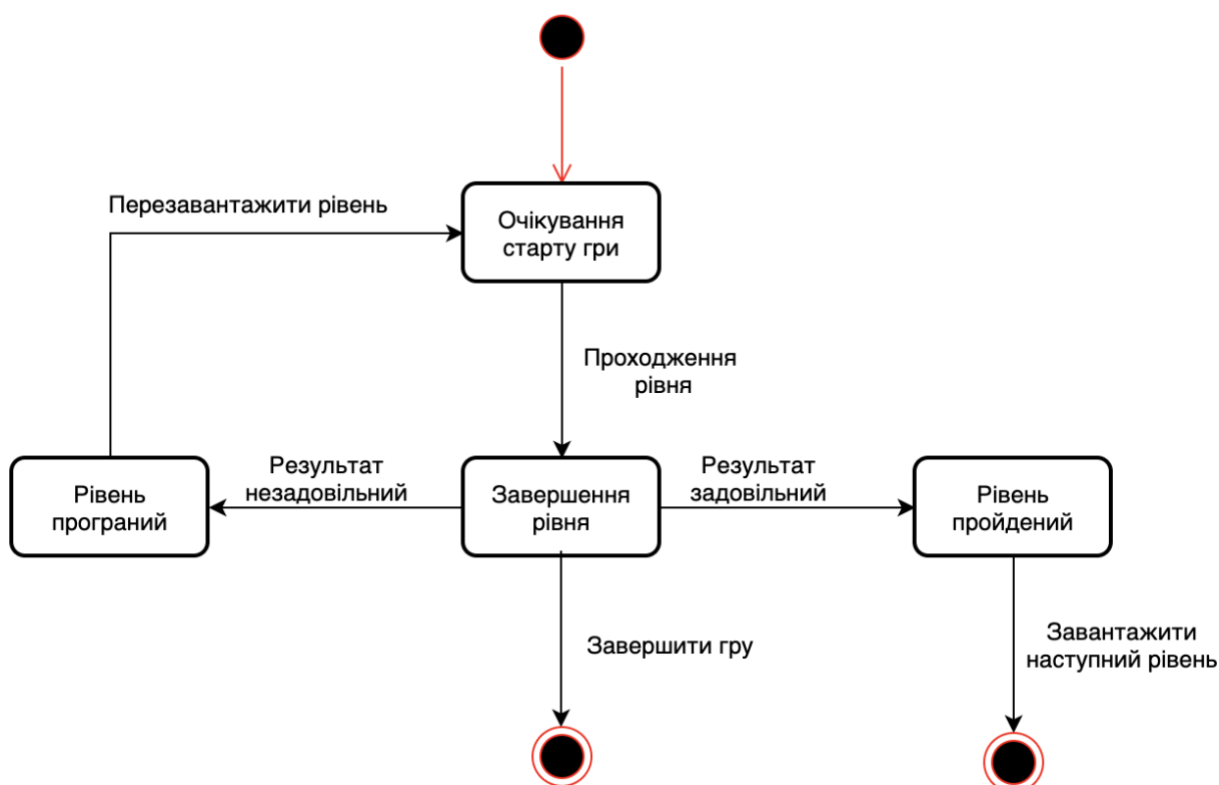


Рисунок 2.11 – Діаграма станів «Проходження одного рівня»

Діаграма діяльності використовується для моделювання послідовного робочого процесу великої діяльності, зосереджуючись на послідовності дій і відповідних умовах для ініціювання дії. Статус активності означає завершення кожного кроку робочого процесу.

Діаграма діяльності зображена фігурами, з'єднаними стрілками. Стрілки йдуть від початку дії до кінця і представляють послідовний порядок виконуваних дій. Чорне коло відображає початковий стан робочого процесу. Обведене чорне коло представляє кінцевий стан. Заокруглені прямокутники представляють виконані дії, які описуються текстом всередині кожного прямокутника.

Після завершення дій необхідно вибрати перехід (або набір послідовних дій) із набору альтернативних переходів для всіх випадків використання.

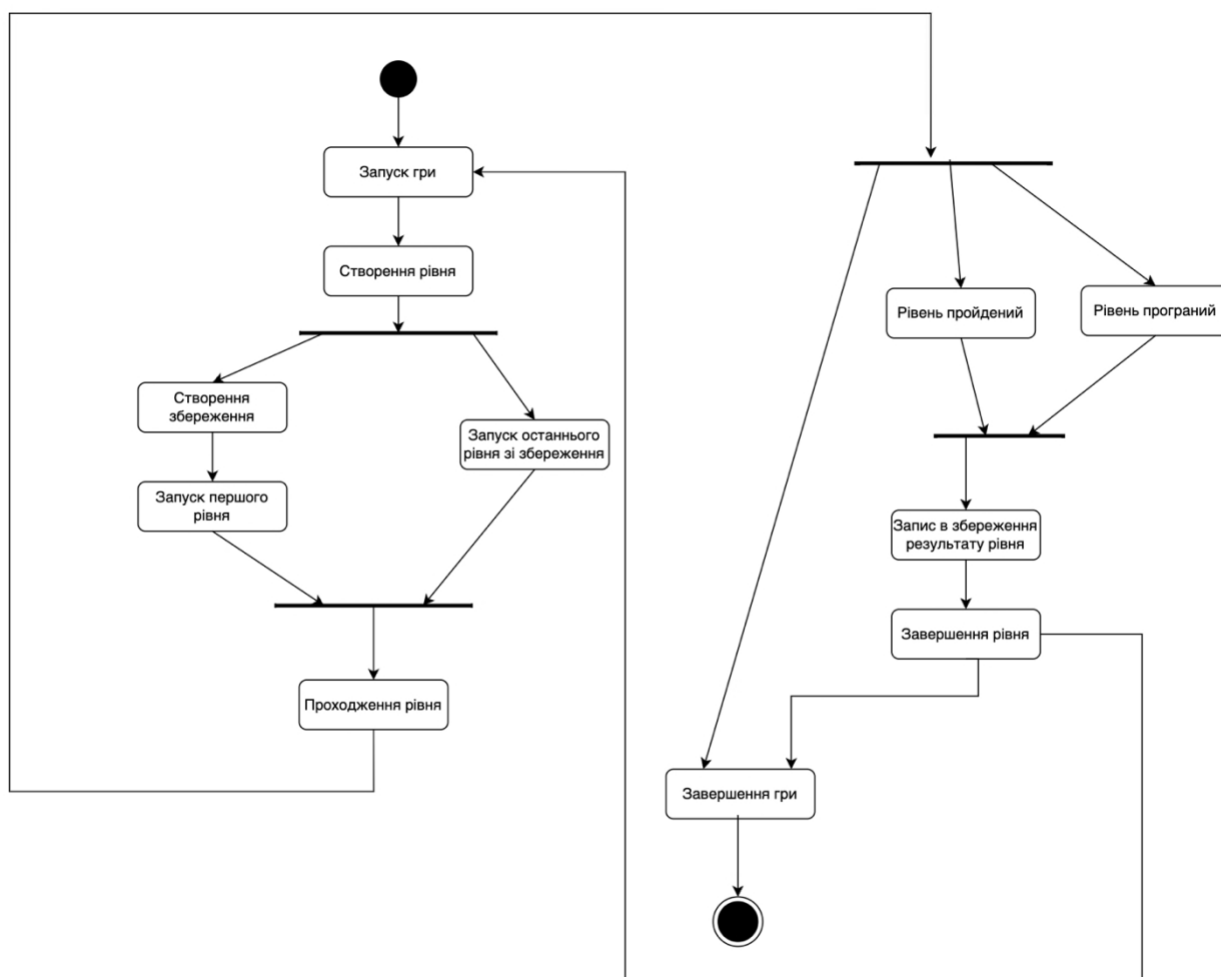


Рисунок 2.12 – Діаграма діяльності

Таким чином, можна відзначити, що UML-моделювання має стати основою для проектування та балансування сценаріїв ігрових додатків набагато раніше його створення, тобто на етапі проектування гри. Широкі можливості UML-моделей для відображення реальних подій і об'єктів у віртуальному світі (аналіз складу і кількості дій, впливів і послідовностей, самовизначення і зв'язків) можуть бути ефективно реалізовані при проектуванні ігрових додатків.

3 ВИБІР ТА ОБГРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис механіки реалізації гіпер казуальної гри

Гіперказуальні ігри мають такі компоненти, як: просте та зрозуміле керування, портретні види, простий візуальний контент та короткі ігрові сесії. Головна вимога до гіперказуальної гри — можливість грати в гру незалежно від місця чи часу.

Гіперказуальні ігри почалися в 2013 році, коли вийшла гра Flappy Bird, яка на той момент не мала аналогів і власного жанру. Якщо подивитися на поточний обсяг ГКІ, то можна виділити кілька найбільш часто використовуваних механік. Важливо відзначити, що єдиного підходу до структурування такої механіки не існує. Для різних джерел існують різні класифікації. На основі дослідження ГКІ можна окреслити та проілюструвати наступні актуальні та актуальні механізми.

