ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

КВАЛІФІКАЦІЙНА РОБОТА

Тема: «Комп'ютерна гра «Hex-Factory» з використанням гексагональної мап та використанням алгоритму пошуку шляху»

Ступінь вищої освіти – бакалавр
Спеціальність – 122 «Комп’ютерні науки»
Освітня програма «Комп’ютерні науки»

ПОЯСНЮВАЛЬНА ЗАПИСКА

Виконав (-ла):здобувач (ка) 4 курсу
групи КН-21
Сергій ПАШКОВСЬКИЙ
Керівник: зав. кафедри комп'ютерних наук,
к.е.н., с.н.с., доцент,
Сергій МІЧКІВСЬКИЙ

Засвідчую, що кваліфікаційна робота оформлена відповідно до ДСТУ 3008:2015 та не містить запозичень з праць інших авторів без відповідних посилань.
Здобувач: _____
(підпис)

м. Київ – 2025 рік

ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«УНІВЕРСИТЕТ ЕКОНОМІКИ ТА ПРАВА «КРОК»

ЗАТВЕРДЖУЮ:
завідувач кафедри
комп’ютерних наук
Сергій МІЧКІВСЬКИЙ
«____»____20___р

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Пашковський Сергій Олександрович

Тема роботи	Комп'ютерна гра «Hex-Factory» з використанням гексагональної мапи та використанням алгоритму пошуку шляху
Номер та дата наказу про затвердження теми	№121-7 від 24 грудня 2024 року
Коротка постановка завдання	Розробити гру з гексагональною мапою, що включає переміщення персонажа, збір ресурсів та інших механік.
Посилання на джерела інформації (не більше п’яти найменувань, які рекомендує науковий керівник)	1. Red Blob Games. Hexagonal Grids. – URL: https://www.redblobgames.com/grids/hexagons/. (дата звернення 24.01.2025) 2. A Pathfinding* – Red Blob Games. – URL: https://www.redblobgames.com/pathfinding/a-star/introduction.html. (дата звернення 24.01.2025)
Вимоги до кваліфікаційної роботи	Кваліфікаційна робота має передбачити теоретичне, системотехнічне або експериментальне дослідження складного спеціалізованого завдання або практичної проблеми в галузі комп’ютерних наук, яке характеризується комплексністю та невизначеністю умов і потребує застосування теорій і методів інформаційних технологій.

Дата видачі завдання 27 грудня 2024 р.

Керівник Сергій МІЧКІВСЬКИЙ

Здобувач освітнього ступеня бакалавра Сергій ПАШКОВСЬКИЙ

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання	Примітка
	Підготовчий етап		
1	Вибір напряму дослідження	02.12.2024 р.	*виконано*
2	Формування теми та призначення керівника	16.12.2024 р.	*виконано*
3	Затвердження теми кваліфікаційної роботи	23.12.2024 р.	*виконано*
4	Затвердження завдання на кваліфікаційну роботу	27.12.2024 р.	*виконано*
	Основний етап		
5	Розробка концепції кваліфікаційної роботи	13.01.2025 р.	*виконано*
6	Підбір та вивчення джерел інформації з напряму дослідження. Огляд існуючих аналогів	20.01.2025 р.	*виконано*
7	Затвердження розширеної постановки завдання. Підготовка та подання керівникові розділу 1 кваліфікаційної роботи	10.03.2025 р.	*виконано*
8	Проєктування. Підготовка та подання керівникові розділу 2 кваліфікаційної роботи	24.03.2025 р.	*виконано*
9	Підготовка доповіді для експертизи стану виконання кваліфікаційної роботи (проміжний контроль)	31.03-04.04.2025 р.	*виконано*
10	Реалізація. Підготовка та подання керівникові розділу 3 кваліфікаційної роботи	07.04.2025 р.	*виконано*
11	Підготовка та подання керівнику першого варіанту всієї кваліфікаційної роботи	14.04.2025 р.	*виконано*
12	Доопрацювання кваліфікаційної роботи з урахуванням зауважень керівника та представлення керівникові доопрацьованого варіанту кваліфікаційної роботи	21.04.2025 р.	*виконано*
	Завершальний етап		
13	Представлення рукопису для перевірки на плагіат	28.04-04.05.2025 р.	*виконано*
14	Підготовка презентації та доповіді на передзахист	05.05-11.05.2025 р.	*виконано*
15	Передзахист кваліфікаційної роботи	12.05-16.05.2025 р.	*виконано*
16	Доопрацювання роботи за результатами передзахисту	19.05-06.06.2025 р.	*виконано*
17	Експертиза роботи керівником та зовнішнім експертом	09.06-15.06.2025 р.	*виконано*
18	Доопрацювання доповіді та презентації для захисту	09.06-15.06.2025 р.	*виконано*
19	Захист кваліфікаційної роботи	16.06-22.06.2025 р.	*виконано*

Керівник Сергій МІЧКІВСЬКИЙ

Здобувач освітнього ступеня бакалавра Сергій ПАШКОВСЬКИЙ

Пашковський С.О. Комп'ютерна гра «Hex-Factory» з використанням гексагональної мапи та використанням алгоритму пошуку шляху.

Пояснювальна записка кваліфікаційної роботи за спеціальністю 122 – Комп’ютерні науки (освітня програма – Комп’ютерні науки) СО Бакалавр. – ВНЗ «Університет економіки та права «КРОК», навчально-науковий інститут інформаційних та комунікаційних технологій, кафедра комп’ютерних наук, Київ, 2025.

Розроблено гру «Hex Factory», з використанням алгоритмів A* для переміщення між точками та систему добування/крафту ресурсів та інструменту розробки Unity.

Ключові слова: комп’ютерні ігри, алгоритм A*, Unity.

Рис. 32. Бібліограф.: 35 найм.

Pashkovsky S.O. Computer game "Hex-Factory" with a hexagonal map and a path-finding algorithm.

Explanatory note of the qualification work in the specialty 122 – Computer Science (educational program – Computer Science) Bachelor’s degree. – Higher Educational Institution “University of Economics and Law “KROK”, Educational and Scientific Institute of Information and Communication Technologies, Department of Computer Science, Kyiv, 2025.

The game "Hex Factory" was developed, using A* algorithms for moving between points and a resource extraction/crafting system and the Unity development tool.

Keywords: computer games, A* algorithm, Unity.

Fig. 32. Bibliography: 35 Items.

ЗМІСТ

ВСТУП

Актуальність теми. Комп'ютерні ігри сьогодні є сучасною розвагою для більшості людей, і з кожним роком активних гравців стає все більше. Завдяки стрімкому розвитку технологій, ігрова індустрія пропонує все більш реалістичні, інтерактивні та захоплюючі продукти, що приваблюють як широку аудиторію, але через це не всі гравці можуть пограти в нові ігри так як нові ігри потребують все більше та більше ресурсі самого комп'ютеру. Через це гра «Hex Factory» є особливо актуальною, оскільки орієнтована на оптимізацію використання ресурсів комп'ютера.

Мета та завдання дослідження. Метою роботи є розробка програмного забезпечення гри «Hex Factory», що демонструє підхід до створення динамічних ігрових сценаріїв. Для досягнення цієї мети поставлено наступні завдання:

1.Дослідити сучасні методи розробки комп'ютерних ігор.

2.Проаналізувати аналоги комп'ютерних ігор що реалізують ігровий процес з використанням генерації ігрового світу та управління персонажу який добуває та працює об'єктами гри.

3.Спроектувати основні механіки та автоматизовані функції гри.

4.Розробити програмне забезпечення гри «Hex Factory».

Об'єктом дослідження є ігровий процес керування персонажем, добування ресурсів та крафту.

Предмет дослідження є комп'ютерні ігри із застосуванням алгоритмів переміщення та патернів програмування для оптимізації ігрових механік та управління ресурсами.

Методологічна основа дослідження. У роботі використовуються методи алгоритмічної оптимізації, патерни програмування та сучасні підходи до розробки на платформі Unity. Особлива увага приділяється дослідженню та впровадженню алгоритму A* для вирішення завдань оптимального переміщення у ігровому світі.

Практичне значення роботи. Розробка гри «Hex Factory» демонструє можливості застосування різних алгоритмів у сфері розробки комп'ютерних ігор та демонструє один з підходів використання різних алгоритмів для подальших досліджень. Отримані результати можуть бути корисними для розробників ігор та є наглядним прикладом алгоритмів та як механіки працюють з середини.

Структура роботи. Кваліфікаційна робота складається зі вступу, одного розділів, висновків та списку посилань (35 найменувань). Пояснювальна записка містить 32 рисунків. Загальний обсяг пояснювальної записки складає 68 сторінці, основний зміст викладено на 59 сторінках.

РОЗДІЛ 1
ПОСТАНОВКА ЗАВДАННЯ НА РОЗРОБКУ КОМП'ЮТЕРНОЇ ГРИ «HEX-FACTORY»

1.1 Опис предметної області

За даними дослідження Statista, яке аналізує розподіл PC-геймерів за категоріями [1] 2024 року (рис. 1.1), аудиторія комп'ютерних ігор є надзвичайно різноманітною. Такий розподіл свідчить про зростаючий попит на сучасні проекти, які розроблюють сучасні студії по розробці ігро.

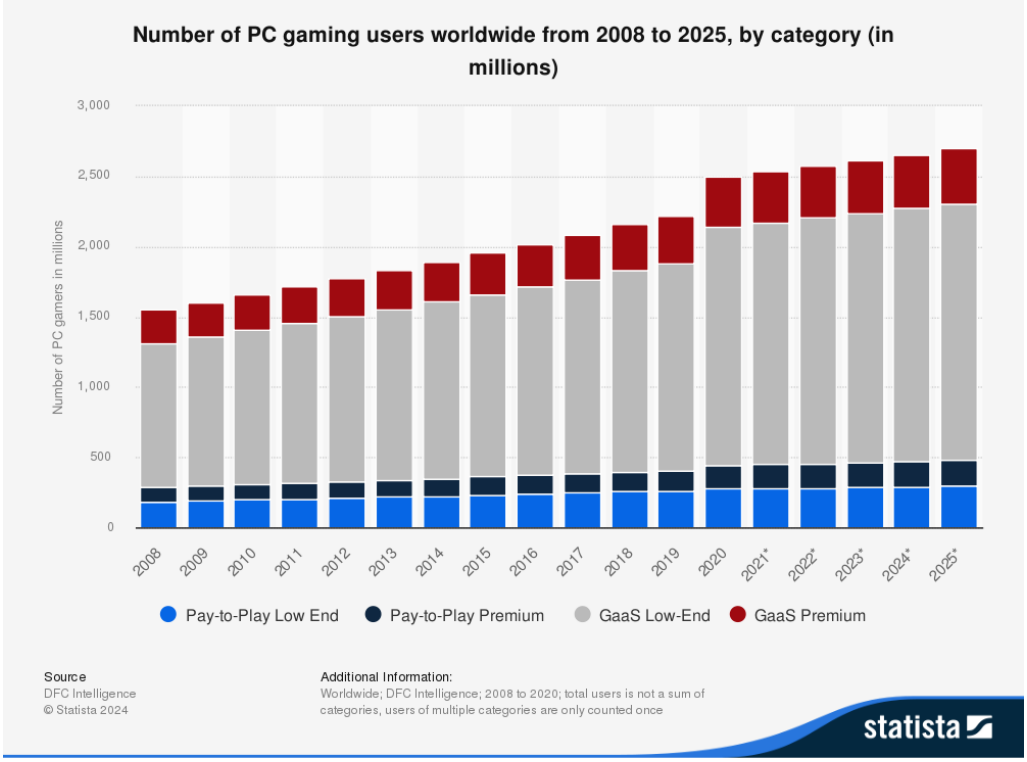

Рисунок 1.1 – Кількість користувачів комп'ютерних ігор у всьому світі з 2008 по 2025 рік, за категоріями Jessica Clement 2024 року

Джерело: [1]

Інтеграція сучасних рішень, таких як штучний інтелект, адаптивні алгоритми та системи оптимізації, дозволяє розробникам створювати продукти, що не лише відповідають очікуванням різних сегментів геймерів, а

й сприяють підвищенню якості геймплею та інтерактивності. Зокрема, використання передових технологій дозволяє адаптувати класичні елементи гри до сучасних потреб, забезпечуючи більш персоналізований та захоплюючий досвід для кожного гравця, але нажаль не кожен гравець зможе пограти в сучасні відео ігри так як в них дуже велика потреба в потужному залізі комп’ютера [14] (рис. 1.2), на цьому графіку видно кількість користувачів які використовують різні покоління відеокарт.

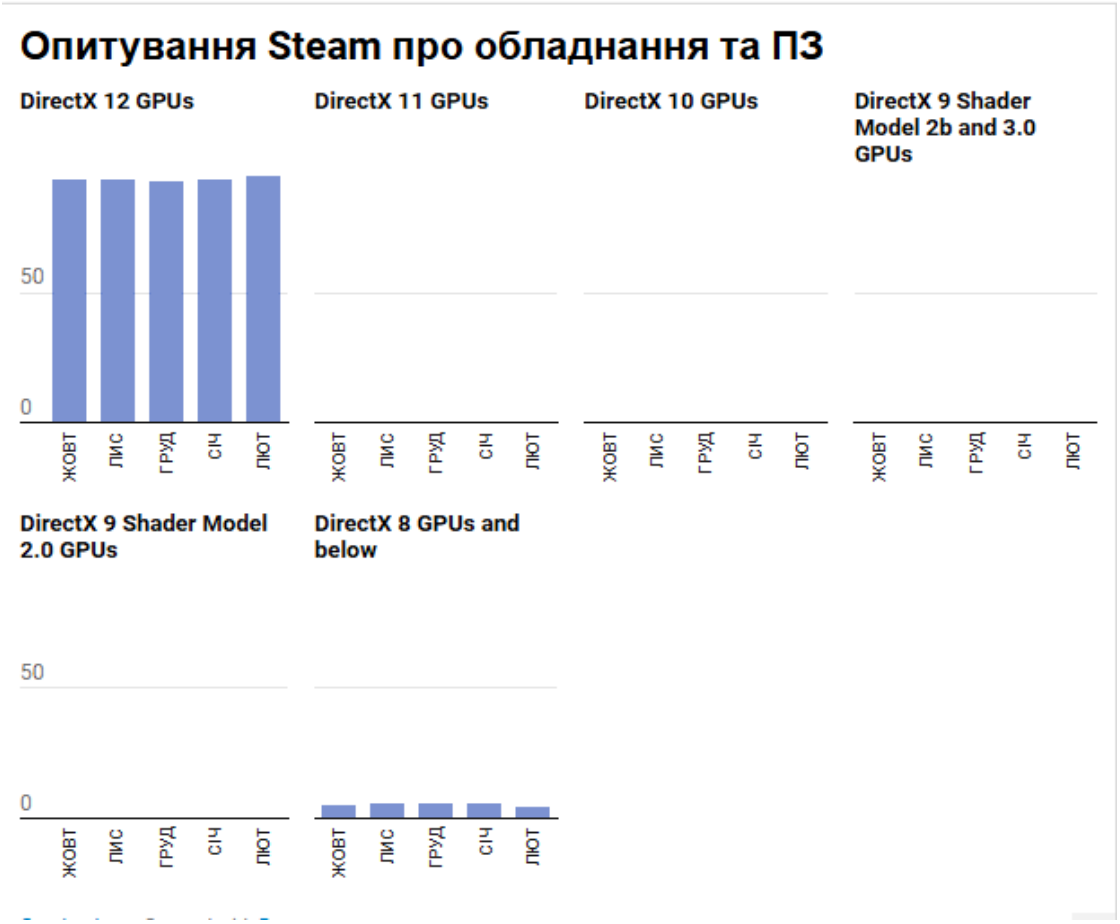

Рисунок 1.2 – Кількість користувачів комп'ютерних ігор у всьому світі з 2008 по 2025 рік, за категоріями Jessica Clement 2024 року

Джерело: [14]

Якщо проілюструвати цю тенденцію, можна використати графіки та діаграми з ресурсів Statista та Steam, які демонструють динаміку зростання

інтересу до інноваційних ігор та те що не всіх гравців є змога пограти в ці ігри через те що в них не дуже потужний комп’ютер для. Такі ілюстрації підтверджують, що сучасна індустрія відеоігор активно впроваджує новітні технологічні рішення, що дозволяє не лише залучати користувачів, але й стимулювати розвиток нових ринкових ніш.

