МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій

«До захисту допущено»

В.о. завідувача кафедри

___________ Світлана ВАЩЕНКО

____ ___________ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»____________________________________,

освітньо-професійної програми «Інформаційні технології проектування»__________

на тему: Інформаційна технологія процедурної генерації контенту у десктопних іграх
____

Здобувача (ки) групи ІТ.м-23 ______Кіптенко Богдана Анатолійовича__________
(шифр групи) (прізвище, ім'я, по батькові)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

____________ ____Богдан КІПТЕНКО_______
(підпис) (Ім'я та ПРІЗВИЩЕ здобувача)

Керівник _____к.т.н.доцент, Наталья ФЕДОТОВА_____________________________
____
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри ІТ

Світлана ВАЩЕНКО
«_____» ________ 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

_______Кіптенко Богдан Анатолійович________
(прізвище, ім'я, по батькові)

1 Тема кваліфікаційної роботи *Інформаційна технологія процедурної генерації контенту у десктопних іграх*___________________________________
затверджена наказом по університету від «08» листопада 2023 р. № 1249-VI

2 Термін здачі студентом кваліфікаційної роботи « __ » грудня 2023 р.

3 Вхідні дані до кваліфікаційної роботи необхідність в методиках та інструментах дизайну контенту настільних ігор, перелік вимог до розробки контенту настільних ігор

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, моделювання інформаційної системи, практична реалізація проекту

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових слайдів презентації) актуальність, функціональні вимоги, структурно-функціональне моделювання, моделювання варіантів використання, параметри варіативності, демонстрація роботи програмних модулів________________________

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання ____________________________.

Керівник ____________________________
(підпис)

Завдання прийняв до виконання ____________________________
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області	24.09.2023	
2	Визначення мети та завдань	24.09.2023	
3	Вибір інструментів реалізації	25.09.2023	
4	Створення ігрового прототипу	02.10.2023	
5	Реалізація програмних модулів	24.10.2023	
6	Дослідження ігрових параметрів	02.11.2023	
7	Доопрацювання ігрового прототипу	16.11.2023	
8	Розробка технологічного забезпечення гри	29.11.2023	
9	Оформлення звіту практики	04.12.2023	
10	Створення матеріалів презентації	05.12.2023	

Магістрант Богдан КІПТЕНКО

Керівник роботи к.т.н., доц. Наталія ФЕДОТОВА

АНОТАЦІЯ

Тема кваліфікаційної роботи магістра «Інформаційна технологія процедурної генерації контенту у десктопних іграх».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 42 найменувань, додатків. Загальний обсяг роботи − 103 сторінки, у тому числі 50 сторінок основного тексту, 4 сторінки списку використаних джерел, 41 сторінка додатків.

Актуальність роботи полягає в недостатній дослідженості прийомів генерації ігрового контенту та їх впливу на ігровий процес в сфері настільних ігор. Дослідження впливу даної технології на ігровий процес надасть можливість створити інструменти та методики для швидкого та більш дієвого дизайну ігрових механік та технологічного забезпечення настільних ігор, що в свою чергу підвищить унікальність та індивідуалізованість ігрового досвіду.

Мета роботи: дослідження впливу процедурної генерації контенту (PCG) на дизайн ігрових механік, визначення параметрів що найбільше впливають на варіативність ігрового процесу, визначення оптимальних алгоритмів для реалізації окремих ігрових механік, розробка технологічного забезпечення прототипу настільної гри на основі отриманих результатів

Ключові слова: процедурна генерація контенту (PCG), алгоритм процедурної генерації, Wave Function Collapse, ігрова механіка, ігрове поле, тайл, технологічне забезпечення, Python.

ЗМІСТ

ВСТУП... 7
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 9
1.1 Процедурна генерація контенту 9
1.2 Алгоритми процедурної генерації.. 11
1.3 Генерація контенту в ігровому дизайні 16
2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ 21
2.1 Мета дослідження 21
2.2 Вимоги до модулів інформаційної технології.. 22
2.3 Інструменти дослідження.. 23
2.4 Вимоги до сценарію створення контенту 24
3. МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ............................... 27
3.1 Структурно-функціональне моделювання процесу 27
3.2 Моделювання варіантів використання.. 28
3.3 Проектування програмної реалізації алгоритму WFC 29
3.4 Проектування програмної реалізації псевдовипадкового алгоритму........ 30
3.5 Проектування програмної реалізації порівняння зображень 31
4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ... 33
4.1 Визначення ігрових параметрів для аналізу... 33
4.2 Аналіз варіативності побудови ігрового поля.. 37
4.3 Реалізація програмних модулів 45
4.4 Завершення ігрового прототипу 55
ВИСНОВКИ.. 57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 58

ДОДАТОК А..62

ДОДАТОК Б ПРОГРАМНИЙ МОДУЛЬ АЛГОРИТМУ WAVE FUNCTION COLLAPSE..68

ДОДАТОК В ПРОГРАМНИЙ МОДУЛЬ АЛГОРИТМУ WAVE FUNCTION COLLAPSE..78

ДОДАТОК Г ПРОГРАМНИЙ МОДУЛЬ ПСЕВДОВИПАДКОВОГО АЛГОРИТМУ 79

ДОДАТОК Д ПРОГРАМНИЙ МОДУЛЬ АНАЛІЗУ ПОДІБНОСТІ ЗОБРАЖЕНЬ 82

ДОДАТОК Е ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОТОТИПУ ГРИ.........85