Якщо подивитися на поточний обсяг ГКІ, то можна виділити кілька найбільш часто використовуваних механік. Важливо відзначити, що єдиного підходу до структурування такої механіки не існує. Для різних джерел існують різні класифікації. На основі дослідження ГКІ можна окреслити та проілюструвати наступні актуальні та актуальні механізми.

Puzzle. У цьому випадку слід зазначити, що це не традиційна механіка головоломки, яка ставить і ставить перед гравцем складні завдання. Це просто механіка, і лише у форматі рівня інтуїтивного мислення. Створювалася ілюзія розв'язування задач і розгадування головоломок. Головоломок ГКІ мають простий ігровий процес і приємне візуальне оформлення. Прогрес швидкий і є відчуття вирішення кількох завдань за одиницю часу, що приносить задоволення від гри.

Tap/Timing і Steer/Swerve. Ви повинні розрахувати, скільки разів вам доведеться натиснути на екран або динамічно перемикає положення, щоб уникнути зіткнення з різного роду перешкодами. Ця механіка добре підходить до формату ГКІ.

Merge. Це механізм, вищий в ієрархії важливості, який представляє взаємозв'язок між двома чи більше факторами, що стимулюють інновації. Механіка прогресує для гравця та створює щось веселе в грі.

.IO — це новий механізм, створений після гри Agar.io, який включає в себе механіку збільшення персонажа, а також є невеликим і багатокористувацьким, тобто ігровий процес залучає кількох гравців одночасно. Для ГКІ це значно спрощено, оскільки ГКІ не потребує постійного доступу до Інтернету. На зміну мультиплеєру приходить штучний інтелект, який імітує вигляд і дії реальних гравців. Це робить ігровий процес більш збалансованим і доступним для неігрових гравців.

Merge механіка є посиланням на тетріс. Зберігання речей завжди було цікавим і цікавим проведенням часу і формат мобільних ігор ідеально вписується в ГКІ.

Simulation/Realism і механіка Drawing в новому ГКІ зараз дуже популярні і стали корисними напрямками. Загалом симуляція та реалістичність створюють для гравця відчуття занурення. Тож це має великий позитивний вплив на конверсію користувачів і утримання користувачів у грі. З іншого боку, малювання дає високу ступінь задоволення своїм підходом і діяльністю, яка відповідає найважливішим вимогам ГКІ. Ці механіки дозволяють бути більш впевненим в успіху гри.

В ігровому додатку була використана механіка Timing. Механіка, яка змушує гравця чекати певного моменту для проходження рівня. Ассети, які використовувалися в ігровому додатку – ARP, Ragdoll Mecanim Mixer[7]. Дані ассети мають скрипти, які, додаються до компоненту або об'єкту та дозволяють створювати реалістичні фізичні тіла з анімаціями.

3.2 Вибір операційної системи

Операційна система (ОС) — це системне програмне забезпечення, яке керує апаратними та програмними ресурсами комп'ютера та забезпечує звичайні послуги для комп'ютерних програм. Для роботи всіх комп'ютерних програм, крім мікропрограм, потрібна операційна система. Операційна система з розподілом часу планує завдання для максимального використання системи та може включати облікове програмне забезпечення для розподілу процесорного часу, масової пам'яті, друку та інших ресурсів.

Домінуючою настільною операційною системою є Microsoft Windows з часткою ринку близько 83,3%. macOS є продуктом Apple Inc. це зроблено. на другому місці (11,2%), а на третьому – різновиди Linux (1,55%).

У мобільному секторі (смартфони та планшети) Android від Google домінує на 87,5% і 10,3% в річному обчисленні, за нею йде iOS від Apple з 12,1% і зниженням частки ринку на 5,2%, операційні системи з іншими системами лише на 0,3% вище.

У секторах серверів і суперкомп'ютерів домінують різновиди Linux. Інші спеціалізовані класи операційних систем, такі як вбудовані системи та системи реального часу, доступні для багатьох програм.

Для ігрового додатку були обрані наступні операційні системи: Android та iOS. Оскільки Linux не є популярною системою, було вирішено не розробляти ігровий додаток на цій системі.

3.3 Вибір середовища розробки ігрового додатку

Unity — це середовище, в якому розробляються кросплатформні ігри. За допомогою Unity ви можете створювати програми, які працюють на більш ніж 20 різних операційних системах, включаючи ПК, мобільні пристрої, ігрові консолі, веб-програми тощо. Unity була випущена в 2005 році і досі регулярно оновлюється.[8]

Головними перевагами Unity є візуальне середовище розробки, кросплатформна підтримка та модульна система компонентів. Недоліками є виникнення проблем при роботі з багатокомпонентними схемами та проблеми при інтеграції зовнішніх бібліотек.

На Unity написані тисячі ігор, програм і симуляторів, які охоплюють більшість платформ і жанрів. При цьому Unity використовують як великі розробники, так і незалежні студії.

Редактор Unity має інтерфейс перетягування, показаний на малюнку 3.1. Сам інтерфейс інтуїтивно зрозумілий і простий у використанні. Існує серія вікон, тому ви можете налагоджувати гру безпосередньо в редакторі. Редактор підтримує дві мови сценаріїв: C#[9] і JavaScript (модифікація). Раніше була підтримка Boo (діалект Python), але її вирішили відмовитися від 5-ї версії. Фізичні розрахунки базуються на фізичному механізмі PhysX від NVIDIA.

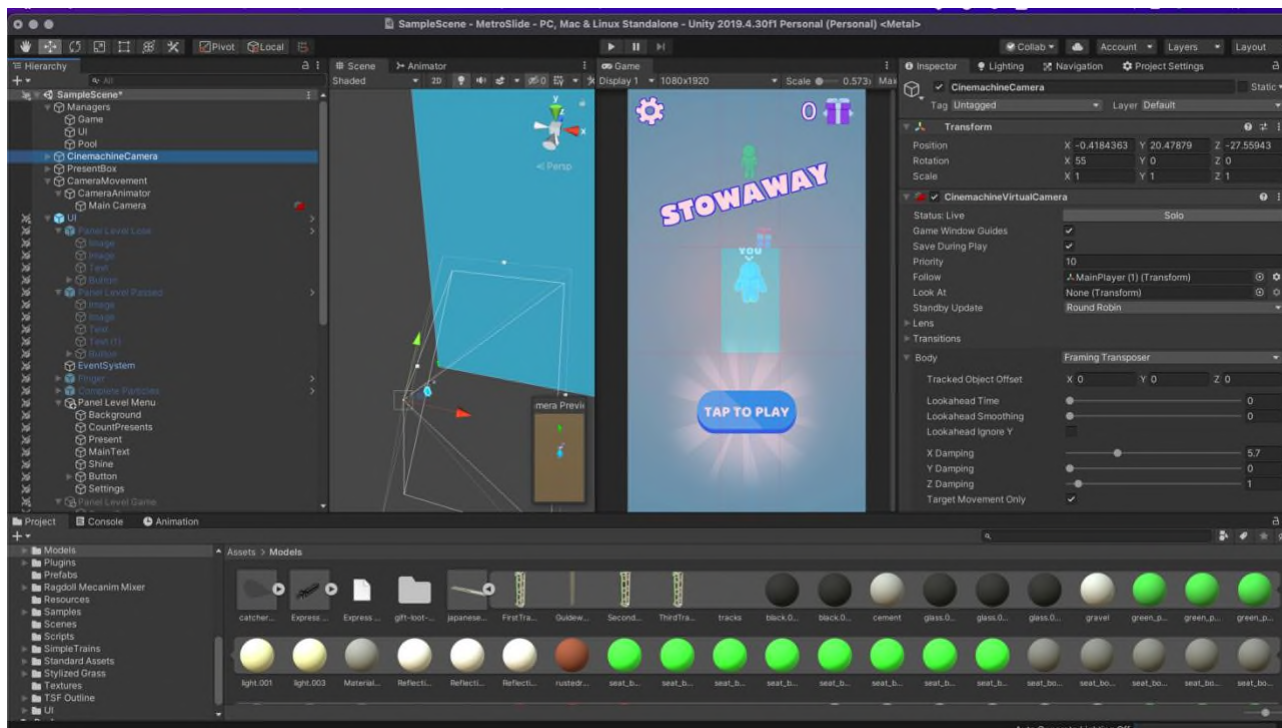


Рисунок 3.1 – Інтерфейс Unity

Проект в Unity ділиться на сцени (рівні) - файли зі своїм ігровим світом, власним набором об'єктів, сценаріями та налаштуваннями. Сцени можуть містити як реальні об'єкти (моделі), так і порожні ігрові об'єкти - об'єкти без моделей («пустота»). З іншого боку, об'єкти містять набір компонентів, з якими взаємодіють скрипти. Об'єкти також мають імена (Unity дозволяє використовувати два або більше об'єктів з однаковими іменами), теги (мітки) і можливі шари для відображення. Таким чином, усі об'єкти на сцені повинні мати компонент Transform – який зберігає координати положення об'єкта, обертання та розміри по всіх трьох осях.