Таким чином, аналіз даних Statista та Steam підтверджує, що зростаюча різноманітність аудиторії та їх потреб створює сприятливе середовище для впровадження інноваційних технологій у розробку комп’ютерних ігор, що відкриває широкі перспективи для подальшого розвитку галузі.

1.2 Огляд аналогів

Перш ніж розпочати розробку власної гри, доцільно вивчити вже існуючі аналоги та проаналізувати, як саме вони реалізують ігрові механіки, системи прогресу та взаємодію з гравцем. Такий ретельний аналіз дає змогу глибше зрозуміти механіки, що користуються популярністю, виявити можливі недоліки та, врешті-решт, застосувати отримані знання для створення ще більш захопливої та якісної гри.

У процесі формування концепції нашого проєкту було виділено три гри, які в різні часи стали джерелом натхнення для розробників усього світу. Це Minecraft, Satisfactory та Sid Meier’s Civilization VI. Кожна з цих ігор має свої унікальні елементи та механіки, котрі суттєво вплинули на формування ідеї нашої власної гри.

Minecraft (рис. 1.3) – це всесвітньо відома гра в жанрі «пісочниця», що пропонує гравцям майже необмежену свободу дій та величезний простір для експериментів [2]. Вона дозволяє видобувати ресурси, знищувати та встановлювати блоки, а також створювати предмети за допомогою системи крафту.

Рисунок 1.3 – Гра Minecraft

Джерело: [2]

Позитивні аспекти Minecraft:

1. Відкритість та креативність. Гра надає необмежені можливості для творчості, де кожен гравець може будувати унікальні конструкції, експериментувати з ресурсами та створювати власні світи за допомогою інтуїтивного інструменту редагування.

2. Широка аудиторія. Завдяки простоті та доступності, Minecraft знаходить відгук у гравців різного віку та інтересів, що робить її однією з найпопулярніших ігор у світі.

3. Простий, інтуїтивний інтерфейс. Зрозумілий інтерфейс дозволяє легко освоїти базові механіки гри, не перевантажуючи користувача надмірною інформацією, що сприяє швидкому входженню у гру.

Недоліки Minecraft:

4. Хаотичність геймплею. Відсутність чіткої структури може призводити до відчуття безладдя, особливо під час спроб організувати комплексні проекти чи управляти великими світами.

5. Слабка структурованість. Для шанувальників глибокої стратегії, відсутність визначеної системи розвитку та обмежень може стати мінусом, зменшуючи елемент стратегічного планування та тактики.

Загалом, Minecraft є яскравим прикладом успішної «пісочниці», яка заохочує гравців експериментувати, досліджувати та розвивати власну креативність. Саме можливість ставити та руйнувати об’єкти, створювати предмети, а також наявність базових механік виживання лягла в основу багатьох популярних ігор.

Satisfactory (рис. 1.4) – це гра в жанрі будівництва фабрик та дослідження відкритого світу від першої особи [4]. Основна мета полягає в оптимізованому видобутку ресурсів і побудові автоматизованих ланцюжків їхньої переробки. Гравці розміщують конструкції, прокладають конвеєрні стрічки та налагоджують систему виробництва, аби ефективно збирати й обробляти ресурси.

Рисунок 1.4 – Гра Satisfactory

Джерело: [4]

Позитивні аспекти Satisfactory:

1. Інноваційність. Передові технології виробництва та автоматизації створюють абсолютно новий рівень геймінгового досвіду, впроваджуючи оригінальні підходи до побудови баз і виробничих ліній.

2. Тактична глибина і складність прийняття рішень. Кожне рішення в грі Satisfactory впливає на оптимізацію виробничих процесів, логістики та розподілу ресурсів, що ставить перед гравцем численні виклики та сприяє розвитку стратегічного мислення.

3. Оригінальний дизайн і механіки, що виділяють гру. Унікальна візуальна естетика, інноваційні механіки побудови та оптимізації баз надають Satisfactory неповторну атмосферу дослідження, творчості та розвитку.

Недоліки Satisfactory:

4. Обмежена аудиторія. Гра орієнтована на нішеву групу гравців, зацікавлених у складних симуляторах виробництва, що може обмежувати її популярність серед широкої аудиторії.

5. Менша підтримка спільноти та оновлень. Порівняно з іншими іграми жанру, Satisfactory може мати менш активну спільноту та обмежені можливості регулярних оновлень, що впливає на довгострокову підтримку продукту.

Саме основна ідея з побудови складних виробничих ланцюжків та вдосконалення структури фабрики стала важливим джерелом натхнення для нашого проєкту, оскільки вона дарує відчуття прогресу й досягнення, а також заохочує до планування та виважених рішень.

Sid Meier’s Civilization VI (рис. 1.5) – це глобальна покрокова стратегія, у якій гравець керує цілою цивілізацією, розвиває науку, культуру, релігію і воєнну справу [3]. Однією з ключових складових гри є система переміщення юнітів по гексагональній сітці, де кожен хід вимагає ретельного планування.

Рисунок 1.5 – Гра Civilization VI

Джерело: [5]

Позитивні аспекти Civilization VI:

1. Глибокий геймплей. Розгалужене дерево технологій, дипломатичні зв’язки, торгівля та культурні особливості створюють багатошаровий ігровий досвід, який занурює гравця у складні стратегії розвитку цивілізації.

2. Продумана система переміщення. Детальна механіка переміщення одиниць дозволяє оптимально планувати тактичні дії, враховуючи особливості місцевості та забезпечуючи стратегічну перевагу в боях.

3. Різноманіття цивілізацій і механік. Унікальні бонуси та індивідуальні особливості кожної цивілізації розширюють арсенал можливостей гравця, роблячи кожну партію неповторною завдяки широкому вибору стратегій.

Недоліки Civilization VI:

4. Складність для нових гравців. Багатогранність механік та високий поріг входу можуть стати викликом для тих, хто тільки починає знайомство з жанром, вимагаючи часу на освоєння всіх нюансів гри.

5. Значний час на навчання і освоєння можливостей. Глибина стратегій та різноманіття функцій гри потребують значних зусиль та часу для повноцінного розуміння, що може ускладнити початковий досвід гри для новачків.

Важливим елементом, узятим із Sid Meier’s Civilization VI, стала система переміщення (логіка «з точки А в точку Б») та покрокова взаємодія з ігровим світом, що у нашому випадку трансформувалася у використання алгоритму A* для пересування персонажа в просторі.

Узагальнені переваги розглянутих аналогів:

1. Відкритий та/або процедурно згенерований світ. Усі три ігри дають великий простір для дослідження та експериментів із середовищем.

2. Сильна мотивація до розвитку. Гравець відчуває прогрес: в Minecraft – це будівництво масштабних споруд, у Satisfactory – налагодження складних виробничих процесів, у Civilization VI – розвиток цілої цивілізації.

3. Можливість варіативних шляхів. Кожна гра має свої механіки, що дозволяють гравцям творчо й різноманітно підходити до вирішення завдань.

Недоліки в аналогах:

1. Завеликий поріг входу. Особливо це стосується Satisfactory та Civilization VI, де новачкам може знадобитися чимало часу для опанування складних систем гри.

2. Можлива монотонність при відсутності самостійної мети. У відкритих світах (Minecraft, Satisfactory) чи масштабних стратегіях (Civilization VI) гравець має сам визначати напрям розвитку.

3. Тривалість гри. У випадку з Civilization VI часом важко швидко завершити партію, що робить гру менш доступною для тих, хто цінує короткі ігрові сесії.

Особливості нашої гри. Власна гра, що розробляється, поєднує у собі натхнення від Minecraft, Satisfactory та Sid Meier’s Civilization VI:

1. Можливість видобувати та встановлювати об’єкти (за прикладом Minecraft), а також зберігати всі отримані ресурси й предмети в інвентарі з подальшим крафтом корисних речей.

2. Основна ідея побудови виробничих елементів (конвеєр, генератор, проводи), успадкована від Satisfactory, дозволяє зосередитися на плануванні ланцюжків переробки та зручній логістиці всіх процесів.

3. Система переміщення та генерація оточення запозичена з Civilization VI, але з адаптацією під алгоритм A* (А-зіор), який реалізує ефективний пошук шляху з точки А до точки Б у середовищі.

Завдяки такому підходу гра має стати водночас і динамічною, і достатньо глибокою за ігровими механіками. Вона стимулюватиме гравців до креативності, управління ресурсами та стратегічного мислення, проте залишить їм достатній простір, аби самостійно визначати власний стиль гри.

Таким чином, розглянуті аналоги стали потужним підґрунтям для формування концепції, адже вони продемонстрували різні підходи до інтеракції із середовищем, забезпечення мотивації гравця та реалізації ігрового прогресу. Наша гра прагне об’єднати найкращі ідеї, уникаючи при цьому частини недоліків, що були виявлені в процесі аналізу наявних продуктів.

1.3 Постановка задачі

У межах даної роботи планується розробка комп’ютерної гри «Hex Factory», яка включає наступні ключові функції:

1. Рух персонажа за допомогою алгоритму A*: Гравець керує персонажем, який повинен переміщуватись від точки А до точки Б. Для цього використовується алгоритм A* (А-зіор), що дозволяє знаходити найкоротший та оптимальний шлях, враховуючи наявні перешкоди. Цей підхід забезпечує плавний і динамічний рух у грі, де кожна деталь маршруту ретельно просумовує різні фактори, такі як відстань, перешкоди та потенційні ризики.

2. Генерація ресурсів за допомогою рандому: Ігровий світ «Hex Factory» постійно змінюється завдяки процедурній генерації ресурсів. Ресурси розташовуються в різних місцях випадковим чином, що гарантує кожного разу новий унікальний досвід для гравця. Це сприяє підвищенню креативності гри, адже світ завжди залишається непередбачуваним.

3. Можливість видобутку, встановлення та крафту ресурсів: Гравець може не лише добувати ресурси, але й використовувати їх для побудови та створення нових об’єктів. Це включає механіку видобутку – процес збирання ресурсів, подальше їх встановлення в ігровому середовищі та можливість крафтувати з них різноманітні предмети. Такий підхід забезпечує глибокий рівень взаємодії з ігровим світом і стимулює творчість, оскільки кожен добутий об’єкт може стати будівельним блоком для створення чогось унікального.

4. Розробка інвентаря з адаптивними слотами: Інвентар – це ключовий елемент, що дозволяє гравцю зберігати всі добуті ресурси та предмети. У грі передбачено створення інвентаря з двома типами слотів: звичайними та швидкими, які можуть бути адаптовані для взаємодії з іншими механіками гри. Це забезпечує зручність управління ресурсами, швидкий доступ до необхідних об’єктів та дозволяє ефективно організувати внутрішню економіку гри.

5. Можливість крафту предметів із використанням ресурсів: За допомогою системи крафту гравець може створювати нові предмети, поєднуючи добуті ресурси за певними рецептами. Це не лише розширює функціональність ігрового процесу, а й стимулює гравців до експериментів та стратегічного мислення, оскільки вони можуть відкривати нові комбінації для покращення своїх можливостей у грі.

6. Можливість переплавлення руд в злитки та інші предмети: Для більшої глибини ігрової механіки введено систему переплавлення, завдяки якій гравець може перетворювати руду на злитки або інші корисні предмети. Цей процес є важливою частиною економічної моделі гри, оскільки дозволяє

гравцям оптимізувати використання ресурсів та вдосконалювати свій арсенал для подальшої роботи.

7. Створення буру для видобутку ресурсів із гексогону: Однією з ключових інновацій гри є спеціальний бур, який використовується для видобутку ресурсів з гексогональних клітин. Цей елемент дозволяє автоматизувати процес добування, роблячи його більш ефективним та інтерактивним.

8. Створення генератора та кабелю для автоматизації роботи буру: Щоб підвищити ефективність видобутку ресурсів, у грі передбачено інтеграцію генератора та кабелю, які з'єднуються з буром. Ця система забезпечує автоматичне живлення буру, що дозволяє безперервно працювати системі видобутку ресурсів і знижує необхідність ручного втручання.

9. Створення адаптивних конвеєрів для переміщення ресурсів: Для оптимізації процесу транспортування добутих ресурсів впроваджено систему адаптивних конвеєрів. Конвеєри автоматично переміщують ресурси до відповідних точок, що забезпечує безперебійну роботу всієї виробничої ланцюга. Це дозволяє гравцеві зосередитися на стратегічному плануванні, а не витрачати час на рутинні операції.

10. Налаштування анімаційного та звукового супроводження для покращення якості ігрового процесу. Для створення більш глибокого занурення в ігровий світ реалізовано деталізовані анімації та якісне звукове оформлення. Анімації підкреслюють динаміку ігрових механік, роблячи взаємодію з об'єктами більш реалістичною, а звукові ефекти додають атмосфери кожній дії. Це сприяє покращенню загального досвіду гравця, роблячи процес гри більш захопливим та емоційної насиченим.

Гра Hex-factory будується на унікальній гексагональній сітці, по якій гравець переміщається з точки А до точки Б, обходячи перешкоди та взаємодіючи з навколишнім світом. У грі реалізовані механіки видобутку ресурсів (наприклад, каменю, дерева та ін.), система інвентарю, а також комплексний процес крафту з тимчасовими затримками. Додатково присутні

елементи енергозабезпечення (генератор, який вимагає заправки, та дроти для передачі електрики), а також механізми видобутку (бур, що видобуває ресурси з гекса) та транспортування (конвеєр для переміщення ресурсів).

Основні можливості системи визначаються наступними функціями:

1. Переміщення та навігація:
   1.1. Рух гравця по гексагональній сітці з урахуванням обходу перешкод.
   1.2. Вибір оптимального маршруту між заданими точками.
2. Добування ресурсів:
   2.1. Механізм видобутку ресурсів (камінь, дерево тощо) із застосуванням інструмента (бур), який забезпечує вилучення матеріалу без руйнування самого гекса.
   2.2. Ведення обліку зібраних ресурсів.
3. Система інвентарю та крафту:
   3.1. Інтерфейс інвентарю з можливістю сортування та керування ресурсами.
   3.2. Процес крафту з відведеним часом для створення предметів.
   3.3. Підсистеми для роботи з печами, що використовуються для обробки матеріалів, та генератора, який потребує заправки для виробництва електроенергії.
4. Енергозабезпечення та логістика:
   4.1. Генератор, що генерує електрику, за умови регулярного поповнення палива.
   4.2. Система проводів для передачі електроенергії між компонентами.
   4.3. Конвеєр для організації транспортування ресурсів між виробничими блоками.

Нефункціональні вимоги забезпечують стабільну, ефективну та безпечну роботу гри:

1. Продуктивність:
   1.1. Швидке завантаження і плавна робота навіть при великій

кількості об’єктів на сітці.

1.2. Оптимізація алгоритмів руху, обробки ресурсів і фізики для мінімізації навантаження на систему.