Об'єкти з видимою геометрією також мають за замовчуванням компонент візуалізації Mesh Renderer або Skinned Mesh Renderer, який робить модель об'єкта видимою. Зіткнення можна застосовувати до об'єктів (коллайдери Unity), яких існує багато типів, щоб відтворити реальні зіткнення об'єктів.

Unity також підтримує фізику твердих тіл і тканин, а також фізику Ragdoll. Редактор має систему успадкування об'єктів, і дочірній об'єкт повторює всі зміни положення, повороту та масштабу батьківського об'єкта. Сценарії в редакторі пов'язані з об'єктами в представленнях компонентів.

Як правило, ігровий редактор надає широкий функціонал для ігор, включаючи моделювання фізичного середовища, карти нормалей, динамічні тіні

та багато іншого. На відміну від багатьох редакторів ігор, Unity має дві основні переваги: візуальне середовище розробки та наявність кросплатформної підтримки.

Перший фактор — це не лише набір інструментів для візуального моделювання, а й інтегроване середовище, складальна лінія, спрямована на підвищення продуктивності розробника, особливо на етапах прототипування та тестування.

Кросплатформна підтримка забезпечує не тільки місце розгортання (встановлення на персональних комп'ютерах, мобільних пристроях, консолях тощо), але й наявність інструментів розробки (інтегроване середовище доступне під Windows і MacOS).

Обмеження візуального редактора при роботі з багатокomпонентними схемами обговорюються в недоліки, коли візуальна робота над складними сценами стає складнішою. Другим недоліком була відсутність підтримки Unity, що означало відсутність посилання на зовнішні бібліотеки, завдання, яке вимагало від програмістів самостійного налаштування, а також складної командної роботи.

Ще одним недоліком є використання готових шаблонів. З одного боку, ця концепція від Unity забезпечує інтуїтивно зрозумілий спосіб візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є громіздким. Крім того, завдяки своїй архітектурі (переклад коду з C# на C++) версія редактора WebGL має кілька невирішених проблем із продуктивністю, споживанням пам'яті та продуктивністю на мобільних пристроях.

3.4 Обґрунтування вибору середовища розробки програмного коду

Microsoft Visual Studio - лінійка продуктів компанії Microsoft, що включають інтегроване середовище розробки програмного забезпечення і ряд інших інструментальних засобів. Дані продукти дозволяють розробляти як консольні додатки, так і додатки з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-додатки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, підтримуваних Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone, Android, IOS, .NET Compact Framework і Silverlight. Підтримує наступні мови:

Visual Basic, C ++, C #, F #. Також IDE має свої технології, такі як : IntelliSense, Code Anilizer, Perfomance Analizer, Test Manager, Extension / Updates Manager, Nuget, Git Manager, Archivator, File Manager, Views, Customization, Setting та багато інших.

IDE Visual Studio пропонує ряд високорівневих функціональних можливостей, які виходять за рамки базового управління кодом. IDE Visual Studio має досить багато переваг. Одним за таких є вбудований Web-сервер. Для обслуговування Web-додатки ASP.NET необхідний Web-сервер, який буде очікувати Web-запити і обробляти відповідні сторінки. Наявність в Visual Studio інтегрованого Web-сервера дозволяє запускати Web-сайт прямо з середовища проектування.

Visual Studio дозволяє писати код своєю рідною мовою чи будь-якою іншою мовою, використовуючи один і той же інтерфейс (IDE). Більш того, Visual Studio також ще дозволяє створювати Web-сторінки на різних мовах, але поміщати їх все в один і той же Web-додаток. Єдиним обмеженням є те, що в кожній Web-сторінці можна використовувати тільки якусь одну мову (очевидно, що в іншому випадку проблем при компіляції було б просто не уникнути).

Visual Studio має свій інтуїтивний стиль кодування. За замовчуванням Visual Studio форматує код у міру його введення, автоматично вставляючи необхідні відступи і застосовуючи колірне кодування для виділення елементів типу коментарів. Такі незначні відмінності роблять код більш зручним для читання і менш схильним до помилок. Також в Visual Studio можливо налаштувати автоматичні параметри форматування, що дуже зручно у випадках, коли розробнику більш комфортний інший стиль розміщення дужок, наприклад, стиль K & R, при якому відкриває дужка розміщується на тому самому рядку, що і оголошення, якому вона передує.

Багато з функціональних можливостей Visual Studio спрямовані на те, щоб допомагати розробнику працювати більш швидше. Зручні функції, на зразок функції IntelliSense (яка вміє перехоплювати помилки і пропонувати правильні варіанти), функції пошуку і заміни (яка дозволяє відшукувати ключові слова як в одному файлі, так і в усьому проекті) і функції автоматичного додавання і видалення коментарів (яка може тимчасово приховувати блоки коду), дозволяють прискоритись в розробці та збільшити ефективність .

Пропоновані в Visual Studio інструменти налагодження є найкращим засобом для відстеження загадкових помилок і діагностування дивної поведінки. Розробник може виконувати свій код по рядку за раз, встановлювати точки

переривання, при бажанні зберігаючи їх для використання в майбутньому і в будь-який час переглядати поточну інформацію з пам'яті.

Visual Studio також має багато інших функцій: можливість управління проектом; вбудована функція управління вихідним кодом; можливість рефакторизації коду; потужна модель розширюваності.

Досить велика вага пакетів документів вважається суттєвим недоліком.

Причина, по якій Visual Studio ще не стоїть на всіх в світі комп'ютерах - це ціна. Середні і малі компанії не завжди можуть дозволити собі корпоративну підписку, що вже говорити про незалежні розробників, вони віддадуть перевагу безкоштовному аналогу.

Істотним недоліком є обмеженість платформ, здатних взаємодіяти з Visual Studio. Сьогодні це тільки Windows і MacOS. Користувачі системи Linux залишилися без передової IDE, хоч і користувачі Linux-осей посідають третє місце відразу за OS X і Windows.

Завдяки величезній кількості налаштувань, підтримуваних технологій, швидкодії і зручності Visual Studio вважається однією з кращих середовищ розробки.

4 ПРОЦЕС РОЗРОБКИ ІГРОВОГО ДОДАТКУ

4.1 Розробка ігрового циклу

Один ігровий рівень являє собою дію гравця, результат цієї дії в ігровому просторі, реакцію гравця на результат і запит гри на повторення нової дії.

Серцем гри є ігровий цикл, який здійснює послідовний виклик всіх цих модулів в правильній послідовності і синхронізацію їх між собою. Наш ігровий цикл буде реалізований в скрипті і підключений до камери сцени.

Використання класичного ігрового циклу в Unity неможливо через особливості архітектури движка, в цьому його особливість. Кожен об'єкт в Unity має в скрипті два обов'язкових методу. Перший метод Start(), який виконується один раз перед першим кадром гри. Другий метод називається Update(), Unity викликає його кожен кадр і у кожного об'єкта.

Ігровий цикл нашої гри буде складатися з:

- опитування стану зовнішніх пристроїв управління;
- зміна положень ігрових об'єктів (ігрового персонажу, камери);
- очікування завершення рівня;
- оновлення елементів меню і інтерфейсу;
- перехід між рівнями.

Сама логіка гри написана в скрипті GameManager, який відповідає за станами гри :

- Menu;
- Playing;
- GameOver;
- Finish.

Саме за допомогою цих станів вся гра розуміє, що зараз потрібно робити та відображувати. На лістингу 4.1 зображено код, за допомогою якого здійснюється зміна станів гри та перерахунок цих станів.

Лістинг 4.1 – GameManager з його перерахунок та зміною станів гри

```
public enum GameState
{
    Menu,
    Playing,
    GameOver,
```

```

        Finish
    }

public class GameManager : MonoBehaviour
{
    public static GameManager Instance { get; private set; }

    public static event System.Action<GameState, GameState>
    GameStateChanged = delegate { };

    public PlayerFollower player;

    public float platformsSpeed;

    public int countPresents, countPresentsInGame;

    public int levelIndex, levelIndexText;

    [SerializeField]
    private List<Level> levels;

    private Level currentLevel;

    [SerializeField]
    private List<ParticleSystem> finishParts;

    private AbstractFactory abstractFactory;

    public GameState GameState
    {
        get
        {
            return _gameState;
        }
        private set
        {
            if (value != _gameState)
            {
                GameState oldState = _gameState;
                _gameState = value;

                GameStateChanged(_gameState, oldState);
            }
        }
    }

    [SerializeField]
    private GameState _gameState;

    public void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(this.gameObject);
        }
    }
}