2. Масштабованість:
   2.1. Можливість розширення функціоналу гри (додавання нових типів ресурсів, об’єктів та механік) без кардинальних змін базової архітектури.
   2.2. Адаптивність до різних апаратних платформ, включаючи менш потужні пристрої.
3. Надійність та безпека:
   3.1. Стабільна робота системи з мінімальними збоями під час інтерактивної взаємодії.
   3.2. Захист збережень, конфігурацій та ігрових даних від несанкціонованого доступу.
4. Зручність використання:
   4.1. Інтуїтивно зрозумілий інтерфейс, що забезпечує просту навігацію та взаємодію з грою.
   4.2. Висока якість графічного оформлення та анімацій, що сприяють зануренню у гру.

Висновки до розділу 1

У цьому розділі проведено всебічний аналіз предметної області, огляд аналогів та постановку задачі для розробки гри «Hex Factory». За даними дослідження Statista, аудиторія комп’ютерних ігор є надзвичайно різноманітною, що створює сприятливе середовище для впровадження інноваційних технологій. Це підтверджує необхідність розробки продуктів, які поєднують класичні ігрові механіки з передовими технологічними рішеннями, зокрема, алгоритмами оптимізації та адаптивними системами.

Огляд аналогів показав, що такі ігри, як Minecraft, Satisfactory та Sid Meier’s Civilization VI, значно вплинули на сучасний ігровий дизайн. Minecraft

надихає свободою творчості завдяки можливості видобутку, встановлення об’єктів та системі крафту , Satisfactory демонструє переваги побудови автоматизованих виробничих ланцюгів та ефективного управління ресурсами , а Civilization VI запроваджує складну систему переміщення та генерації оточення через гексагональну сітку, що було адаптовано в нашому проєкті за допомогою алгоритму A*. Ці аналоги дали можливість виявити сильні сторони існуючих ігор, а також визначити їх недоліки, що дозволяє нам зосередитись на створенні більш інтуїтивного, адаптивного та мотиваційного продукту.

Постановка задачі передбачає розробку інтегрованої системи, яка включає:

ефективний рух персонажа за допомогою алгоритму A*;

процедурну генерацію ресурсів із випадковим розташуванням;

механіки видобутку, встановлення, крафту та переплавлення ресурсів;

створення адаптивного інвентаря;

інтеграцію виробничих елементів (бур, генератор, кабелі, конвеєри) для автоматизації процесів.

Таким чином, розглянуті в цьому розділі дані дозволяють створити міцну основу для подальшої розробки «Hex Factory». Комплексний підхід, що поєднує аналіз сучасних тенденцій, досвід успішних аналогів та чітко сформульовані функціональні вимоги, сприятиме створенню інноваційного продукту з високою конкурентоспроможністю на ринку комп’ютерних ігор.

РОЗДІЛ 2
ПРОЕКТУВАННЯ КОМП'ЮТЕРНОЇ ГРИ «HEX-FACTORY»

2.1 Проєктування структури

Діаграма прецедентів (рис. 2.1) у мові UML відображає взаємозв'язки між акторами та конкретними діями, які вони виконують у системі.

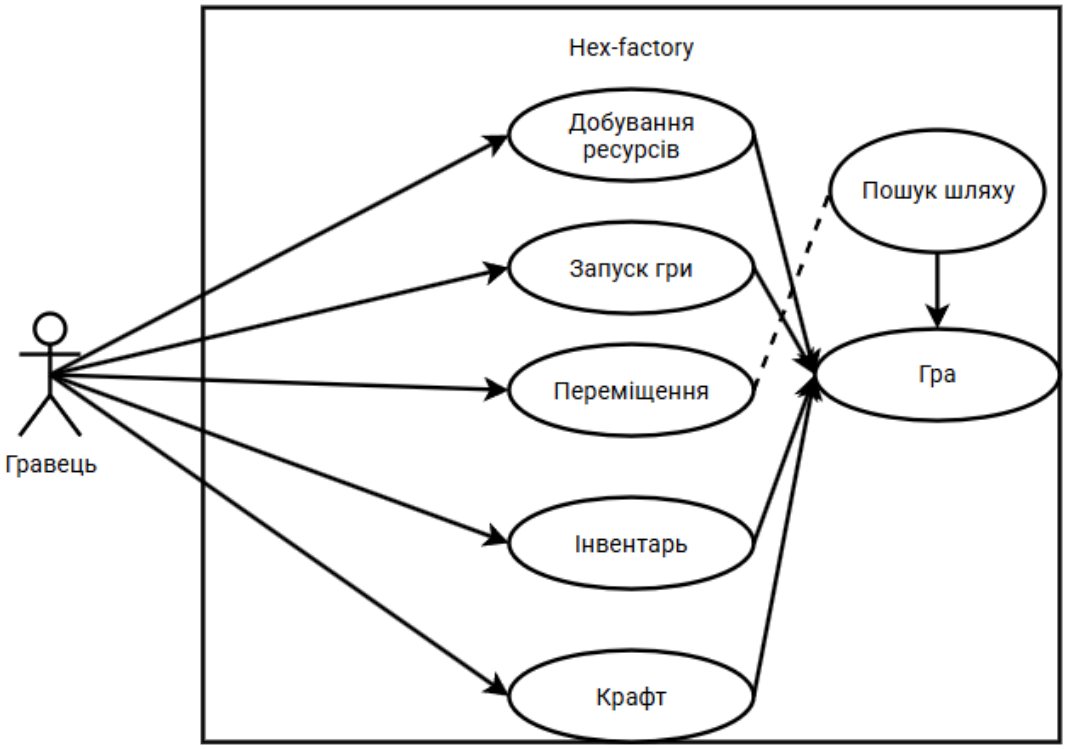

Рисунок 2.1 – Діаграма прецедентів (Use Case Diagram)

Джерело: розроблено автором

Діаграма прецедентів – це інструмент UML, який відображає взаємодію користувачів (акторів) із системою через певні сценарії використання (прецеденти) [10]. Наприклад, у грі система пропонує гравцеві запитання, а гравець на них відповідає. Якщо запитання відсутнє, система створює нове, аналізує відповідь і, за необхідності, підвищує складність наступного

запитання [11]. За правильну відповідь гравець отримує бали. Такий підхід дозволяє чітко описати поведінку системи та її реакцію на дії користувача.

Діаграма послідовності (рис. 2.2) відображає взаємодії об'єктів впорядкованих за часом [12].

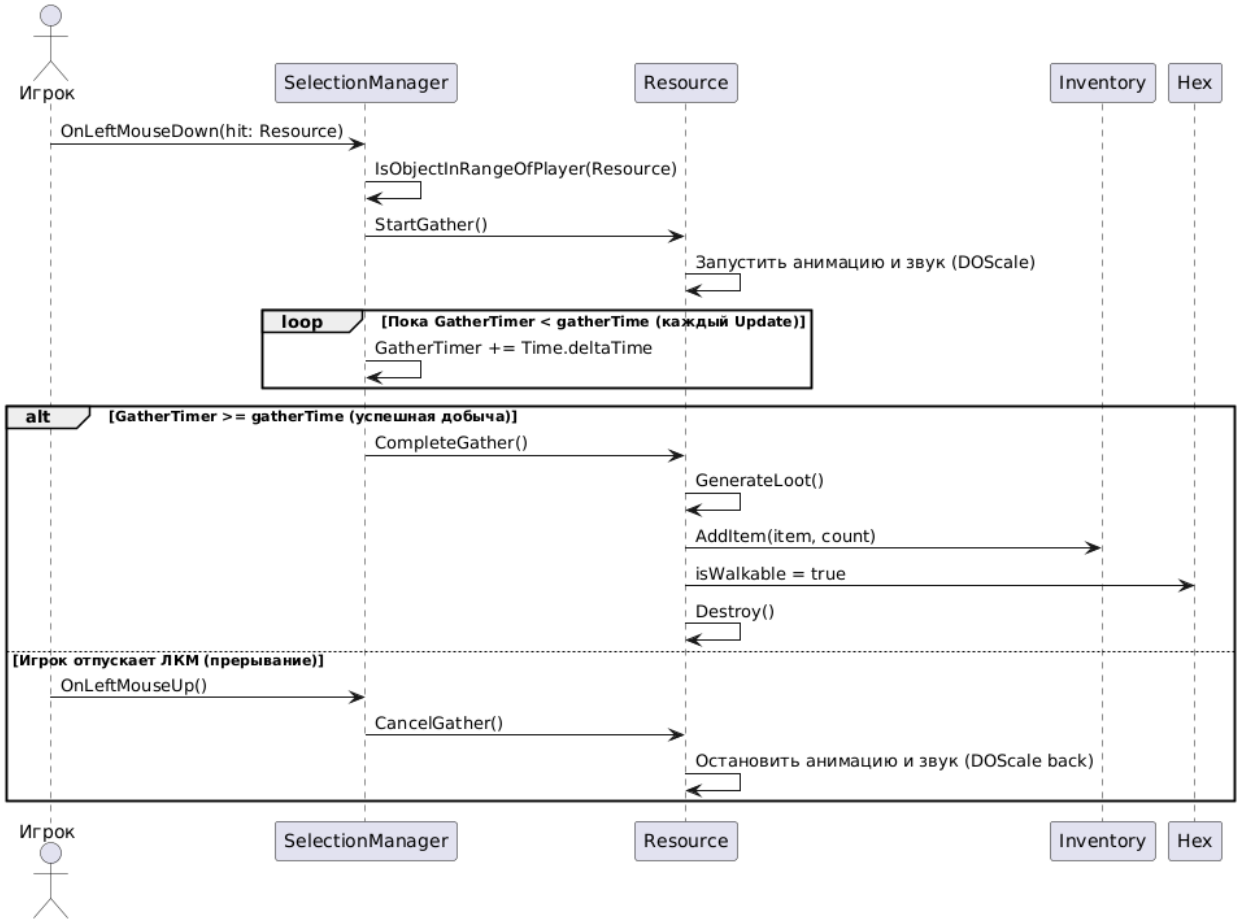

Рисунок 2.2 – Діаграма послідовності для добичі ресурсів (Sequence Diagram)

Джерело: розроблено автором

Запропоновані діаграми візуалізують структуру взаємодії об'єктів та послідовність операцій під час збору ресурсів у грі [13]. На прикладі механіки збирання демонструється трансформація геймплейних дій у системні виклики методів.

Основний метод workflow:

1. Ініціація дії: ліво-клавішний клік гравця на ресурс генерує подію

OnLeftMouseDown, яка передає координати до SelectionManager.

2. Валідація умов: система перевіряє дистанцію до ресурсу. При успіху активується StartGather(), що:
   2.1. Запускає анімацію збору
   2.2. Відтворює відповідний звуковий супровід
3. Прогрес виконання:
   3.1. У циклі оновлення кадрів інкрементується GatherTimer
   3.2. При досягненні порогового значення викликається каскад методів:
      3.2.1. CompleteGather() – фіксує завершення процесу
      3.2.2. GenerateLoot() – генерує предмети
      3.2.3. InventoryManager.AddItem() – інтегрує здобич у інвентар
      3.2.4. Оновлює стан гекса для подальшого переміщення

Сценарій переривання: якщо гравець відпускає кнопку (OnLeftMouseUp) до завершення часу:

1. Відбувається скид GatherTimer
2. Зупиняються медіа-ефекти
3. Ресурс залишається активним для повторного використання

Діаграми підкреслюють як high-level геймдизайн (взаємодія гравця з оточенням), так і low-level імплементацію (ланцюжки викликів, стан об'єктів, управління подіями). Особливу увагу приділено часовим залежностям та обробці користувацьких інпутів.

Діаграма діяльності (рис. 2.3) це візуальний інструмент який представляє графу діяльносте та відображає послідовність дій та процесів с системі, акцентуючи увагу на потоці управління між активностями [15]. В середовищі по створенням відео ігор во описує механіку процесі збору ресурсів та ілюструє взаємодію між гравцем та системою гри.

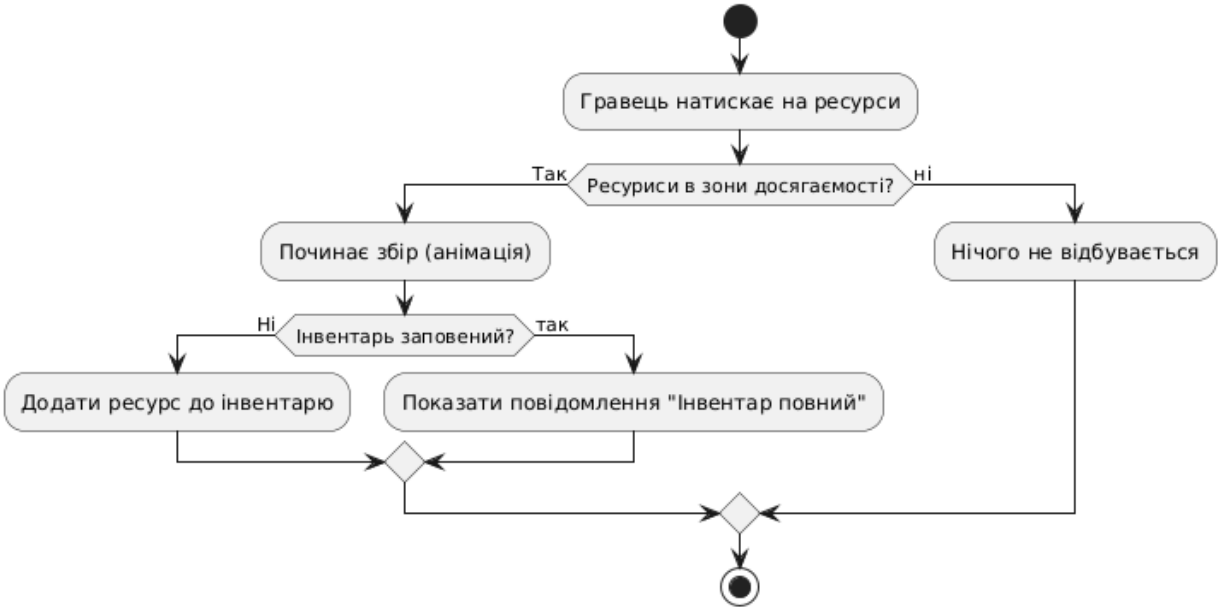

Рисунок 2.3 – Діаграма діяльності добування ресурсів (Activity Diagram)

Джерело: розроблено автором

Заборонована діаграма показує як працює система добування ресурсів. Основний процес добування ресурсів складається з таких етапів:

1. Початок дії: гравець натискає ліву кнопку миші на об’єкт ресурсу, запускаючи подію OnLeftMouseDown. Це ініціює процес збору.

2. Перевірка умов: система оцінює, чи доступний ресурс для збору (наприклад, чи перебуває він у зоні досяжності гравця). Якщо умови виконані, процес триває; якщо ні – дія припиняється, і гравець отримує повідомлення про помилку.

3. Виконання збору:

- запускається анімація процесу збору ресурсу.
- відтворюється звуковий ефект, що підтверджує дію.
- лічильник часу (GatherTimer) відстежує прогрес збору.
- після завершення таймера зібрані ресурси додаються до інвентарю гравця.

4. Завершення процесу:

- ігровий світ оновлюється: ресурс зникає або змінюється його стан (наприклад, дерево перетворюється на пень).
- система повертається до початкового стану, готова до наступної дії.

Якщо гравець відпускає кнопку миші (OnLeftMouseUp) до завершення збору:

1. Процес зупиняється.
2. Лічильник часу скидається.
3. Анімація та звукові ефекти припиняються.
4. Ресурс залишається доступним для повторної спроби.

2.2 Моделювання даних

Діаграми класів – структурна діаграма мови моделювання UML, що демонструє загальну структуру ієрархії класів системи, їх кооперацій, атрибутів, методів, інтерфейсів та взаємозв'язків між ними [13].

На зображені (рис 2.4) зображені класи Hex, HexGrid, PlayerMovement вони відповідають за карту та переміщення.

Клас Hex – представляє одиничний гекс на карті гри.

Атрибути:

- coordinates: Vector2Int – координати гекса в двовимірній сітці.
- isWalkable: bool – прапорець, що вказує, чи можна пройти по цьому гексу.
- occupyingObject: GameObject – посилання на об’єкт, який займає гекс (наприклад, будівлю або ресурс).
- *Методи:*
- Equals(obj: object): bool – перевіряє рівність гексів за координатами.
- GetHashCode(): int – повертає хеш-код, заснований на координатах.

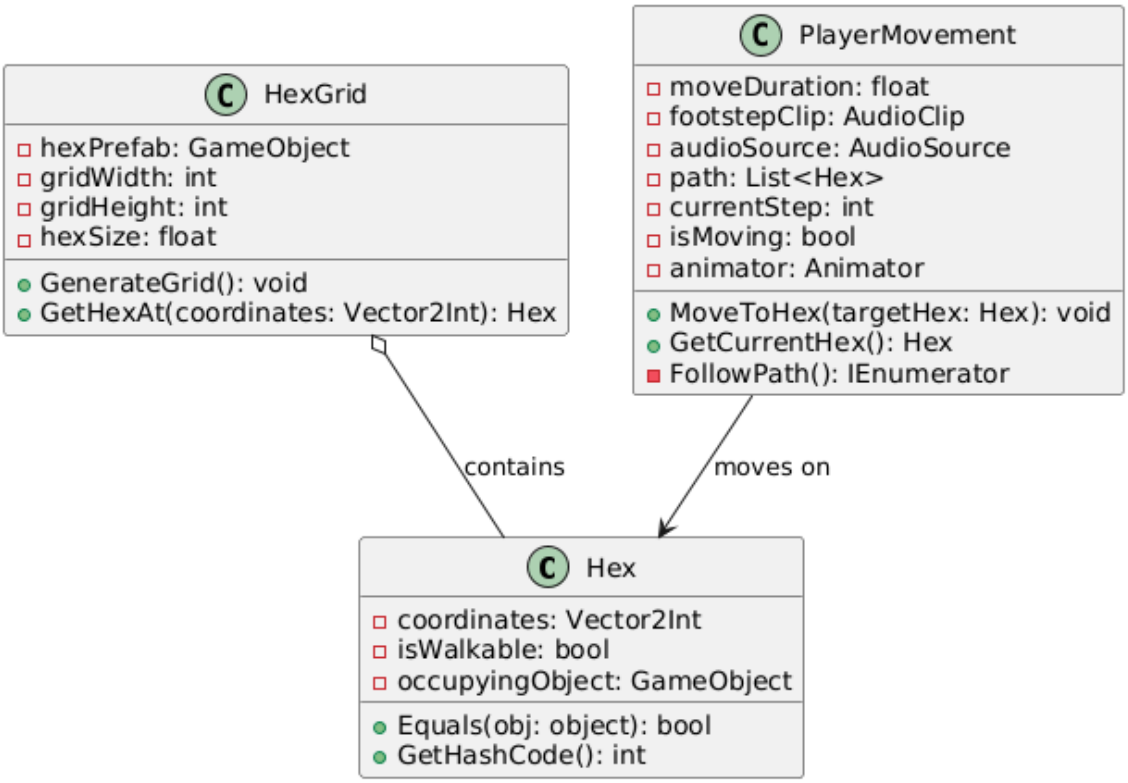

Рисунок 2.4 – Діаграма класів (Class Diagram) карта і переміщення

Джерело: розроблено автором

Hex інкапсулює базові властивості клітинки гексагональної сітки, дозволяючи перевіряти її прохідність і зберігати посилання на зайняті нею об’єкти.

Взаємодіє:

- клас HexGrid містить множину екземплярів Hex (агрегація).
- PlayerMovement переміщується по об’єктам типу Hex.

Клас HexGrid – відповідає за генерацію та управління гексагональною сіткою.

Атрибути:

- hexPrefab: GameObject – префаб для створення візуального представлення гекса.
- gridWidth: int – кількість гексів по горизонталі.
- gridHeight: int – кількість гексів по вертикалі.

- hexSize: float – розмір одного гекса у світових координатах.

Методи:

- GenerateGrid(): void – створює сітку гексів за заданими параметрами.
- GetHexAt(coordinates: Vector2Int): Hex – повертає посилання на гекс за його координатами або null, якщо він не існує.

Опис: HexGrid генерує двовимірну колекцію гексів і зберігає їх у внутрішньому словнику для швидкого доступу за координатами.

Клас PlayerMovement – реалізує переміщення персонажа по гексах із анімацією та звуковими ефектами.

- moveDuration: float – час переміщення між сусідніми гексами.
- footstepClip: AudioClip – аудіокліп звуку кроків.
- audioSource: AudioSource – компонент для відтворення звуків.
- path: List<Hex> – список гексів, що становлять маршрут руху.
- currentStep: int – індекс поточної цілі в маршруті.
- isMoving: bool – прапорець, чи відбувається зараз переміщення.
- animator: Animator – компонент для керування анімацією персонажа.

Методи:

- MoveToHex(targetHex: Hex): void – запускає пошук шляху та корутину руху до вказаного гекса.
- GetCurrentHex(): Hex – повертає гекс, на якому зараз знаходиться персонаж.
- FollowPath(): IEnumerator – внутрішня корутина, що поетапно переміщує персонажа вздовж маршруту.

Опис: PlayerMovement використовує алгоритм A* для побудови маршруту між гексами, а потім анімує рух персонажа з відтворенням звуку кроків.

На зображені (рис 2.5) зображені класи Resource, Generator, IEnergyConsumer, Auger, ConveyorBelt , Crate вони управління ресурсами.

Клас Resource – керує процесом збору ресурсів у грі.

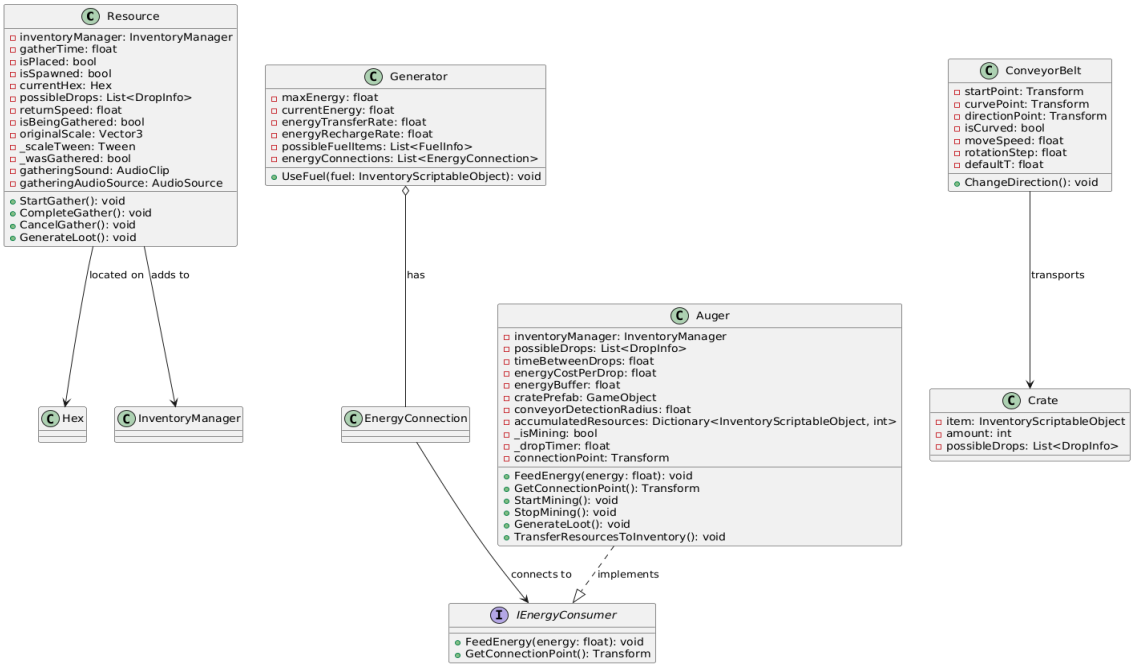

Рисунок 2.5 – Діаграма класів (Class Diagram) управління ресурсами

Джерело: розроблено автором

Атрибути:

- inventoryManager: InventoryManager – посилання на менеджер інвентаря для додавання добутих предметів.
- gatherTime: float – час, необхідний для завершення збору ресурсу.
- isPlaced: bool – чи був ресурс розміщений гравцем.
- isSpawned: bool – чи був ресурс спавнений системою.
- currentHex: Hex – гекс, на якому знаходиться ресурс.
- possibleDrops: List<DropInfo> – список інформації про можливі предмети, що видаються при зборі.
- returnSpeed: float – швидкість повернення масштабу при скасуванні збору.
- isBeingGathered: bool – прапорець активності процесу збору.
- originalScale: Vector3 – початковий розмір об’єкта.

- _scaleTween: Tween – Tween-анімація для зміни масштабу.
- _wasGathered: bool – прапорець, чи був ресурс зібраний.
- gatheringSound: AudioClip – звук процесу збору.
- gatheringAudioSource: AudioSource – джерело звуку для процесу збору.

Методи:

- StartGather(): void – ініціює анімацію та звук збору.
- CompleteGather(): void – завершує процес, генерує лут та видаляє об’єкт.
- CancelGather(): void – скасовує збір та повертає об’єкт до початкового стану.
- GenerateLoot(): void – обчислює та додає предмети до інвентаря за шансом.

Опис: Resource відповідає за життєвий цикл об’єкта-ресурсу: від анімації руйнування до генерації добутих предметів.

Взаємодіє:

Resource розташований на Hex.

Після генерації добутку викликає InventoryManager для додавання предметів.

Клас Generator – джерело енергії для інших пристроїв.

Атрибути:

- maxEnergy: float – максимальний запас енергії.
- currentEnergy: float – поточний рівень енергії.
- energyTransferRate: float – швидкість передачі енергії споживачам.
- energyRechargeRate: float – швидкість відновлення енергії.
- possibleFuelItems: List<FuelInfo> – список типів палива.
- energyConnections: List<EnergyConnection> – список з’єднань із споживачами.

Методи:

- UseFuel(fuel: InventoryScriptableObject): void – додає енергію з вказаного палива.

Опис: Generator акумулює енергію та розподіляє її між підключеними споживачами через об’єкти EnergyConnection.

Клас Auger – бурова установка для автоматичного видобутку ресурсів.

Атрибути:

- inventoryManager: InventoryManager – посилання на менеджер інвентаря.
- possibleDrops: List<DropInfo> – перелік можливих добутих предметів.
- timeBetweenDrops: float – інтервал між створенням одиниць ресурсу.
- energyCostPerDrop: float – вартість енергії за одиницю видобутку.
- energyBuffer: float – накопичений запас енергії.
- cratePrefab: GameObject – префаб ящика для збереження ресурсів.
- conveyorDetectionRadius: float – радіус пошуку конвеєра.
- accumulatedResources: Dictionary<InventoryScriptableObject, int> – накопичені ресурси.
- _isMining: bool – чи відбувається зараз буріння.
- _dropTimer: float – внутрішній таймер для генерації.
- connectionPoint: Transform – точка підключення кабелю.

Методи:

- FeedEnergy(energy: float): void – приймає енергію від генератора.
- GetConnectionPoint(): Transform – повертає точку підключення.
- StartMining(): void – запускає процес видобутку.
- StopMining(): void – зупиняє видобуток.
- GenerateLoot(): void – створює ресурси на основі шансів.
- TransferResourcesToInventory(): void – передає накопичені ресурси в інвентар.

Опис: Auger автоматизує збір ресурсів, споживаючи енергію та передаючи добуті предмети в інвентар.

Взаємодіє:

- реалізує інтерфейс IEnergyConsumer.
- підключається до Generator через EnergyConnection.

Клас ConveyorBelt – конвеєр для транспортування ящиків із ресурсами.

Атрибути:

- startPoint: Transform – початкова точка руху ящика.
- curvePoint: Transform – точка вигину траєкторії.
- directionPoint: Transform – точка напрямку руху.
- isCurved: bool – чи має конвеєр криволінійну ділянку.
- moveSpeed: float – швидкість руху ящиків.
- rotationStep: float – крок повороту при зміні напрямку.
- defaultT: float – початковий параметр для кривої Безьє.

Методи:

- ChangeDirection(): void – змінює напрямок руху на протилежний.

Опис: ConveyorBelt транспортує об’єкти Crate уздовж заданої траєкторії, підтримуючи прямі та криві ділянки.

Взаємодіє:

- Транспортує Crate уздовж маршруту.

Клас Crate – контейнер для ресурсів на конвеєрі.

Атрибути:

- item: InventoryScriptableObject – тип ресурсу.
- amount: int – кількість одиниць ресурсу.
- possibleDrops: List<DropInfo> – інформація про можливі бонусні дропи.

Опис: Crate зберігає ресурси та передає їх наступним обробникам на конвеєрі.

Інтерфейс IEnergyConsumer – контракт для класів, що споживають енергію.

Методи:

- FeedEnergy(energy: float): void
- GetConnectionPoint(): Transform

Опис:

Інтерфейс визначає методи, які повинен реалізувати будь-який клас-споживач енергії.

На зображені (рис 2.6) зображені класи InventoryManager, InventorySlot, InventoryScriptableObject інвентар і предмети.

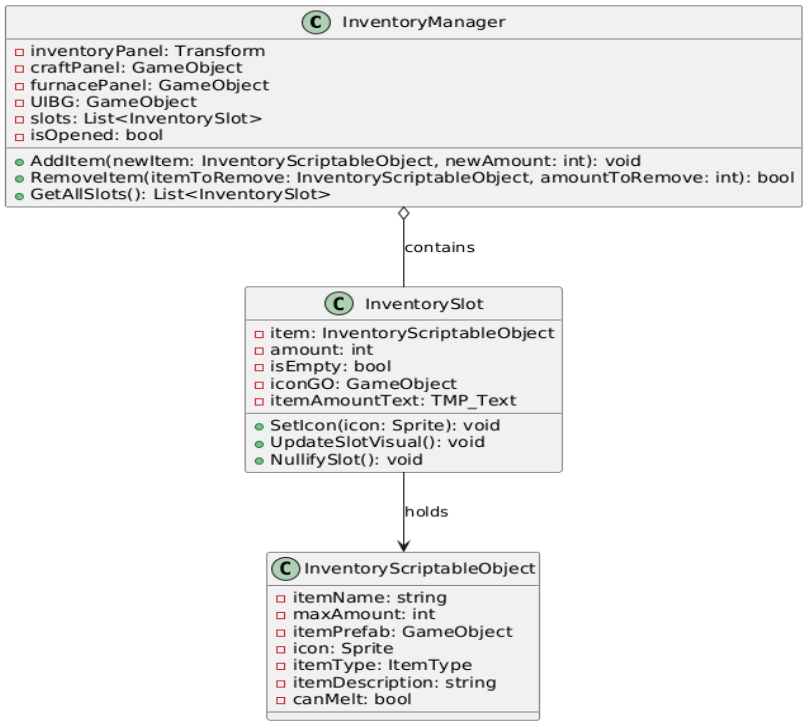

Рисунок 2.6 – Діаграма класів (Class Diagram) інвентар і предмети

Джерело: розроблено автором

Клас InventoryManager – керує основним інвентарем та UI.