```

```

        else
        {
            Instance = this;
        }
        DontDestroyOnLoad(gameObject);
        SpawnLevel();
    }

    void Start()
    {
        Menu();
    }

    private void Update()
    {
        if (Input.GetKey(KeyCode.R))
        {
            Restart();
        }
    }

    private void SpawnNPC()
    {
        for (int i = 0; i < currentLevel.levelTrain.vagons.Count;
i++)
        {
            var newTransform = new GameObject().transform;
            newTransform.parent =
currentLevel.levelTrain.vagons[i].transform;
            for (int j = 0; j <
currentLevel.levelTrain.vagons[i].countNPC; j++)
            {
                newTransform.localPosition =
currentLevel.levelTrain.vagons[i].GetRandomPointInVagon();
                abstractFactory = new FactoryNPC(newTransform);
                var newNPC =
abstractFactory.Create().transform.GetChild(1).GetComponent<NPCCon
troller>();
                newNPC.npcState = NPCController.NPCState.Idle;

currentLevel.levelTrain.vagons[i].NPCs.Add(newNPC);
            }
            for (int k = 0; k <
currentLevel.levelTrain.vagons[i].NPCSeats.Count; k++)
            {
                newTransform.localPosition =
currentLevel.levelTrain.vagons[i].NPCSeats[k].sitPoint.position;
                abstractFactory = new FactoryNPC(newTransform);
                var newNPC =
abstractFactory.Create().transform.GetChild(1).GetComponent<NPCCon
troller>();
                newNPC.npcState = NPCController.NPCState.Sit;

currentLevel.levelTrain.vagons[i].NPCs.Add(newNPC);
            }
        }
    }

```

```

    }
}

private void SpawnInspector()
{
    for (int i = 0; i <
currentLevel.inspectorSpawnPoints.Count; i++)
    {
        abstractFactory = new
FactoryInspector(currentLevel.inspectorSpawnPoints[i]);
        var newInspector = abstractFactory.Create();

currentLevel.inspectors.Add(newInspector.transform.GetChild(1).Get
Component<InspectorController>());
    }
}

public void StartGame()
{
    GameState = GameState.Playing;
    Time.timeScale = 1f;
}

public void Menu()
{
    PoolManager.Instance.SpawnFromPoolLocal("G", new Vector3(-
5f, -2.9f, -354.75f), Quaternion.Euler(new Vector3(-90f, 0, 0)));
    PoolManager.Instance.SpawnFromPoolLocal("G", new Vector3(-
5f, -2.9f, -118.25f), Quaternion.Euler(new Vector3(-90f, 0, 0)));
    PoolManager.Instance.SpawnFromPoolLocal("GV", new
Vector3(-5f, -2.9f, 118.25f), Quaternion.Euler(new Vector3(-90f,
0, 0)));

    GameState = GameState.Menu;
}

public void GameOver()
{
    GameState = GameState.GameOver;
}

public void Finish()
{
    GameState = GameState.Finish;
    foreach (ParticleSystem part in finishParts)
        part.Play();
    PlayerPrefs.SetInt("levelIndex", levelIndex);
    PlayerPrefs.SetInt("levelIndexText", levelIndexText);
    PlayerPrefs.SetInt("countPresents", countPresents +
countPresentsInGame);
    levelIndex++;
    levelIndexText++;
    if (levelIndex > levels.Count - 1) levelIndex = 0;
}

```

```

public void Restart()
{
SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
}

private void SpawnLevel()
{
    countPresents = PlayerPrefs.GetInt("countPresents", 0);
    levelIndex = PlayerPrefs.GetInt("levelIndex", 0);
    levelIndexText = PlayerPrefs.GetInt("levelIndexText", 1);
    currentLevel = Instantiate(levels[levelIndex], new
Vector3(0, 2.5f, 0), Quaternion.Euler(Vector3.zero));
    currentLevel.gameObject.SetActive(true);
    SpawnNPC();
    SpawnInspector();
}
}

```

4.2 Розробка системи меню

Система меню розроблена за допомогою скрипту UIManager.cs, який використовує патерн проектування Singleton. Саме UIManager, контролює всю логіку взаємодії та відображенням меню. Кожен рівень гри має свій набір Canvas під кожен стан гри:

- Panel Level Lose;
- Panel Level Passed;
- Panel Level Menu;
- Panel Level Game.

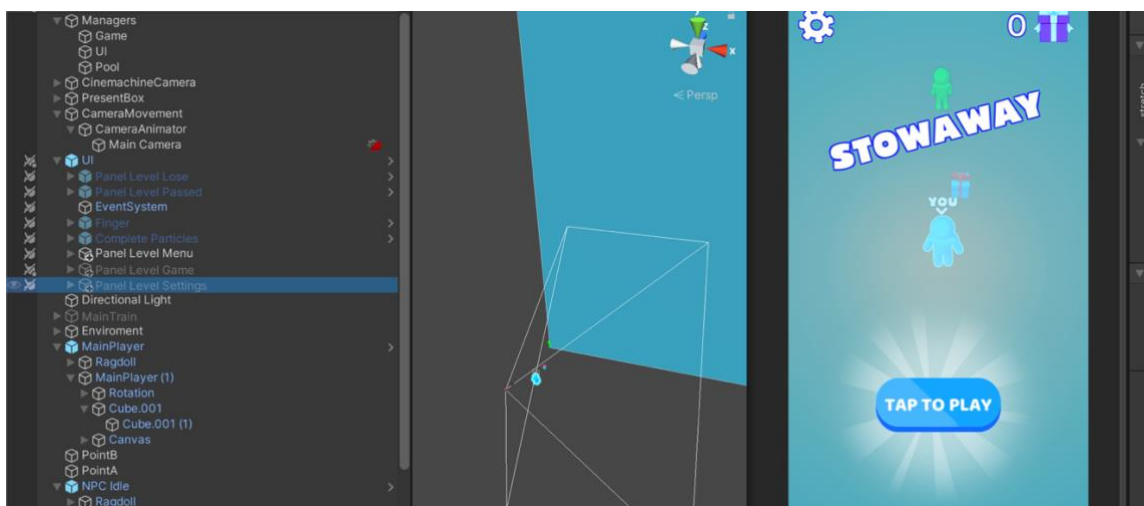


Рисунок 4.1 – Ієрархія всіх компонентів інтерфейсу користувача

На рисунку 4.1 зображено ієрархію інтерфейсу користувача. Взаємодія з інтерфейсом відбувається виключно з скриптом UIManager, саме в ньому реалізовані методи взаємодії користувача та GameManager. В залежності від перелічення стану змінюється і потрібна панель. Кожна панель має потрібний їй сет з картинок, кнопок, текст та кнопки для взаємодії з UI. На лістинку 4.2 зображено як саме UIManager реагує на зміну стану гри. Декілька методів ShowMenuUI(), ShowPlayingUI(), ShowGameOverUI(), ShowFinishUI() роблять активним потрібний компонент Canvas та вимикають всі інші.

Лістинг 4.2 – Частина коду UIManager, яка відповідає за зміну інтерфейсу користувача

```
public class UIManager : MonoBehaviour
{
    public static UIManager Instance { get; private set; }

    [Header("Panels")]
    public GameObject menuCanvas;
    public GameObject playingCanvas;
    public GameObject gameOverCanvas;
    public GameObject finishCanvas;
    public GameObject settingsCanvas;

    [Header("Texts")]
    public Text countMenuText;
    public Text countGameText;

    public Animation plusOne;

    public void Awake()
    {
        if (Instance != null && Instance != this)
        {
            Destroy(this.gameObject);
        }
        else
        {
            Instance = this;
        }
        DontDestroyOnLoad(gameObject);
    }
}
```

```

    }

    void OnEnable()
    {
        GameManager.GameStateChanged +=
GameManager_GameStateChanged;
    }

    private void Start()
    {
        ShowMenuUI();
    }

    private void Update()
    {
        SetText();
    }

    private void SetText()
    {
        countMenuText.text =
GameManager.Instance.countPresents.ToString();
        countGameText.text =
GameManager.Instance.countPresentsInGame.ToString();
    }

    void OnDisable()
    {
        GameManager.GameStateChanged -=
GameManager_GameStateChanged;
    }

    void GameManager_GameStateChanged(GameState newState, GameState
oldState)
    {
        Debug.Log(newState);
        if (newState == GameState.Menu)
        {
            ShowMenuUI();
        }
        else if (newState == GameState.Playing)
        {

```

```

        ShowPlayingUI();
    }
    else if (newState == GameState.GameOver)
    {
        ShowGameOverUI();
    }
    else if (newState == GameState.Finish)
    {
        ShowFinishUI();
    }
}

private void ShowMenuUI()
{
    menuCanvas.SetActive(true);
}

private void ShowPlayingUI()
{
    menuCanvas.SetActive(false);
    playingCanvas.SetActive(true);
}

private void ShowGameOverUI()
{
    playingCanvas.SetActive(false);
    gameOverCanvas.SetActive(true);
}

private void ShowFinishUI()
{
    playingCanvas.SetActive(false);
    finishCanvas.SetActive(true);
}

public void OpenSettings()
{
    settingsCanvas.SetActive(true);
}

public void CloseSettings()
{

```

```

        settingsCanvas.SetActive(false);
    }

    public void StartGame()
    {
        GameManager.Instance.StartGame();
    }

    public void PlusOneActivate()
    {
        plusOne.Play();
    }

    public void RestartLevel()
    {
        GameManager.Instance.Restart();
    }
}

```

4.3 Інтеграція абстрактної фабрики

Рішення обрати саме цей патерн, прийшло з того, що набір різних персонажів, прс, ворогів буде в майбутньому дуже різноманітним, тому потрібен деякий інструмент для зручної та швидкої взаємодії між розробниками і геймдизайнерами. Абстрактна фабрика дуже гарно себе зарекомендувала у розробці, тому і є одним з найпопулярніших патерні проектування у сучасному світі.