Атрибути:

- inventoryPanel: Transform – контейнер UI-слотів інвентаря.
- craftPanel: GameObject – панель крафту.
- furnacePanel: GameObject – панель печі.

- UIBG: GameObject – фон UI інвентаря.
- slots: List<InventorySlot> – список слотів.
- isOpened: bool – чи відкритий інвентар.

Методи:

- AddItem(newItem: InventoryScriptableObject, newAmount: int): void – додає предмет у слот або об’єднує з існуючим.
- RemoveItem(itemToRemove: InventoryScriptableObject, amountToRemove: int): bool – видаляє вказану кількість предметів.
- GetAllSlots(): List<InventorySlot> – повертає список усіх слотів, включаючи швидкі.

Опис: InventoryManager реалізує патерн Singleton для централізованого управління предметами та відображення UI.

Взаємодіє:

- Містить InventorySlot.

Клас InventorySlot – представляє один слот інвентаря.

Атрибути:

- item: InventoryScriptableObject – предмет, що зберігається в слоті.
- amount: int – кількість предметів.
- isEmpty: bool – чи є слот порожнім.
- iconGO: GameObject – GameObject для відображення іконки.
- itemAmountText: TMP_Text – текстове поле для відображення кількості.

Методи:

- SetIcon(icon: Sprite): void – встановлює спрайт іконки.
- UpdateSlotVisual(): void – оновлює відображення слота.
- NullifySlot(): void – очищує слот від предметів.

Опис: InventorySlot відповідає за візуалізацію та внутрішній стан одного слота.

Взаємодіє:

Зберігає посилання на InventoryScriptableObject.

Клас InventoryScriptableObject – дані предмета для інвентаря.

Атрибути:

- itemName: string – назва предмета.
- maxAmount: int – максимальна кількість в одному слоті.
- itemPrefab: GameObject – префаб для розміщення в світі.
- icon: Sprite – іконка предмета.
- itemType: ItemType – категорія предмета.
- itemDescription: string – опис предмета.
- canMelt: bool – чи може предмет плавитися.

Опис: InventoryScriptableObject слугує контейнером даних для конфігурації предметів у грі.

На зображені (рис 2.7) зображені класи CraftingManager, FurnaceManager крафт і плавка.

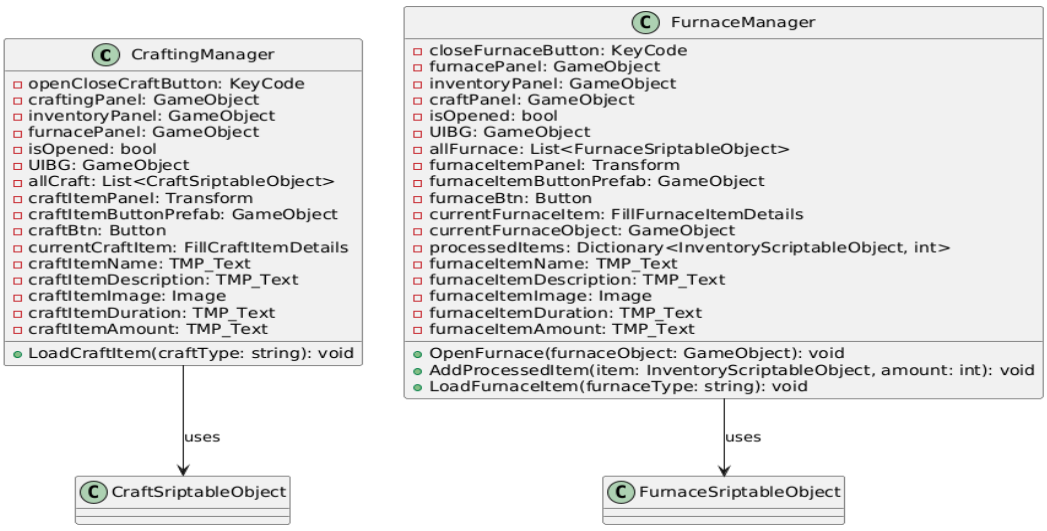

Рисунок 2.7 – Діаграма класів (Class Diagram) крафт і плавка

Джерело: розроблено автором

Клас CraftingManager – керує інтерфейсом крафту та створенням предметів.

Атрибути:

- openCloseCraftButton: KeyCode – клавіша для відкриття/закриття UI крафту.
- craftingPanel: GameObject – панель крафту.
- inventoryPanel: GameObject – панель інвентаря.
- furnacePanel: GameObject – панель печі.
- isOpened: bool – чи відкритий інтерфейс крафту.
- UIBG: GameObject – фон UI.
- allCraft: List<CraftSriptableObject> – список рецептів.
- craftItemPanel: Transform – контейнер для кнопок рецептів.
- craftItemButtonPrefab: GameObject – префаб кнопки рецепту.
- craftBtn: Button – кнопка для запуску крафту.
- currentCraftItem: FillCraftItemDetails – компонент для відображення деталей обраного рецепту.
- craftItemName: TMP_Text – назва рецепту.
- craftItemDescription: TMP_Text – опис рецепту.
- craftItemImage: Image – зображення предмета.
- craftItemDuration: TMP_Text – час крафту.
- craftItemAmount: TMP_Text – кількість результату.

Методи:

- LoadCraftItem(craftType: string): void – завантажує дані вибраного рецепту в UI.

Опис: CraftingManager відображає список рецептів, дозволяє обирати їх і запускати процес створення предметів.

Взаємодіє: використовує CraftSriptableObject для завантаження рецептів.

Клас FurnaceManager – керує плавкою в печі та отриманням результатів.

Атрибути:

- closeFurnaceButton: KeyCode – клавіша для закриття UI печі.
- furnacePanel: GameObject – панель печі.

- inventoryPanel: GameObject – панель інвентаря.
- craftPanel: GameObject – панель крафту.
- isOpened: bool – чи відкритий інтерфейс печі.
- UIBG: GameObject – фон UI.
- allFurnace: List<FurnaceSriptableObject> – список рецептів плавки.
- furnaceItemPanel: Transform – контейнер кнопок плавки.
- furnaceItemButtonPrefab: GameObject – префаб кнопки.
- furnaceBtn: Button – кнопка для початку плавки.
- currentFurnaceItem: FillFurnaceItemDetails – відображає деталі обраного рецепту.
- currentFurnaceObject: GameObject – поточна печі у сцені.
- processedItems: Dictionary<InventoryScriptableObject, int> – словник оброблених предметів.
- furnaceItemName: TMP_Text – назва плавильного рецепту.
- furnaceItemDescription: TMP_Text – опис рецепту.
- furnaceItemImage: Image – зображення результату.
- furnaceItemDuration: TMP_Text – час плавки.
- furnaceItemAmount: TMP_Text – кількість готового предмета.

Методи:

- OpenFurnace(furnaceObject: GameObject): void – відкриває UI для заданої печі.
- AddProcessedItem(item: InventoryScriptableObject, amount: int): void – додає результат плавки в інвентар.
- LoadFurnaceItem(furnaceType: string): void – завантажує дані обраного рецепту.

Опис: FurnaceManager керує інтерфейсом печі, дозволяє завантажувати інгредієнти та отримувати розплавлені матеріали.

Взаємодіє:

Використовує FurnaceSriptableObject для отримання даних рецептів.

На зображені (рис 2.8) зображені класи SelectionManager інтерфейс і управління станом.

Рисунок 2.8 – Діаграма класів (Class Diagram) інтерфейс і управління станом

Джерело: розроблено автором

Клас SelectionManager – координує введення гравця, вибір об’єктів і взаємодію зі світом.

Атрибути:

- furnaceUI: GameObject – UI печі.
- inventoryManager: InventoryManager – менеджер інвентаря.
- craftingManager: CraftingManager – менеджер крафту.
- furnaceManager: FurnaceManager – менеджер печі.
- quickslotInventory: QuickslotInventory – швидкі слоти інвентаря.
- ghostMaterial: Material – матеріал для прев’ю об’єктів.
- jumpHeight: float – висота анімації вибору персонажа.

- jumpDuration: float – тривалість анімації.
- jumpEase: Ease – тип інтерполяції для стрибка.
- selectedGenerator: Generator – обраний генератор для підключення.
- wirePrefab: GameObject – префаб дроту для візуалізації з’єднання.
- IsGatheringActive: bool – чи триває збір ресурсу.
- ActiveResource: Resource – поточний ресурс для збору.
- GatherTimer: float – таймер збору.
- _selectedCharacter: GameObject – вибраний персонаж.
- _selectionTween: Tween – анімація вибору.
- _ghostObject: GameObject – прев’ю об’єкта перед розміщенням.
- _currentHexUnderMouse: Hex – гекс під курсором.
- _rotationIndex: int – індекс повороту прев’ю.
- currentState: IInputState – поточний стан вводу.

Методи:

- SelectCharacter(character: GameObject): void – вибір персонажа та запуск анімації.
- MoveCharacterToHex(targetHex: Hex): void – переміщення вибраного персонажа.
- TryStartResourceGathering(resource: Resource): void – початок збору ресурсу.
- CancelResourceGathering(): void – скасування збору.
- ResetGatherState(): void – скидання стану збору.
- DecreaseSlotAmount(): void – зменшення кількості в швидкому слоті.
- IsObjectInRangeOfPlayer(obj: GameObject): bool – перевірка досяжності об’єкта.
- IsHexInRangeOfPlayer(targetHex: Hex): bool – перевірка досяжності гекса.
- GetNeighbors(hex: Hex): List<Hex> – отримання сусідніх гексів.
- IsAnyMenuOpened(): bool – перевірка, чи відкритий якийсь UI.

- ConnectGeneratorAndAuger(generator: Generator, auger: Auger): void – встановлення з’єднання між генератором і буром.

Опис: SelectionManager реалізує машину станів вводу (Idle та Placement), обробляє кліки миші для вибору персонажа, руху, збору ресурсів та розміщення об’єктів.

Взаємодіє:

- Керує PlayerMovement, взаємодіє з Resource, використовує InventoryManager, CraftingManager, FurnaceManager та QuickslotInventory.

2.3 Моделювання процесів

У даній роботі основним алгоритмом є A* (A-зірка) – це один із найпоширеніших алгоритмів пошуку шляху, який використовується в комп’ютерних іграх для знаходження найкоротшого маршруту між двома точками на карті з урахуванням перешкод [16]. Він поєднує підхід алгоритму Дейкстри, що гарантує оптимальність, із жадібним пошуком, який прискорює обчислення завдяки евристичній функції. Евристика оцінює приблизну відстань до цілі, що дозволяє алгоритму ефективно спрямовувати пошук.

У комп’ютерних іграх A* часто застосовується для навігації персонажів або об’єктів, особливо на картах із нетривіальною структурою, таких як гексагональні сітки. У грі "Hex-factory" алгоритм A* (рис 2.9) використовується для переміщення персонажа по гексагональній карті, дозволяючи йому обходити непрохідні зони, наприклад, будівлі чи інші перешкоди, і знаходити оптимальний шлях до цільової точки.

```
private static List<Hex> GetNeighbors(Hex hex)
{
    List<Hex> neighbors = new List<Hex>();
    Vector2Int currentCoords = hex.coordinates;
    Vector2Int[] offsets = (currentCoords.y % 2 == 0) ? evenRowOffsets : oddRowOffsets;

    foreach (var offset :Vector2Int in offsets)
    {
        Vector2Int neighborCoords = currentCoords + offset;
        Hex neighbor = HexGrid.GetHexAt(neighborCoords);
        if (neighbor != null)
        {
            neighbors.Add(neighbor);
        }
    }

    return neighbors;
}
```

Рисунок 2.9 – Частина алгоритму A(А зірочка)*

Джерело: розроблено автором

Принцип роботи алгоритму A* базується на концепції вузлів (позицій на карті) та ребер (зв’язків між ними). Кожен вузол оцінюється за формулою (1).

$$f(n)=g(n)+h(n) \qquad (1)$$

де:

g(n) – вартість шляху від початкової точки до поточного гексагона.

h(n) – евристична оцінка відстані від поточного гексагона до цілі.

Оскільки правильний гексагон має однакову довжину сторін, а також радіус вписаного та описаного кіл, відстань між центрами всіх сусідніх гексагонів є сталою. Це значно спрощує обчислення та дозволяє ефективно реалізувати пошук шляху, оскільки всі переходи між сусідніми гексами мають однакову вартість (рис. 2.10).

```
foreach (Hex neighbor in GetNeighbors(current))
{
    if (closedSet.Contains(neighbor) || !neighbor.isWalkable)
        continue;

    float tentativeGScore = gScore[current] + GetDistance(a: current, b: neighbor);
    if (!gScore.ContainsKey(neighbor) || tentativeGScore < gScore[neighbor])
    {
        cameFrom[neighbor] = current;
        gScore[neighbor] = tentativeGScore;
        fScore[neighbor] = gScore[neighbor] + GetHeuristic(a: neighbor, b: target);

        if (!openSet.Contains(neighbor))
            openSet.Add(neighbor);
    }
}
```

*Рисунок 2.10 – Використання алгоритму A**

Джерело: розроблено автором

Розглянемо ситуацію в грі "Hex-factory": гравець обирає точку на гексагональній карті, куди персонаж має дістатися. Алгоритм A* аналізує поточне положення персонажа та цільову точку, враховуючи прохідність кожного гекса (наприклад, атрибут isWalkable класу Hex). Якщо між початковою та кінцевою точками є непрохідні гекси, A* обчислює обхідний шлях, використовуючи евристику, таку як відстань між гексами.

Висновки до розділу 2

У результаті проектування структури гри Hex-factory було визначено чітку та масштабовану архітектуру, що поєднує гексагональну сітку з модульними компонентами для керування рухом, добуванням ресурсів, енергозабезпеченням та логістикою.

Реалізація UML-діаграм (прецедентів, послідовності, діяльності) дозволила формалізувати взаємодію гравця та системи на різних рівнях абстракції: від високорівневих сценаріїв використання до детальних

ланцюжків викликів методів. Це сприяє зрозумілому документуванню процесів та полегшує подальшу розробку.

Розглянуто алгоритм A* для навігації персонажів на картах із нетривіальною структурою.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ КОМПЮТЕРНОЇ ГРИ «HEX-FACTORY»

3.1 Реалізація та конструювання програмного забезпечення

Гра «Hex factory» розроблена на мультиплатформному ігровому рушії Unity Engine [7] з використанням мови програмування C# [6]. Такий вибір обумовлений тісною інтеграцією C# із Unity, її підтримкою об’єктно орієнтованого підходу та багатою екосистемою бібліотек і інструментів, що значно спрощує створення складних ігрових механік.

Основні інструменти та технології:

- C# – основна мова розробки. Завдяки чіткій структурі класів і компонентів, а також потужним засобам налагодження, C# ідеально підходить для реалізації ігрових систем;
- Unity Editor – візуальне середовище розробки, що забезпечує інструменти для створення сцен, налаштування фізики, анімацій і рендерингу. Його гнучкість дозволяє швидко тестувати ігрову логіку в реальному часі;
- цільова платформа – ПК (Windows). Обрана через широку аудиторію користувачів та можливість використовувати потужне апаратне забезпечення для складних ігрових процесів. У разі потреби Unity дозволяє легко портувати гру на macOS чи Linux;
- Unity Standard Libraries – стандартні бібліотеки рушія Unity, що забезпечують доступ до основних функцій: рендеринг, фізика, анімація, керування сценами тощо;
- DOTween [17] – зовнішня бібліотека для створення плавних анімацій і переходів. Використовується для покращення візуальних ефектів у грі, зокрема для анімацій збору ресурсів та переміщення об’єктів по конвеєру. DOTween забезпечує вищий рівень контролю, ніж стандартні інструменти Unity;

- Blender [19] – інструмент для створення та редагування 3D-моделей. Використовується для створення унікальних об’єктів, які потім імпортуються до Unity;
- Rider [18] – інтегроване середовище розробки (IDE), орієнтоване на мову C#. Воно має розширені можливості автодоповнення, інтелектуального аналізу коду та повну інтеграцію з Unity, що значно підвищує ефективність розробки;
- Meshy.ai [20] – онлайн-сервіс для генерації 3D-моделей на основі зображень або текстових описів. Дає змогу швидко отримувати якісні 3D-моделі без глибоких знань моделювання;
- Unity Asset Store [21] – офіційний маркетплейс для Unity, з якого використовувалися безкоштовні та платні ресурси (моделі, текстури, скрипти) для прискорення розробки та прототипування гри.

Обґрунтування вибору:

- C# – оптимальна мова для Unity, що підтримує ООП і добре підходить для створення ігрових компонентів;
- Unity Editor – потужний візуальний редактор з інструментами для швидкої розробки ігор;
- ПК (Windows) – надає найкращу продуктивність і охоплює найбільшу кількість гравців;
- DOTween – дозволяє реалізувати складні, динамічні та плавні анімаційні ефекти;
- Rider – сучасне IDE з глибокою інтеграцією з Unity, яке полегшує написання, рефакторинг і налагодження коду;
- Blender – безкоштовний і гнучкий інструмент для 3D-моделювання;
- Meshy.ai – сучасне AI-рішення для швидкого створення 3D-контенту;

- Unity Asset Store – економить час, дозволяє зосередитися на унікальному геймплеї, а не рутинних задачах.

Реалізація гри розпочалася з дослідження гексагональної сітки та побудови гексагональної мапи. Правильний гексагон можна охарактеризувати як багатокутник, сторони якого дотикаються до вписаного та описаного кругу. Існують два основних типи орієнтації гексагонів: з вершиною, спрямованою догори (загострений верх), та з горизонтальною верхньою гранню (тупий верх, рис 3.1).

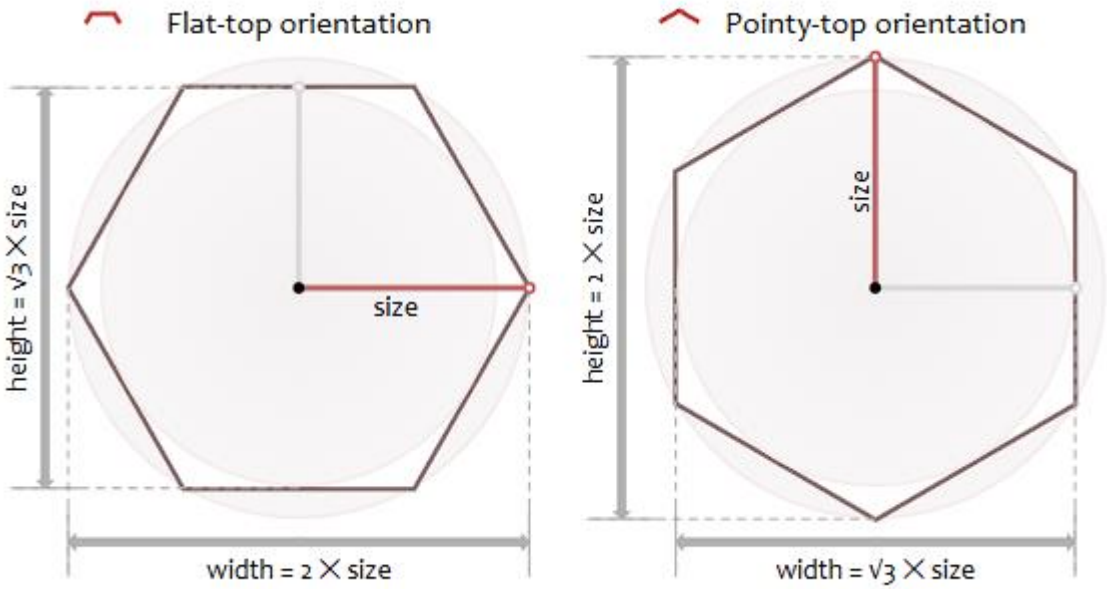

Рисунок 3.1 – Види гексагонів

Джерело: [9]

Для реалізації гри було обрано гексагон із загостреним верхом (pointy-top orientation). Завдяки використанню звичайного двовимірного масиву вдалося ефективно побудувати гексагональну карту, що лягла в основу ігрового світу. Вибір саме гексагону з гострим верхом він є більше естетичним що підходить під задумку гри (рис. 3.2).

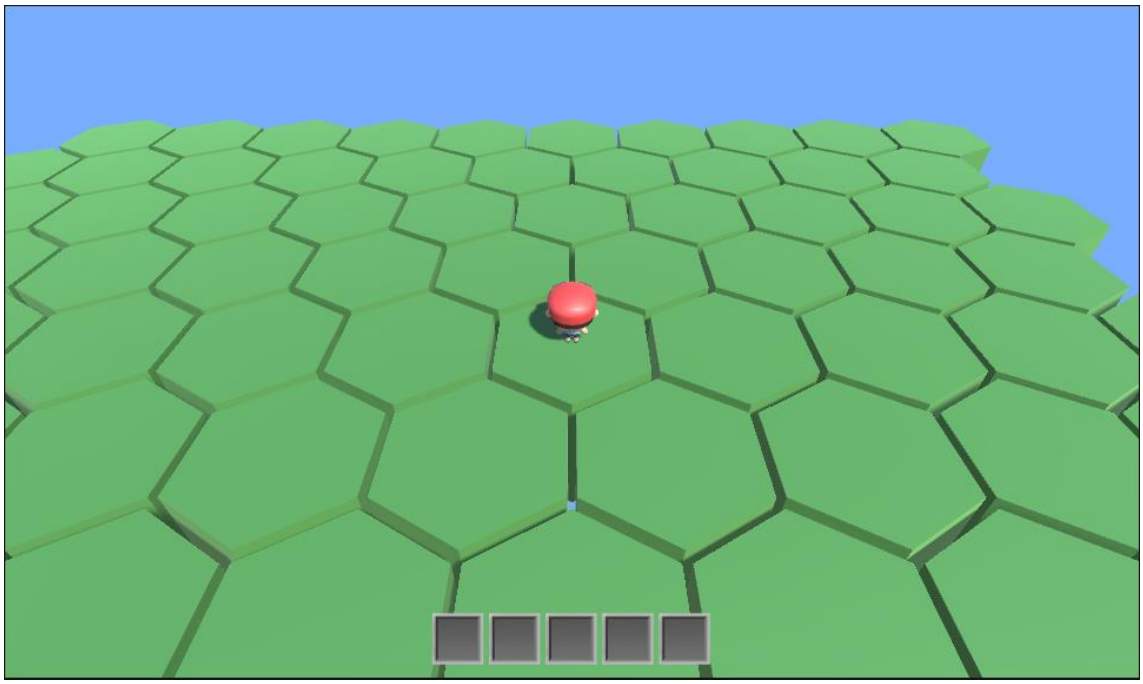

Рисунок 3.2 – Демонстрація в грі гексагоної мапи

Джерело: розроблено автором

Для реалізації переміщення по гексагональній мапі було використано алгоритм A* (A-зірка) – один із найефективніших алгоритмів для пошуку шляху в графах, який широко застосовується в розробці ігор. Принцип цього алгоритму був розказаний в пункті 2.3.

Так як персонаж вже вміє знаходити ефективний шлях за допомогою алгоритму A*, наступним етапом стала реалізація його переміщення по цьому шляху. Для цього було використано бібліотеку DOTween, яка надає зручні інструменти для анімації об’єктів безпосередньо у коді. Зокрема, DOTween дозволяє переміщувати об’єкти з однієї позиції в іншу з плавною анімацією за заданий проміжок часу. Завдяки цьому персонаж тепер може не просто миттєво “телепортуватися”, а природно рухатися по гексагональній сітці згідно з побудованим маршрутом (рис. 3.3).

```
while (currentStep < path.Count)
{
    Hex targetHex = path[currentStep];
    Vector3 targetPosition = targetHex.transform.position;

    Vector3 direction = (targetPosition - transform.position).normalized;
    float targetAngle = Mathf.Atan2( y: direction.x,  x: direction.z) * Mathf.Rad2Deg;

    // Повернути персонажа
    transform.DORotateQuaternion( endValue: Quaternion.Euler( x: 0,  y: targetAngle,  z: 0),  duration: moveDuration / 2);

    // Перемістити персонажа
    transform.DOMove(targetPosition, moveDuration).SetEase(Ease.Linear);

    yield return new WaitForSeconds(moveDuration);

    currentStep++;
}
```

Рисунок 3.3 – Переміщення персонажу

Джерело: розроблено автором

Наступним кроком були перешкоди, а саме 3D-моделі ресурсів (рис. 3.4, дерева, каміння тощо), були частково завантажені з Unity Asset Store, а частково згенеровані за допомогою штучного інтелекту Meshy.ai. Для забезпечення взаємодії з цими об'єктами (добування ресурсів, взаємодія з об’єктами, взаємодія з персонажем, «Placement Preview» (рис. 3.5) був реалізований скрипт, що працює за шаблоном партера "Стан" (State Pattern).

Патерн State [23] дозволяє об’єкту змінювати свою поведінку залежно від його внутрішнього стану. Замість того, щоб використовувати великі блоки умовних операторів (if/else або switch), логіка взаємодії розподіляється між окремими класами – кожен з яких відповідає за конкретний стан (рис. 3.6).

Наступним кроком у розробці гри стало створення системи інвентаря (рис. 3.7). Завдяки вбудованим UI-компонентам Unity, зокрема Grid Layout Group, реалізація базової структури інвентаря відбулася швидко та ефективно. Цей компонент дозволяє автоматично розміщувати елементи (слоти) у вигляді сітки, забезпечуючи адаптивність і зручність масштабування.

Рисунок 3.4 – 3D моделі які використовуються в грі

Джерело: [24]

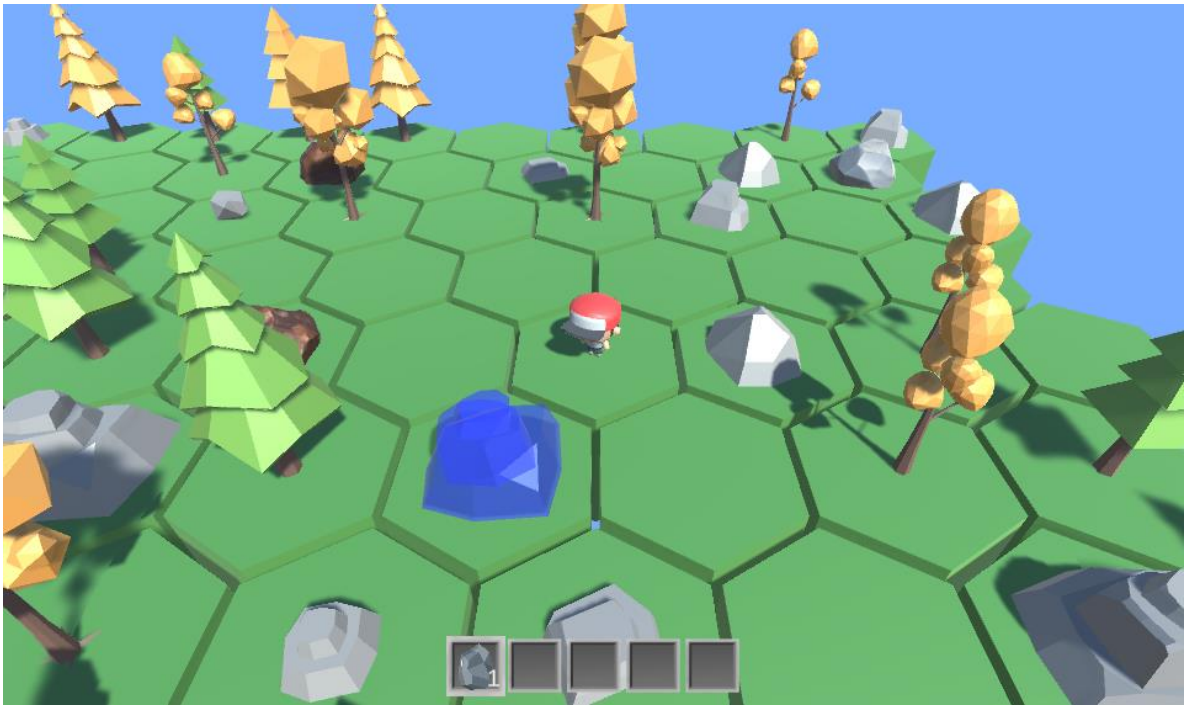

Рисунок 3.5 – Демонстрація Placement Preview

Джерело: розроблено автором

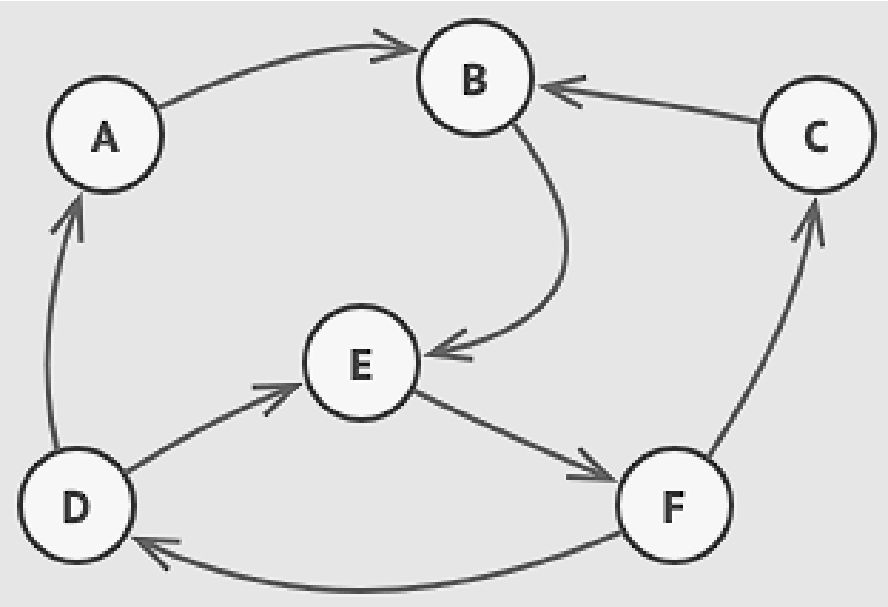

Рисунок 3.6 – Принцип роботи патерна «Стан»

Джерело: [23]