Лістинг 4.3 – Код абстрактної фабрики

```

public abstract class AbstractFactory : MonoBehaviour
{
    public abstract GameObject Create();
}

```

Для початку було створено абстрактний клас, а вже його було реалізовано у класах FactoryInspector, FactoryNPC. Був перевизначений метод Create() за допомогою якого буде відбуватись створення нових об'єктів. З

`Resources.Load<GameObject>("")` ми можемо вільно підгружати потрібні нам об'єкти за іменем цього об'єкта. Це не є найбільш оптимізованим способом, але при кількості до 500 об'єктів 95% мобільних телефонів не відчують просадку в покадровому відображенні.

Лістинг 4.4 – Код фабрики інспектора

```
public class FactoryInspector : AbstractFactory
{
    private Transform spawnPoint;

    public FactoryInspector(Transform spawnPoint)
    {
        this.spawnPoint = spawnPoint;
    }

    public override GameObject Create()
    {
        var newPrefab = Resources.Load<GameObject>("Inspector");
        var newObject = Instantiate(newPrefab, spawnPoint.position,
Quaternion.Euler(Vector3.zero));
        newObject.transform.localScale = Vector3.one;
        return newObject;
    }
}
```

Лістинг 4.5 – Код фабрики прс

```
public class FactoryNPC : AbstractFactory
{
    private Transform spawnPoint;

    public FactoryNPC(Transform spawnPoint)
    {
        this.spawnPoint = spawnPoint;
    }

    public override GameObject Create()
    {
        var newPrefab = Resources.Load<GameObject>("NPC");
        var newObject = Instantiate(newPrefab, spawnPoint.position,
Quaternion.Euler(Vector3.zero));
        newObject.transform.localScale = Vector3.one;
    }
}
```

```

        return newObject;
    }
}

```

За допомогою методів `SpawnNPC()`, `SpawnInspectors()` реалізована поява нових об'єктів на сцені.

Лістинг 4.6 – Поява об'єктів на сцені

```

private void SpawnNPC()
{
    for (int i = 0; i < currentLevel.levelTrain.vagons.Count;
i++)
    {
        var newTransform = new GameObject().transform;
        newTransform.parent =
currentLevel.levelTrain.vagons[i].transform;
        for (int j = 0; j <
currentLevel.levelTrain.vagons[i].countNPC; j++)
        {
            newTransform.localPosition =
currentLevel.levelTrain.vagons[i].GetRandomPointInVagon();
            abstractFactory = new FactoryNPC(newTransform);
            var newNPC =
abstractFactory.Create().transform.GetChild(1).GetComponent<NPCCon
troller>();
            newNPC.npcState = NPCController.NPCState.Idle;

currentLevel.levelTrain.vagons[i].NPCs.Add(newNPC);
        }
        for (int k = 0; k <
currentLevel.levelTrain.vagons[i].NPCSeats.Count; k++)
        {
            newTransform.localPosition =
currentLevel.levelTrain.vagons[i].NPCSeats[k].sitPoint.position;
            abstractFactory = new FactoryNPC(newTransform);
            var newNPC =
abstractFactory.Create().transform.GetChild(1).GetComponent<NPCCon
troller>();
            newNPC.npcState = NPCController.NPCState.Sit;

currentLevel.levelTrain.vagons[i].NPCs.Add(newNPC);
        }
    }
}

private void SpawnInspector()
{
    for (int i = 0; i <
currentLevel.inspectorSpawnPoints.Count; i++)
    {

```

```
        abstractFactory = new  
FactoryInspector(currentLevel.inspectorSpawnPoints[i]);  
        var newInspector = abstractFactory.Create();  
  
currentLevel.inspectors.Add(newInspector.transform.GetChild(1).Get  
Component<InspectorController>());}}
```

5 ТЕСТУВАННЯ ІГРОВОГО ДОДАТКУ

5.1 Тестування в середовищі розробки ігрового додатку

Тестування відіграє важливу роль у процесі розробки ігрової програми, оскільки без належного тестування ігрову програму неможливо ефективно використовувати.

В Unity найпростішим і найкращим способом тестування компонента є клас `Debug`, який має власні методи. Повідомлення можуть містити будь-яку бажану інформацію, таку як рядки, цілі значення, логічні значення або комбінації типів змінних. Головним серед цих повідомлень є `Debug.Log()`, який друкує повідомлення за вашим вибором.

Наступні дві функції схожі на `Debug.Log()`. Насправді всі вони починаються з "Log". Ці дві команди: `Debug.LogError()` і `Debug.LogWarning()`. Ви, мабуть, можете сказати, чим займаються ці команди. `LogError()` записує повідомлення про помилку на консоль Unity, тоді як `LogWarning()` друкує попереджувальне повідомлення на консоль. На рисунку 5.1 показано, як дані про стан рівня гри виводяться на консоль за допомогою `Debug.Log()`.

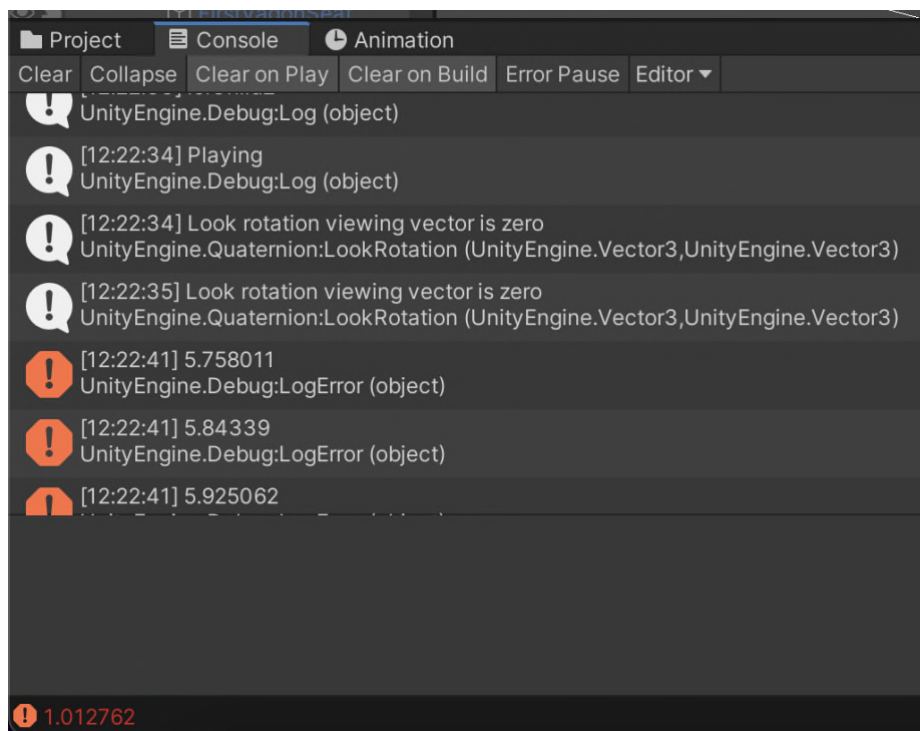


Рисунок 5.1 – Приклад методу `Debug.Log()`.

Налагодження означає більше, ніж друк повідомлень; Ви можете наказати Unity призупинити гру за допомогою `Debug.Break()`. Це корисно, коли потрібно призупинити гру в певних областях, але не можна зробити паузу вручну.

Наприклад, якщо ви хочете переконатися, що снаряд, випущений з об'єкта, слідує правильною траєкторією, ви можете використати `Debug.Break()`, щоб призупинити гру, щоб ви могли проаналізувати цю траєкторію та внести необхідні виправлення в код, який я можу бути зроблено. Додайте код під оператором `Debug.Log()`.

Повертаючись до друку повідомлення, у вас є `Debug.Assert()`. Друкує повідомлення про помилку так само, як `Debug.LogError()`. Але є одна ключова відмінність - він друкується лише в тому випадку, якщо призначене логічне значення `false`. У цьому прикладі оголошено логічне значення для використання в `Debug.Assert()`.

Нарешті, є `Debug.LogException()`, який виводить будь-яке дане повідомлення про винятки в консолі Unity. Тут немає несподіванок, оскільки це веде себе як друк повідомлення про винятки в будь-якому іншому проекті C#. Виняток перехоплюється та відображається на консолі Unity.