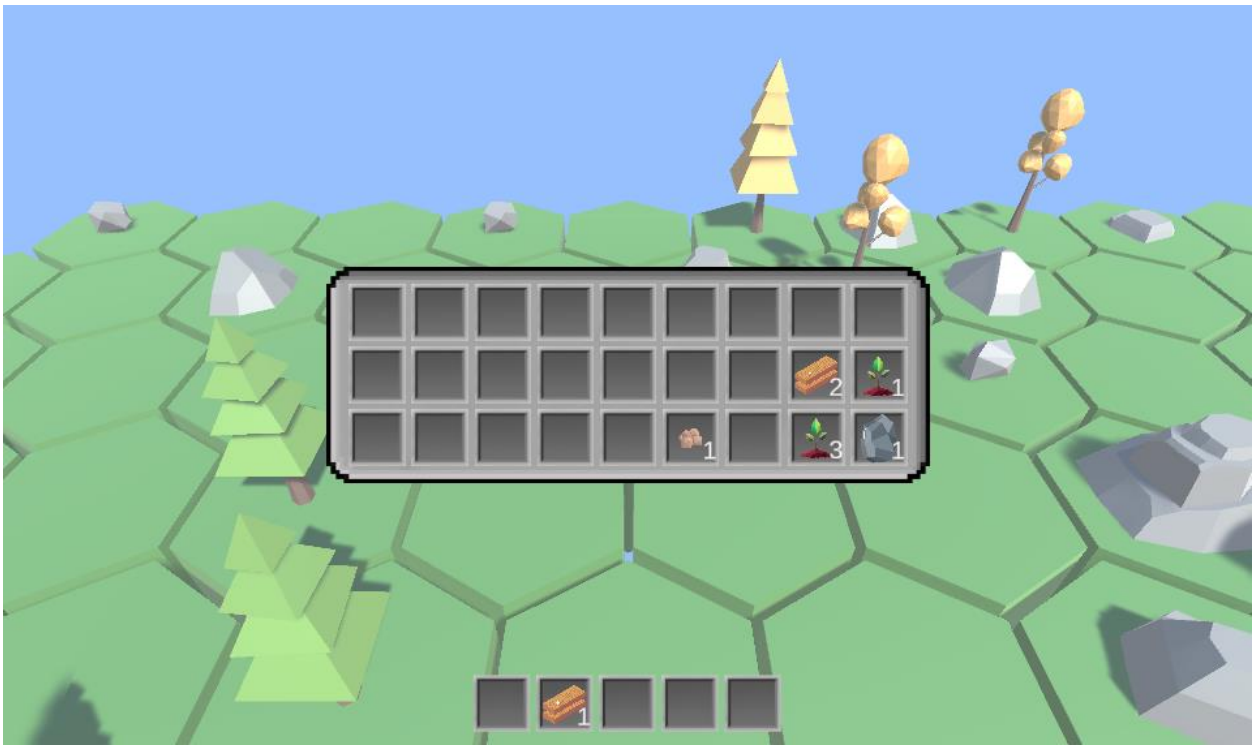

Рисунок 3.7 – Інтерфейс інвентаря

Джерело: розроблено автором

Для управління всією системою інвентаря було реалізовано окремий клас InventoryManager, який використовує шаблон проєктування Singleton (Одинак).

Патерн Singleton (Одинак)[25] – це шаблон проєктування, який гарантує, що у програмі існує тільки один екземпляр класу, до якого можна глобально звертатися з будь-якої частини коду. Він часто використовується для керування ресурсами, налаштуваннями або, як у цьому випадку – глобальними системами гри (інвентар, звук, UI тощо).

Наступним кроком у розробці гри стало створення системи крафта та печі (рис. 3.8). Ці системи майже ідентичні, тому їх UI реалізовано однаково – за допомогою Grid Layout Group та префабів кнопок і панелей, що дозволяє швидко масштабувати й налаштовувати макет без додаткового коду.

Для опису рецептів було використано ScriptableObject, що реалізує шаблон Prototype: кожний рецепт (CraftSriptableObject) зберігає дані про кінцевий предмет, необхідні ресурси й час виготовлення, тож додаючи новий рецепт у проєкт, не потрібно змінювати код

Для управління логікою крафта створено клас CraftingManager, який відповідає за:

- відкриття/закриття панелі крафта;
- динамічне завантаження кнопок рецептів у craftItemPanel;
- реакцію на натискання клавіш та оновлення UI.

Чергу крафта реалізує CraftQueueManager, що перевіряє наявність ресурсів, списує їх із InventoryManager.Instance (патерн Singleton для доступу до глобального інвентаря), додає чи оновлює елементи черги та відлічує час виготовлення через InvokeRepeating

Останнім кроком у розробці гри стало створення конвеєру (рис. 3.9), який фізично штовхає ресурси за допомогою компонента Rigidbody. Через особливості гексагональної мапи конвеєр має закруглені сегменти, а рух по них описується квадратичною кривою Безьє.

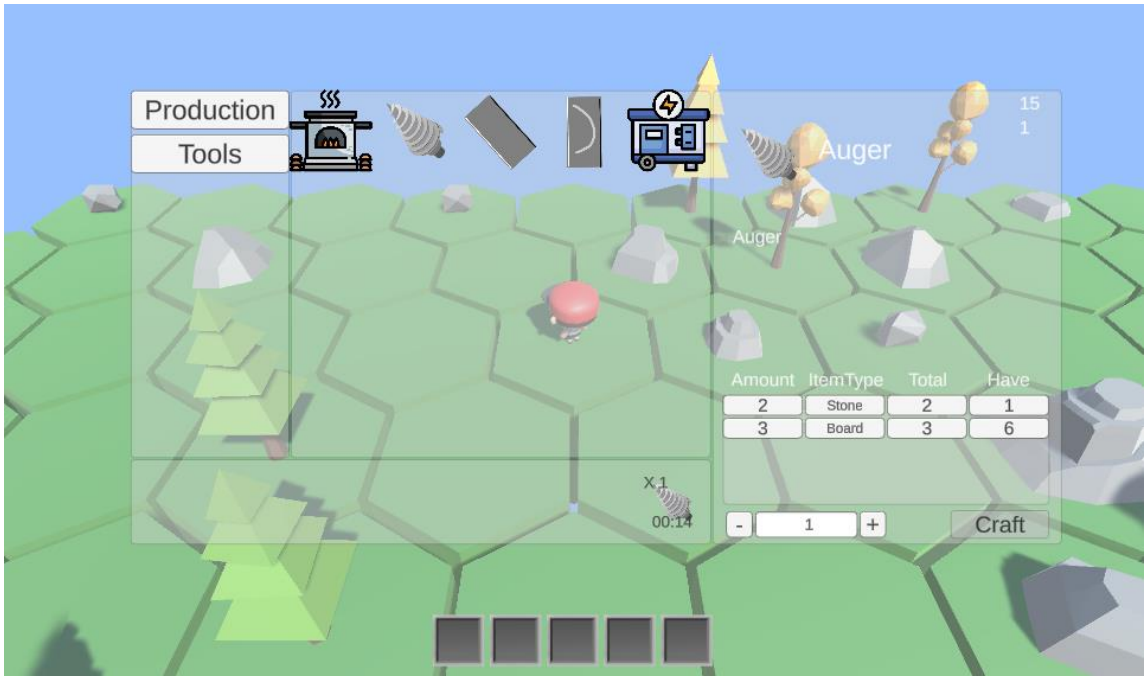

Рисунок 3.8 – Інтерфейс крафту

Джерело: розроблено автором

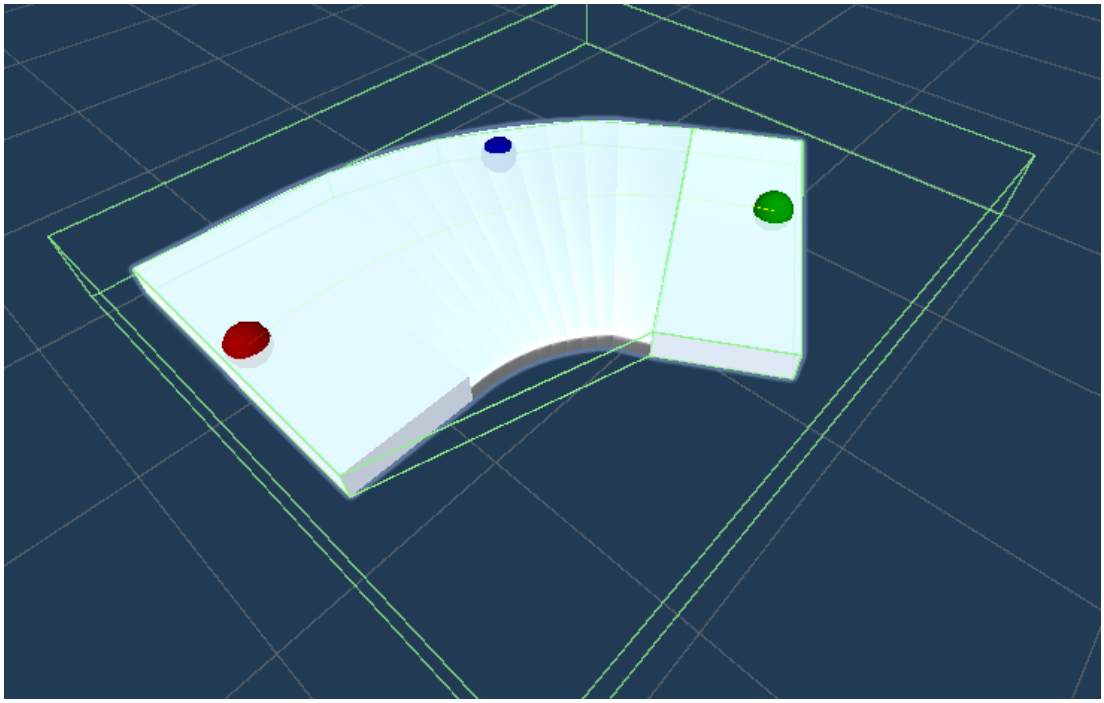

Рисунок 3.9 – Конвеєр

Джерело: розроблено автором

Для отримання напрямку руху об’єкта по кривій спочатку обчислюється параметр *t* на основі поточного положення ящика (метод GetTFromPosition). Далі за трьома контрольними точками (*P0* – startPoint, *P1* – curvePoint, *P2* – directionPoint) обчислюється касательний вектор – перша похідна кривої Безьє $B'(t)=2(1-t)(P1-P0)+2t(P2-P1), t\in[0,1]$[26]. В коді це зображено так (рис. 3.10)

```
if (startPoint != null && curvePoint != null && directionPoint != null)
{
    float t = GetTFromPosition(other.transform.position);
    // Обчислюємо дотичну криву Безьє:
    // B'(t) = 2(1-t)(curvePoint - startPoint) + 2t(directionPoint - curvePoint)
    tangent = (2 * (1 - t) * (curvePoint.position - startPoint.position) +
               2 * t * (directionPoint.position - curvePoint.position)).normalized;
}
else
{
    tangent = transform.forward;
}
```

Рисунок 3.10 – Приклад в коді кривої Безьє

Джерело: розроблено автором

3.2 Тестування програмного продукту

Тестування програмного продукту "Hex-factory" є важливим етапом розробки, що дозволяє перевірити відповідність гри заявленому функціоналу, виявити помилки та забезпечити стабільну роботу. Для цього було проведено функціональне та модульне тестування, які охоплюють ключові аспекти роботи гри.

Функціональне тестування [27] було проведено для перевірки відповідності заявленого функціоналу гри "Hex-factory". Метою було переконатися, що всі ігрові механіки працюють коректно. Тестування проводилося вручну в Unity Editor.

Переміщення гравця:

1. Перевірено, що гравець може вибирати гекс на карті та переміщатися до нього за допомогою алгоритму A*.
2. Перевірено коректність обходу непрохідних гексів (наприклад, зайнятих будівлями).
3. Перевірено відтворення анімації та звукових ефектів під час руху.

Збір ресурсів:

1. Перевірено, що гравець може розпочати та завершити збір ресурсу.
2. Перевірено правильність додавання зібраних предметів до інвентарю.
3. Перевірено анімацію зменшення масштабу ресурсу та звукові ефекти.

Система інвентарю:

1. Перевірено додавання та видалення предметів з інвентарю.
2. Перевірено відображення предметів у UI та коректність роботи швидких слотів.
3. Перевірено обмеження місткості інвентарю.

Система крафту:

1. Перевірено відображення доступних рецептів у UI.
2. Перевірено правильність перевірки наявності необхідних ресурсів.
3. Перевірено створення предметів після завершення крафту.

При альфа тестуванні було виявлено баг з пічкою, а саме некоректне відображення інтерфейсу печі, цей баг буде вирішено в майбутніх версіях гри.

Модульне тестування [28] проводилося для ізоляції та перевірки окремих компонентів коду гри, щоб переконатися в їх ефективності та коректності. Тестування виконувалося за допомогою Unity Test Framework, який дозволяє створювати автоматизовані тести для перевірки логіки окремих класів.

Клас AStarPathfinding:

Тест 1: Перевірка знаходження шляху між двома прохідними гексами.

1. Вхідні дані: Початковий гекс (0,0), цільовий гекс (2,2), сітка без перешкод.

2. Очікуваний результат: Список гексів, що формує найкоротший шлях.

3. Результат: Тест пройдено, шлях коректний.

Тест 2: Перевірка обробки непрохідних гексів.

1. Вхідні дані: Сітка з непрохідним гексом між початковим і цільовим.

2. Очікуваний результат: Обхідний шлях або null, якщо шлях неможливий.

3. Результат: Тест пройдено, алгоритм повертає обхідний шлях.

Клас Resource:

Тест 1: Перевірка методу StartGather.

1. Вхідні дані: Ресурс із заданим gatherTime.

2. Очікуваний результат: Запуск анімації та встановлення прапорця isBeingGathered.

3. Результат: Тест пройдено, анімація запускається коректно.

Тест 2: Перевірка методу CompleteGather.

1. Вхідні дані: Завершення збору ресурсу.

2. Очікуваний результат: Додавання предметів до інвентарю.

3. Результат: Тест пройдено, предмети додаються.

Клас InventoryManager:

Тест 1: Перевірка методу AddItem.

1. Вхідні дані: Предмет із заданою кількістю.

2. Очікуваний результат: Предмет додається до інвентарю, UI оновлюється.

3. Результат: Тест пройдено, інвентар оновлюється коректно.

Тест 2: Перевірка переповнення інвентарю.

1. Вхідні дані: Додавання предметів понад ліміт.

2. Очікуваний результат: Відмова у додаванні з повідомленням.
3. Результат: Тест пройдено, ліміт дотримується.

Усі тести для ключових модулів були успішно пройдені після виправлення незначних помилок, таких як неправильна обробка нульових значень у методі FindPath класу AStarPathfinding. Модульне тестування підтвердило надійність окремих компонентів і їх готовність до інтеграції в загальну систему.

3.3 Використання програмного продукту

Гра "Hex-factory" – це захоплююча пісочниця, розроблена на платформі Unity, що дозволяє гравцям досліджувати, добувати ресурси, будувати та автоматизувати виробничі процеси на гексагональній карті.

Інструкція користувача:

- клацніть лівою кнопкою миші по самому персонажу для того щоб його вибрати.
- виберіть гексагон на який бажаєте перемістити персонажа, якщо до того гексагона немає перешкод то персонаж піде на цей гексагон.
- підійдіть до ресурсу (наприклад, каменю) і натисніть ліву кнопку миші, щоб почати збір.
- утримуйте кнопку, поки збір не завершиться. Якщо відпустите раніше, процес буде скасовано.
- для того щоб розмістити об’єкт на гексагоні його потрібно перемістити в швидкий слот та вибрати його там натиснувши на клавішу від 1 до 5 та потім натиснути на вільний гексагон ПКМ (праву кнопкою миші).
- натисніть клавішу "E", щоб відкрити інвентар.
- для закриття UI інтерфейсу, потрібно буде натиснути на клавішу ESC.