5.2 Аналіз контрольного прикладу

Відповідно до обраних патернів проектування було створено ігровий додаток, який відповідає всім вимогам до гіпер казуального жанру. Гра керується «невидимим» джойстиком, який змушує рухатись ігрового персонажа. Головна ціль гри – дійти до фінішної прямої і не бути схопленим інспектором, тобто «проїхати зайцем». Ви можете оминати інспекторів або просто сісти і замаскуватись під звичайного пасажера. Також, по шляху до фінішу будуть зустрічатись подарунки, які можна підбирати. В майбутніх оновленнях буде створена мета-геймплей, в якому буде реалізація цих подарунків і обмін їх на кастомізацію або інші доповнення

Ігровий додаток створений на Unity, кросплатформному середовищі для розробки комп'ютерних ігор. Unity включає не тільки інструментарій візуального моделювання, а й інтегроване середовище, складальний конвеєр, спрямований на підвищення продуктивності розробки. Середовищем розробки

коду було обрано Visual Studio, оскільки Visual Studio IDE надає деякі функції високого рівня, окрім базового керування кодом.

На рисунку 5.2 зображено головна сторінка та меню входу до ігри.

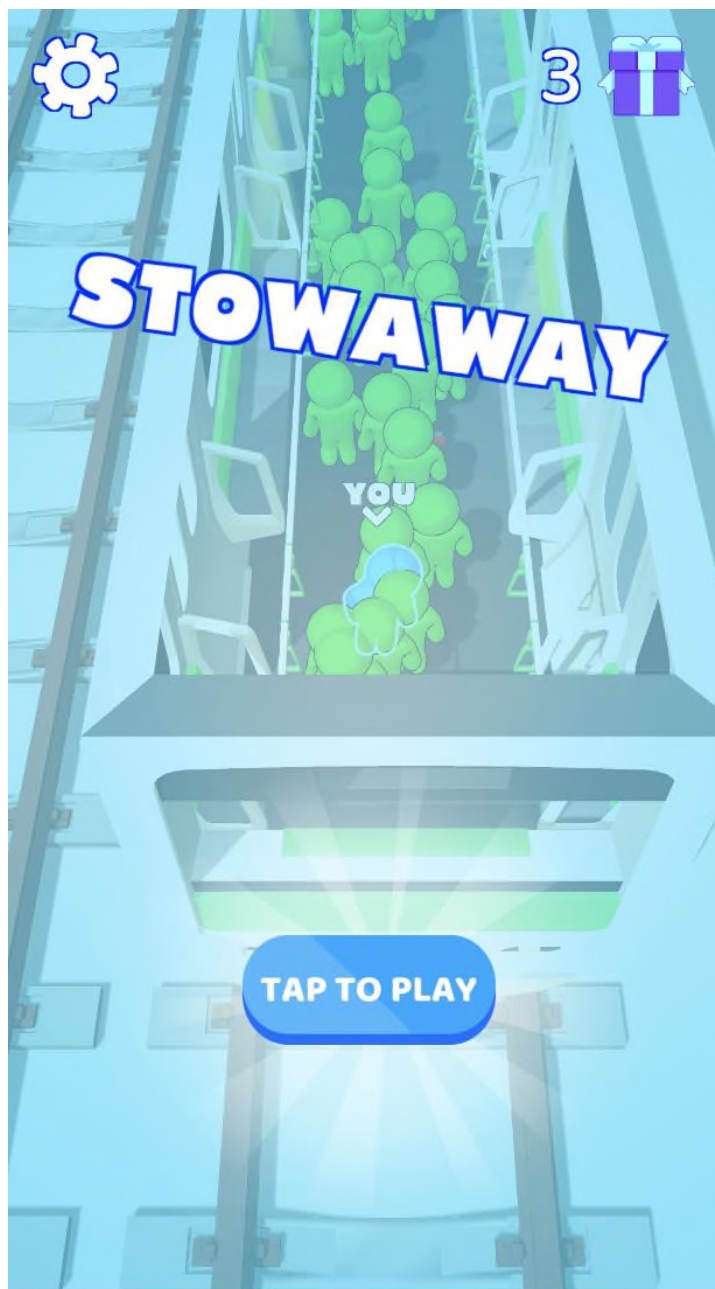


Рисунок 5.2 – Головне меню ігрового додатку.

Потім відразу розпочинається основний геймплей ігрового додатку – біг та підбір влучного моменту, щоб не попастись. Необхідно обирати багато способів, щоб оминути інспектора. На рисунку 5.3 зображено приклад ігрового рівня.

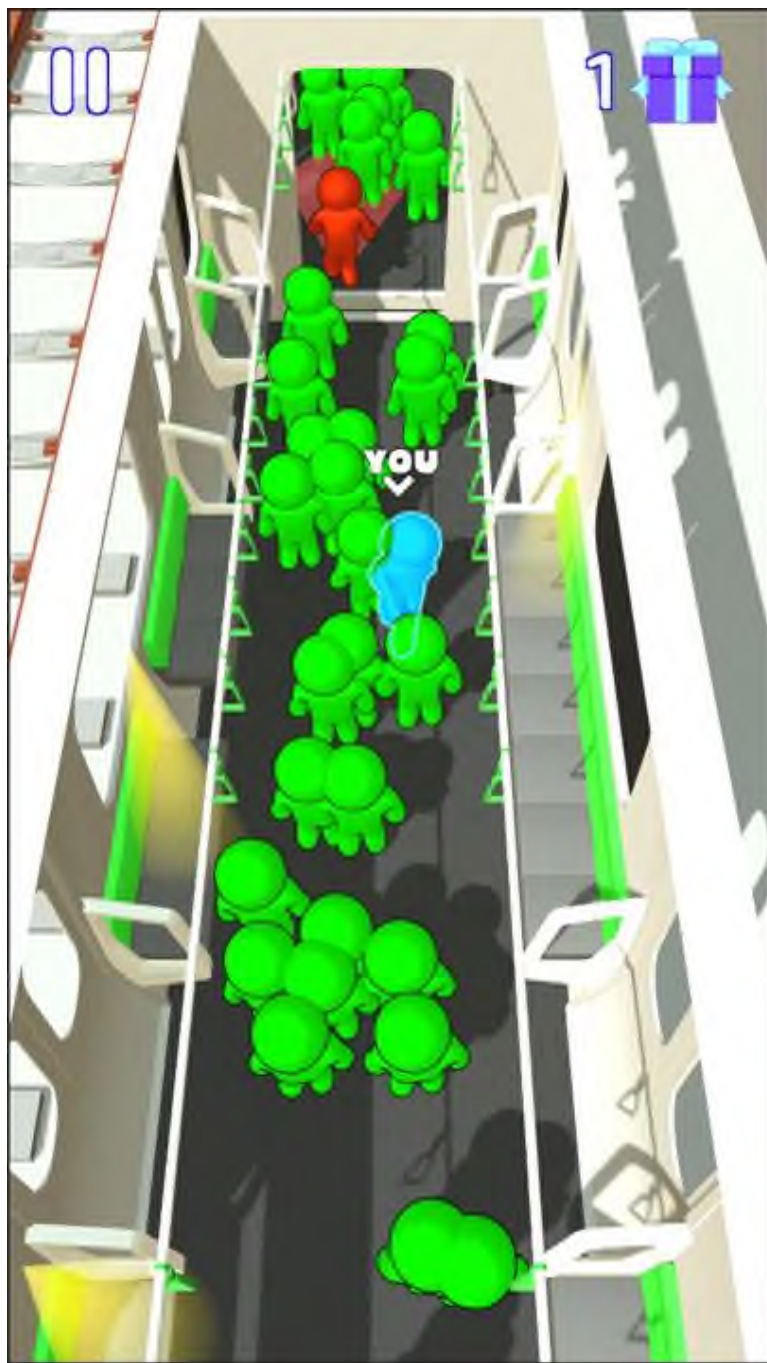


Рисунок 5.3 – Приклад ігрового рівня.

На рисунку 5.2 користувачу пропонується інтуїтивне та просте меню. «Шестерня» - значок, який одразу вказує на налаштування гри. Подарунок і цифра біля нього відображають кількість зібраних подарунків, які є внутрішньою валютою гри. Велика синя кнопка внизу екрану «TAP TO PLAY» означає початок гри. На рисунку 5.4 зображено налаштування гри. В налаштуваннях гри знаходиться 2 пункти, які вмикають та вимикають вібрацію та звуки у грі.