- переглядайте та керуйте своїми ресурсами, перетягуючи предмети між слотами.
- натисніть клавішу "F", щоб відкрити панель крафту.
- виберіть рецепт і натисніть "Craft", якщо у вас є всі необхідні ресурси.
- дочекайтеся завершення крафту, щоб отримати новий предмет.
- розмістіть піч на карті та натисніть на неї ПКМ (праву кнопкою миші).
- додайте інгредієнти та запустіть процес плавки.
- після завершення процесу заберіть готові злитки з інвентарю печі.
- розмістіть бур на гексагоні, та зважавши цей бур на ПКМ (праву кнопкою миші) раз в 5 секунд він буде давати ресурси.
- підключіть генератор до буру за допомогою кабелю, а саме, натисніть ПКМ (праву кнопкою миші) на генератор, а потім на сам бур.
- розмістіть конвеєр для транспортування добутих ресурсів до печі або до гравця, для зміни напрямку конвеєра натисніть на нього ПКМ (праву кнопкою миші).
- якщо в генераторі мало топлива ви можете натиснути деревом яке потрібно добути на сам генератор.
- з дерева випадають сажанці які можна посадити та виростити дерево

Висновок до розділу 3

У результаті розробки гри "Hex-factory" було реалізовано цілісний процес створення продукту – від вибору технологій до підготовки до використання. Unity та C# забезпечили інтеграцію, гнучкість і високу якість розробки, а додаткові інструменти (DOTween, Blender, Meshy.ai) прискорили роботу.

Гексагональна сітка, рух через A*, інвентар на основі патерну Singleton, крафт через ScriptableObject і автоматизація виробництва конвеєрами з фізикою створили глибокий ігровий процес.

Функціональне та модульне тестування підтвердили стабільність гри. Інтуїтивний інтерфейс і детальна інструкція забезпечують зручність для гравців.

Проект "Hex-factory" досяг поставлених цілей і готовий до запуску з потенціалом розвитку.

ВИСНОВКИ

Проведено дослідження сучасних методів розробки комп’ютерних ігор. Проаналізовано аналоги комп’ютерних ігор що реалізують ігровий процес з використанням генерації ігрового світу та управління персонажу який добуває та працює об’єктами гри.

Спроектовано та розроблено програмне забезпечення комп’ютерної гри "Hex-Factory" з використання Unity, C#, DOTween, Blender і Meshy.ai, що дозволило створити продукт із плавними анімаціями та стабільною роботою. Тестування підтвердило коректність механік, хоча виявлені дрібні помилки, як-от з інтерфейсом печі, планується виправити.

Розробка гри "Hex-Factory" реалізувала створення динамічної пісочниці на гексагональній карті, оптимізованої для широкої аудиторії. Гра поєднує алгоритм A* для ефективного переміщення, системи інвентарю, крафту, переплавлення та автоматизації через бури, генератори й конвеєри, що забезпечують глибокий ігровий процес.

"Hex-Factory" демонструє застосування сучасних алгоритмів і патернів у геймдеві, є готовим до запуску продуктом із потенціалом для розвитку, зокрема додавання нових механік чи багатокористувацького режиму. Проєкт досяг мети, створивши захоплюючу гру, що відповідає стандартам ігрової індустрії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Number of PC gaming users worldwide from 2008 to 2025 URL: https://www.statista.com/statistics/420628/pc-gamers-by-category/ (дата звернення: 09.03.2025).

2. Інформація про гру Minecraft. Wikipedia URL: https://uk.wikipedia.org/wiki/Minecraft (дата звернення: 09.03.2025).

3. Інформація про гру Satisfactory. Wikipedia URL: https://uk.wikipedia.org/wiki/Satisfactory (дата звернення: 09.03.2025).

4. Гра Satisfactory на персональний комп'ютер. Steam URL: https://store.steampowered.com/app/526870/Satisfactory/ (дата звернення: 09.03.2025).

5. Інформація про гру Satisfactory. Wikipedia URL: https://uk.wikipedia.org/wiki/Civilization_VI (дата звернення: 09.03.2025).

6. Офіційна документація Microsoft. Мова програмування C# URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 09.03.2025).

7. Unity Engine. Офіційний сайт Unity. URL: https://unity.com/products/unity-engine (дата звернення: 09.03.2025

8. Introduction to the A* Algorithm URL: https://www.redblobgames.com/pathfinding/a-star/introduction.html (дата звернення: 09.03.2025).

9. Red Blob Games. Hexagonal Grids. Доступно за URL: https://www.redblobgames.com/grids/hexagons/ (дата звернення 09.03.2025).

10. Діаграма прецедентів. URL: https://uk.wikipedia.org/wiki/Діаграма_прецедентів (дата звернення: 22.03.2025).

11. Як будувати UML-діаграми. Доступно за URL: https://dou.ua/forums/topic/40575/ (дата звернення 22.03.2025).

12. Діаграми послідовності. Доступно за URL: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D

0%B0%D0%BC%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D1%96%D0%B4%D0%BE%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D1%96 (дата звернення 22.03.2025).

13. Діаграми класів. Доступна за URL: https://uk.wikipedia.org/wiki/Діаграма_класів (дата звернення: 22.03.2025).

14. Опитування Steam про обладнання та ПЗ. Доступна за URL: https://store.steampowered.com/hwsurvey/videocard/ (дата звернення: 30.03.2025).

15. Діаграма діяльності. Доступна за URL: https://uk.wikipedia.org/wiki/Діаграма_діяльності (дата звернення: 30.03.2025).

16. A зірка. Доступна за URL: https://uk.wikipedia.org/wiki/Алгоритм_пошуку_A* (дата звернення: 16.04.2025).

17. Бібліотека DOTWeen. Доступна за URL: https://dotween.demigiant.com/ (дата звернення: 16.04.2025).

18. Rider. Доступна за URL: https://www.jetbrains.com/rider/ (дата звернення: 16.04.2025).

19. Blender. Доступна за URL: https://store.steampowered.com/app/365670/Blender/ (дата звернення: 16.04.2025).

20. ШІ для створення 3Д моделів Meshy.ai. Доступна за URL: https://www.meshy.ai/ (дата звернення: 17.04.2025).

21. Unity Asset Store. Доступна за URL: https://assetstore.unity.com/ (дата звернення: 17.04.2025).

22. Вікіпедія A*. Доступна за URL: https://uk.wikipedia.org/wiki/Алгоритм_пошуку_A* (дата звернення: 18.04.2025).

23. Патер «Стан». Доступна за URL: https://refactoring.guru/uk/design-patterns/state (дата звернення: 19.04.2025).

24. 3D моделі які використовуються в грі. Доступна за URL: https://assetstore.unity.com/packages/3d/environments/lowpoly-nature-bundle-286938 (дата звернення: 19.04.2025).

25. Патер «Одина». Доступна за URL: https://refactoring.guru/uk/design-patterns/singleton (дата звернення: 19.04.2025).

26. Крива Безьє. Доступна за URL: https://uk.wikipedia.org/wiki/Крива_Безьє (дата звернення: 23.04.2025).

27. Функціональне тестування. Доступна за URL: https://uk.wikipedia.org/wiki/Функціональне_тестування (дата звернення: 24.04.2025).

28. Модульне тестування. Доступна за URL: https://uk.wikipedia.org/wiki/Модульне_тестування (дата звернення: 24.04.2025).

29. Мічківський С. Microsoft Office (Word, Excel, Outlook …) : навч. посіб. / С. Мічківський, Д. Балдик, В. Головань; Східноукр. нац. ун-т ім. В. Даля, Аграр. ф-т. – Київ : [Вид-во Східноукр. нац. ун-т ім. В. Даля], 2023. – 128 с. – URL: https://dspace.snu.edu.ua/handle/123456789/1723

30. Vysochyn I., Michkivskyy S. Using machine learning for translation and speech generation in e-book reading applications. Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку: матеріали VI Міжнародної наукової конференції (5-6 грудня 2024 р., м. Київ). Київ: Університет "КРОК", 2024. С.62-64 – URL: https://dspace.krok.edu.ua/items/6e1488e1-9e21-4282-af10-2e3bca48d2bc

31. Алимова О. В., Мічківський С. М. Розробка антагоністичної комп'ютерної гри «Лабіринт на двох». XII Міжнародній науково-технічній конференції «Проблеми Інформатизації» (м. Київ, 13 грудня 2018 року). Київ: Державний університет телекомунікацій, 2018.

32. Алимова О.В. РОЗРОБКА АНТАГОНІСТИЧНОЇ КОМП'ЮТЕРНОЇ ГРИ «ЛАБІРИНТ НА ДВОХ» / О.В. Алимова, С.М. Мічківський // Матеріали І Всеукраїнської науково-практичної інтернет-конференції студентів, аспірантів та молодих вчених за тематикою «Сучасні комп’ютерні системи та мережі в управлінні»: Збірка наукових праць[Текст] / Під редакцією Г.О. Райко (м.Херсон, 30 листопада 2018 року). – Херсон: ХНТУ, 2018 - 299 с. – С. 112-114

33. Фещенко М.І. Кіберспорт та освітній процес / М. І. Фещенко, Л.Б. Ігнатова // Держава, регіони, підприємництво: інформаційні, суспільно-правові, соціально-економічні аспекти розвитку: тези доповідей V Міжнародної конференції (Київ, 7 грудня 2023 р.). - Київ: Університет "КРОК", 2023. URL: https://conf.krok.edu.ua/SRE/SRE-2023/paper/view/1989

34. Системи та методи прийняття рішень: методичні вказівки / С. М. Мічківський, О. В. Прігунов, П. В. Римар. Вінниця, ДонНУ імені Василя Стуса, 2019, 76 с.

35. Троцько В.В. Методи штучного інтелекту: навчально-методичний і практичний посібник / В.В. Троцько. – Київ: Університет економіки та права «КРОК», 2020. – 86 с. URL: https://library.krok.edu.ua/media/library/category/navchalni-posibniki/trotsko_0001.pdf

ДОДАТОК А

```
private static Hex GetHexWithLowestFScore(List<Hex> hexes, Dictionary<Hex, float> fScore)
{
    Hex bestHex = hexes[0];
    float bestScore = fScore.ContainsKey(bestHex) ? fScore[bestHex] : float.MaxValue;

    foreach (Hex hex in hexes)
    {
        float score = fScore.ContainsKey(hex) ? fScore[hex] : float.MaxValue;
        if (score < bestScore)
        {
            bestHex = hex;
            bestScore = score;
        }
    }

    return bestHex;
}
```

Рисунок А.1 – Фрагмент класа «AStarPathfinding»

Джерело: розроблено автором

```
public class Hex : MonoBehaviour
{
    public Vector2Int coordinates;   Serializable
    public bool isWalkable;   "true"
    public GameObject occupyingObject;    Unchanged

    1 usage
    public override bool Equals([CanBeNull] object obj)
    {
        if (obj is Hex other)
            return coordinates.Equals(other.coordinates);
        return false;
    }

    public override int GetHashCode()
    {
        return coordinates.GetHashCode();
    }
}
```

Рисунок А.2 – Фрагмент класа «Hex»

Джерело: розроблено автором

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

1 asset usage  8 usages
public class HexGrid : MonoBehaviour
{
    public static event Action OnGridGenerated;
    public GameObject hexPrefab;  Changed in 1 asset
    public int gridWidth = 10;  Unchanged
    public int gridHeith = 10;  Unchanged
    public float hexSize = 10f;  "2"

    private static Dictionary<Vector2Int, Hex> hexMap = new Dictionary<Vector2Int, Hex>();
    Event function
    private void Start()
    {
        GenerateGrid();
        OnGridGenerated?.Invoke();
    }

    1 usage
    void GenerateGrid()
    {
        for (int x = 0; x < gridWidth; x++)
        {
            for (int y = 0; y < gridHeith; y++)
            {
                float xOffset = (y % 2 == 0) ? 0 : hexSize * 0.75f;
                Vector3 position = new Vector3(x: x * hexSize * 1.5f + xOffset, y: 0, z: y * hexSize * 3 / 2.25f);
                GameObject hex = Instantiate(hexPrefab, position, rotation: Quaternion.Euler(x: 0, y: 90, z: 0), transform);
                Hex hexComponent = hex.GetComponent<Hex>();
                if (hexComponent != null)
                {
                    hexComponent.coordinates = new Vector2Int(x, y);
                    hexMap.Add(hexComponent.coordinates, hexComponent);
                }

            }
        }
    }
```

Рисунок А.3 – Фрагмент класа «HexGrid»

Джерело: розроблено автором

```
using UnityEngine;

4 usages  2 inheritors
public interface IInputState
{
    Frequently called  1 usage  2 implementations
    void Enter();
    Frequently called  1 usage  2 implementations
    void Exit();
    Frequently called  1 usage  2 implementations
    void Update();

    Frequently called  1 usage  2 implementations
    void OnLeftMouseDown(RaycastHit hit);
    Frequently called  1 usage  2 implementations
    void OnLeftMouseHold();
    Frequently called  1 usage  2 implementations
    void OnLeftMouseUp();

    Frequently called  1 usage  2 implementations
    void OnRightMouseDown(RaycastHit hit);
    Frequently called  1 usage  2 implementations
    void OnRightMouseUp();
}
```

Рисунок А.4 – Фрагмент інтерфейса «IInputState»

Джерело: розроблено автором

```
public void AddItem(InventoryScriptableObject newItem, int newAmount)
{
    foreach (InventorySlot slot in slots)
    {
        if (slot.item == newItem)
        {
            if (slot.amount + newAmount <= newItem.maxAmount)
            {
                slot.amount += newAmount;
                slot.itemAmountText.text = slot.amount.ToString();
                return;
            }
        }
    }

    foreach (InventorySlot slot in slots)
    {
        if (slot.isEmpty)
        {
            slot.item = newItem;
            slot.amount = newAmount;
            slot.isEmpty = false;
            slot.SetIcon(newItem.icon);
            slot.itemAmountText.text = newAmount.ToString();
            return;
        }
    }
}
```

Рисунок А.5 – Фрагмент класа «InventoryManager»

Джерело: розроблено автором

```
public void LoadCraftItem(string craftType)
{
    for (int i = 0; i < craftItemPanel.childCount; i++)
    {
        Destroy(craftItemPanel.GetChild(i).gameObject);
    }
    foreach (CraftSriptableObject cso in allCraft)
    {
        if (cso.craftType.ToString().ToLower() == craftType.ToLower())
        {
            GameObject craftItemButton = Instantiate(craftItemButtonPrefab, craftItemPanel);
            craftItemButton.GetComponent<Image>().sprite = cso.finalCraft.icon;
            craftItemButton.GetComponent<FillCraftItemDetails>().currentCraftItem = cso;
        }
    }
}
```

Рисунок А.6 – Фрагмент класа «CraftingManager»

Джерело: розроблено автором

```
public GameObject playerPrefab;  Player
public Vector2Int startCoordinates;  Serializable
public float heightOffset = 0.5f;  "0"
public ObjectSpawner objectSpawner;  Object (ObjectSpawner)

Event function
private void OnEnable()
{
    HexGrid.OnGridGenerated += SpawnPlayer;
}

Event function
private void OnDisable()
{
    HexGrid.OnGridGenerated -= SpawnPlayer;
}

2 usages
private void SpawnPlayer()
{
    Vector2Int spawnCoordinates = FindFreeCoordinates(startCoordinates);

    Hex targetHex = HexGrid.GetHexAt(spawnCoordinates);
    if (targetHex != null)
    {
        Vector3 spawnPosition = targetHex.transform.position;
        spawnPosition.y += heightOffset;

        GameObject spawnedPlayer = Instantiate(playerPrefab, spawnPosition, Quaternion.identity);
        spawnedPlayer.GetComponent<PlayerMovement>()?.MoveToHex(targetHex);

        objectSpawner.MarkCoordinateAsOccupied(spawnCoordinates);
    }
}
```

Рисунок А.7 – Фрагмент класа «PlayerSpawner»

Джерело: розроблено автором