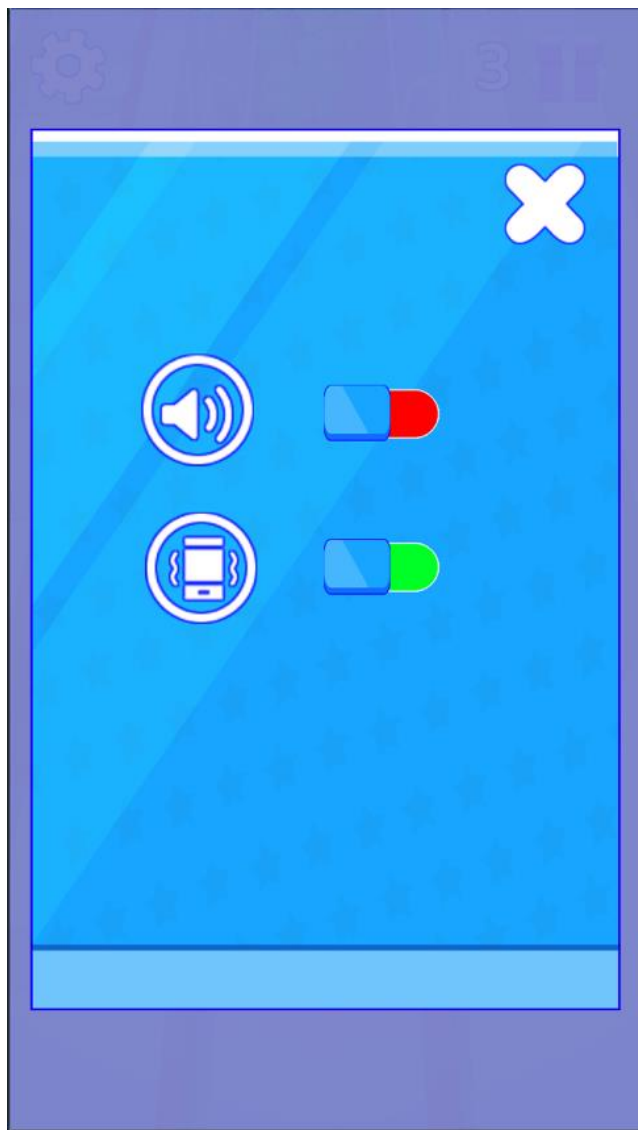


Рисунок 5.4 – Налаштування гри.

Геймплеєм, який покладений у цю гру, є саме переміщення по вагону та уникання інспектора. Для реалізації фізичних персонажів було використано два ассети – ARP, Ragdoll Mecanim Mixer. Дані ассети мають змогу змішувати фізичність об'єктів з анімаціями, які контролюють об'єкт. За допомогою скриптів в ассетах можливо налаштувати сильну чи слабку взаємодію між цима двома станами.

Також в грі реалізоване сховище між ігровими персонажами, які сидять на місцях. Місця на які можна сідати відображаються жовтим напрямом світла. Зображено на рисунку 5.5 приклад міксування фізичного тіла та анімацій.



Рисунок 5.5 – Приклад фізичної анімації.

На рисунку 5.6 наведено переховування головного персонажа між сидячи персонажами. Також відбувається зміна кольору головного героя, щоб злитися з оточуючими

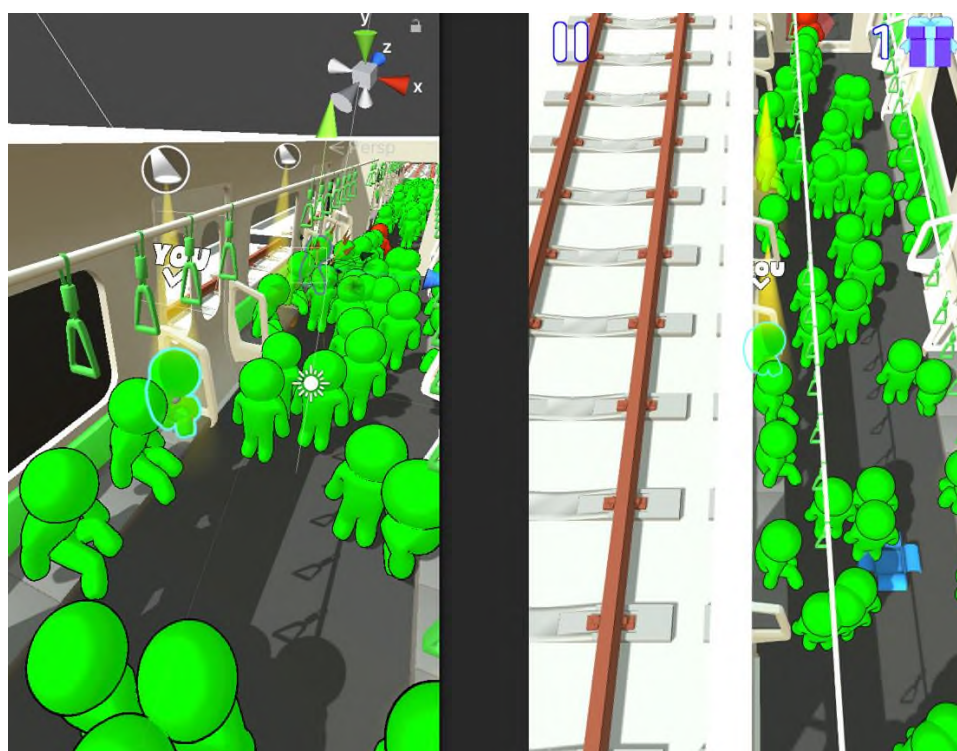


Рисунок 5.6 – Приклад переховування.

На рисунках 5.7 ситуацію коли інспектор піймав головного персонажа. Відображається панель програшу і дозволяється перезапустити гру і почати рівень спочатку.

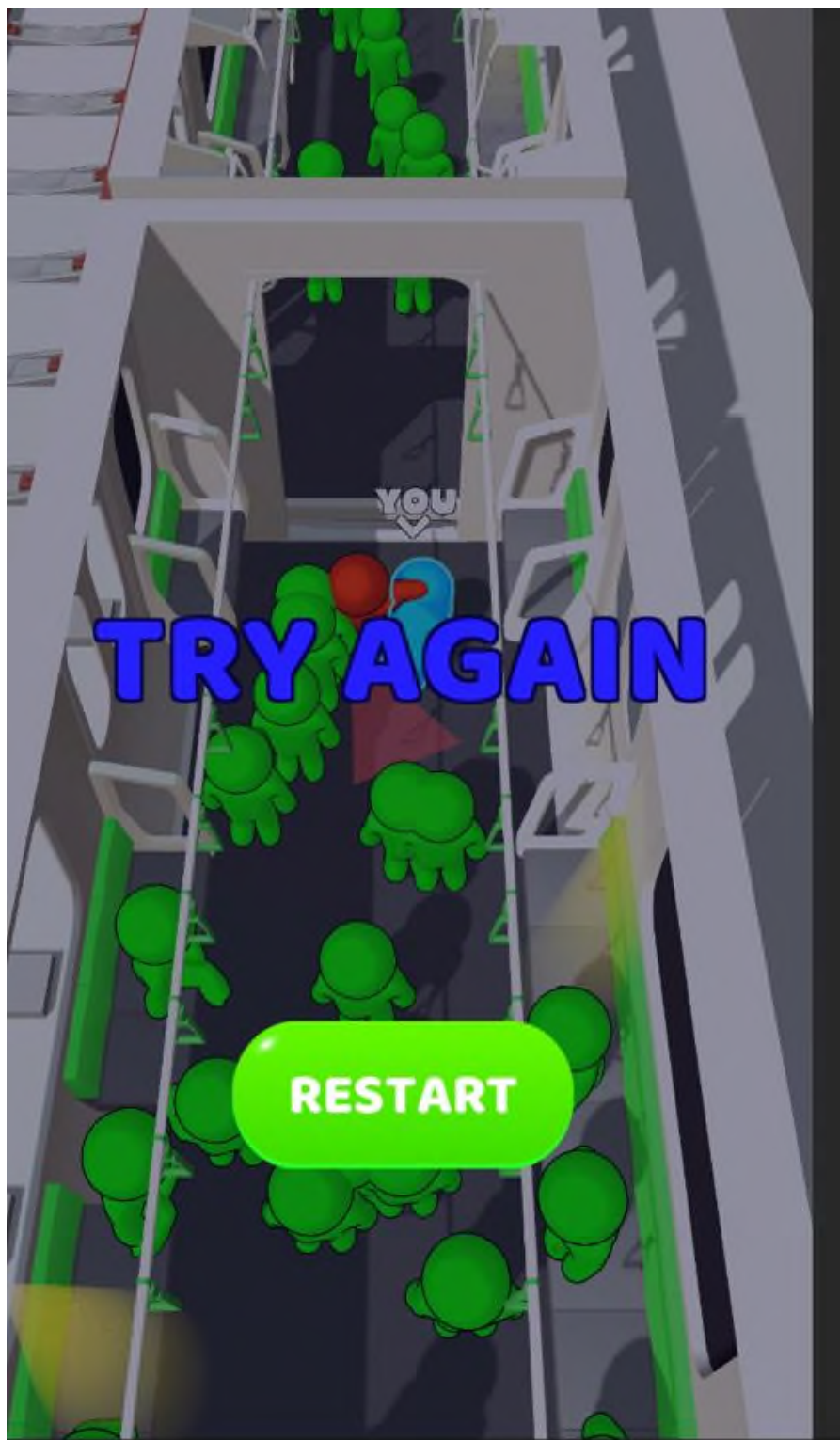


Рисунок 5.7 – Приклад гри на другому рівні

Для проходження рівня нам потрібно дійти до фінішної прямої, яка в свою чергу запустить потрібну панель. Приклад фінішу зображено на рисунку 5.8.

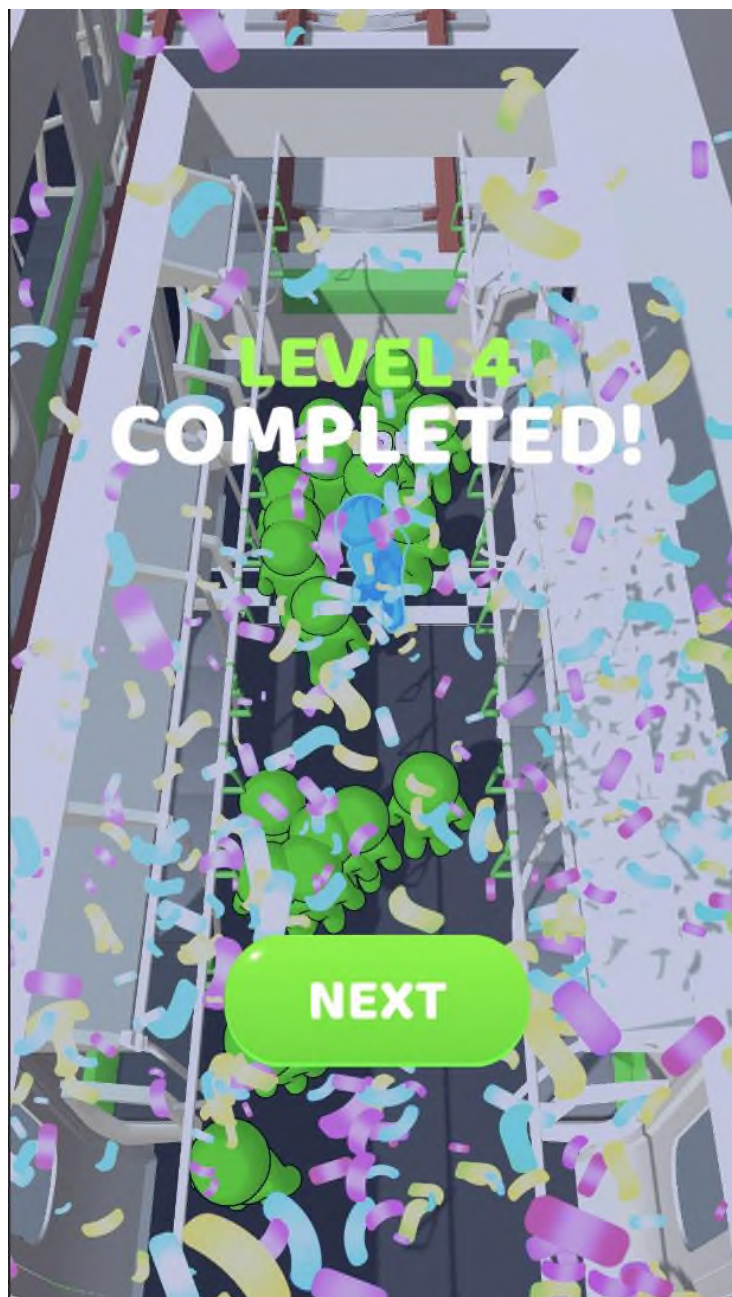


Рисунок 5.8 – Завершення рівня.

Для ігрового додатку були обрані такі ОС: Android та iOS.

Було проведено тестування на різних операційних системах ігрового додатку, явних недоліків, які б заважали б грати в цей додаток не виявлено.

Дослідження застосування ігрового додатку на більш широкому колі аудиторії повинно бути організоване за допомогою видавця та обговорений бюджет для публікації реклами цього застосунку.

ВИСНОВКИ

В кваліфікаційній роботі досліджувалась задача розробки та дослідження інструментального засобу в жанрі гіпер казуальних ігор. За результатами аналізів, які були проведені в ході цієї роботи, було встановлено, що гіперказуальний ринок має більш молодший середній вік ігрової аудиторії. Прибуток від ігрової індустрії буде тільки нарощувати свій потенціал і буде приблизно 211 млрд. доларів на 2025 рік. Було створено ігровий додаток в жанрі гіпер казуальних ігор, який відповідає цьому жанру.

В кваліфікаційній роботі було розроблено функціональні вимоги до процесу розробки гіперказуальної гри методами структурного та об'єктно-орієнтованого моделювання, побудовані функціональні моделі IDEF0, діаграми прецедентів, класів, діяльності, визначені послідовність розв'язання задачі реалізації та проектування ігрового додатку.

Розроблена гіперказуальна гра була створена за цими вимогами. Було дослідження патернів проектування, таких як – Абстрактна фабрика та Синглтон. Ці патерни зарекомендували себе при розробці гіпер казуальних ігор, так як можуть бути швидко реалізовані та мають дуже широкий спектр застосування. Додаток був реалізований в різних програмних середовищах: Android та IOS на мобільних пристроях.

Було проведено тестування ігрового додатку на цих операційних системах та виправлені первинні недоліки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

- 1 Гіперказуальна гра: найприбутковіший жанр мобільних ігор? Режим доступу: <https://ITПм.blog.udonis.co/mobile-marketing/hyper-casual-game>.
- 2 Семенчук В. Мобільний додаток як інструмент бізнесу. Альпіна Паблішер, 2017. 240с.
- 3 Клеметти К., Гренберг Х. Майстри геймдизайну: як створювались Angry Birds, Max Payne та інші бестселлери. Альпіна Паблішер, 2020. 120с.
- 4 Number of available applications in the Google Play Store from December 2009 to March 2022 Режим доступу: <https://ITПм.statista.com/statistics/266210/number-of-availableapplications-in-the-google-play-store/>
- 5 Геллоуей С. Велика четвірка. Секрети успіху Amazon, Apple, Facebook и Google. Манн, Іванов и Фербер, 2020. 352с.
- 6 Петров Е.Г., Новожилова М.В., Гребенник І.В., Соколова Н.А. Методи і засоби прийняття рішення в соціально-економічних і технічних системах/Навчальний посібник. - Херсон: ОЛДІ-плюс, 2003. - 380 с.
- 7 ARP-Player. Unity Asset Store. [Електронний ресурс]. - Режим доступу: <https://assetstore.unity.com/packages/tools/physics/apr-player-169241>.
- 8 Ragdoll Mecanim Mixer. Unity Asset Store. [Електронний ресурс]. - Режим доступу: <https://assetstore.unity.com/packages/tools/physics/ragdoll-mecanim-mixer-bonus-153360>.
- 9 Що таке Unity? [Електронний ресурс]. - Режим доступу: <http://web.spt42.ru/index.php/chto-takoe-unity-3d>.
- 10 Dreamteck Spline. Unity Asset Store. [Електронний ресурс]. - Режим доступу: <https://assetstore.unity.com/packages/tools/utilities/dreamteck-splines-61926>.
- 11 ДСТУ 8302:2015. Інформація та документація. Бібліографічні посилання. Загальні положення та правила складання. – Чинний від 04.03.2016. – Київ: ДП «УкрНДНЦ», 2016. – 20 с.
- 12 Методичні вказівки до передатестаційної практики (для студентів спеціальності 122 «Комп'ютерні науки та інформаційні технології») / Упоряд.: Петров К.Є., Доля О.Є. - Харків: ХНУРЕ, 2020
- 13 Бондар Є.А. Дослідження гіпер-казуальних ігор в навчанні// 25-й Міжнародний молодіжний форум «Радіoeлектроніка та молодь у ХХІ столітті». Зб. матеріалів форуму. Т. 6. – Харків: ХНУРЕ. 2021. – 227 - 228 с.

- 14 Гребеннік І.В., Проектування і створення ігрових додатків / І.В.Гребеннік, Є.В.Губаренко, О.В.Хряпкін: Навч. посібник – Харків: ХНУРЕ, 2018. – 116с.
- 15 Теорія та практика змішаного навчання : монографія / В.М. Кухаренко, С.М. Березенська, К.Л. Бугайчук, Н.Ю. Олійник, Т.О. Олійник, О.В. Рибалко, Н.Г. Сиротенко, А.Л. Столяревська; за ред. В.М. Кухаренка – Харків: «Міськдрук», НТУ «ХП», 2016. – 284 с.
- 16 Blendedlearning. [Електронний ресурс]. - Режим доступу: http://en.wikipedia.org/wiki/Blended_learning.
- 17 Unity. [Електронний ресурс]. - Режим доступу: <https://unity.com/products>.
- 18 Bondar Yehor, Bezugla Anna, Gubarenko Yevhen. DEVELOPMENT OF HYPERCASUAL GAMES FOR DISTANCE EDUCATION // The 5 th International scientific and practical conference “Science, innovations and education: problems and prospects” (December 8-10, 2021) CPN Publishing Group, Tokyo, Japan. 2021. 1068 p.c343-350.
- 19 Безугла Г.Є. Гейміфікація як інструментальний засіб в дистанційному навчанні//Міжнародна науково-практична конференція — Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку/Збірник тез доповідей (м. Харків, 15 березня 2021 р.). – Харків. –2021. –49с
- 20 Game Engine Architecture 2nd Edition. Jason Gregory. 2014 – 1052с.