ДОДАТОК Є ІГРОВІ ПРАВИЛА...94

ВСТУП

Актуальність. В сучасному інформаційному суспільстві, яке визначається стрімким технологічним розвитком, ігрова індустрія стає динамічним полем для впровадження новаторських підходів та концепцій. Хоч найбільшу кількість уваги бере на себе відносно нова індустрія комп’ютерних ігор, не менш цікавою є сфера настільних ігор. І не дивно, явище «гри» має глибоку історію, що своїм корінням іде в далеке минуле. Історія ігор бере початок ще з давніх часів. Однією з перших вважається «Royal Game of Ur» в яку щонайменше 4500 років тому грав один з древніх народів Близького Сходу – шумери [10]. З плином часу явище «гри» поступово переросло в цілу культуру. На сьогодні люди й досі проявляють інтерес до ігор, який лише збільшується з розвитком технологій [2].

На думку Рафа Костера люди проявляють інтерес до ігор через те, що вони надають їм новий досвід та виклик у формі задач та головоломок, вирішення яких і дає змогу отримати цей досвід. Таким чином ігри перестають бути цікавими коли ми перестаємо отримувати від них щось нове [18]. Дане висловлювання, водночас, підтверджує і те, що ігри у своїй основі мають навчальний характер, будь то вивчення ігрових правил, отримання знань через ігрові елементи чи покращення певних навичок в процесі гри. Можна зробити висновок що настільні ігри, в додаток до розважального характеру, є дієвим інструментом у навчальному процесі маючи здатність розвивати специфічні аспекти особистості, як критичне мислення, стратегічне планування та соціальні навички [1, 3, 38].

Технологія процедурної генерації зародилася як інструмент для дослідження складних фізичних явищ, які неможливо було в повній мірі вивчити існуючими на той час методами. Сьогодні вона широко використовується у різних сферах медіа простору, графічному дизайні та ігровому дизайні. Інформаційна технологія процедурної генерації контенту є

дієвим інструментом для створення унікального ігрового досвіду як в сфері комп’ютерних так і настільних ігор.

В роботах пов’язаних з тематикою ігрового дизайну [11, 14, 15] мало досліджені прийоми генерації ігрового контенту та їх вплив на ігровий процес. Натомість більший фокус прослідковується у зв’язку ігрової механіки з поведінкою та психологією гравців. Дослідження впливу даної технології на ігровий процес дозволить створити інструменти та методики для швидкого та більш дієвого дизайну ігрових механік та технологічного забезпечення настільних ігор. Це надає більше можливостей для створення непередбачуваного та індивідуалізованого досвіду, унікальних приємних вражень гравців від ігрового процесу.

Об’єкт дослідження – інформаційна технологія процедурної генерації контенту.

Предмет дослідження – параметр варіативності генеруємого контенту алгоритмами процедурної генерації.

Мета – дослідження впливу процедурної генерації на варіативність генеруємого контенту, створення оптимального технологічного забезпечення настільної гри з процедурною генерацією.

Практична цінність – створення переліку параметрів для оптимального дизайну/вибору алгоритмів процедурної генерації для створення контенту та ігрових механік.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Процедурна генерація контенту

Процедурна генерація контенту (PCG) – це автоматизоване створення контенту за допомогою формально-визначених алгоритмів. Вона використовується для прискорення та спрощення процесу створення контенту, який інакше б творився вручну людиною. Процедурна генерація зазвичай означає застосування програмних засобів сучасної обчислювальної техніки – комп'ютерних програм та додатків.

PCG – широко розповсюджений засіб як у комп'ютерній графіці (створення 3D моделей та текстур), так і у медіа просторі загалом (генерація тексту, зображень). Сьогодні даний інструмент більше асоціюється з дизайном та розробкою ігрових додатків, оскільки вона включає в себе елементи обох попередньо згаданих сфер. Методами процедурної генерації можливо створювати різноманітні ігрові елементи: елементи ігрового середовища (текстури, 3-D моделі, візуальні ефекти, структури, рівні, місцевість), ігрові ситуації, діалоги. Історично, дані методи рідко застосовувались для генерації цілих ігрових рівнів, проте помітним винятком є підземелля – специфічний тип ігрових рівнів, що найчастіше зустрічається у пригодницьких та рольових іграх. Прикладами таких ігор є «Rogue: Exploring Dungeons of Doom» (1980) (див. рис. 1.1) та «Beneath Apple Manor» (1978) (див. рис. 1.2). Завдяки особливостям ігрового процесу та ігрового простору, рівні підземелля є одними з найбільш придатних для демонстрації переваг алгоритмів процедурної генерації. Методи генерації такого контенту варіюються від систем відносно заснованих на наборі правил до систем що працюють на кропітких процесах пошуку [32].

Рисунок 1.1 — Знімок екрана «Rogue: Exploring Dungeons of Doom»

Джерело: [12]

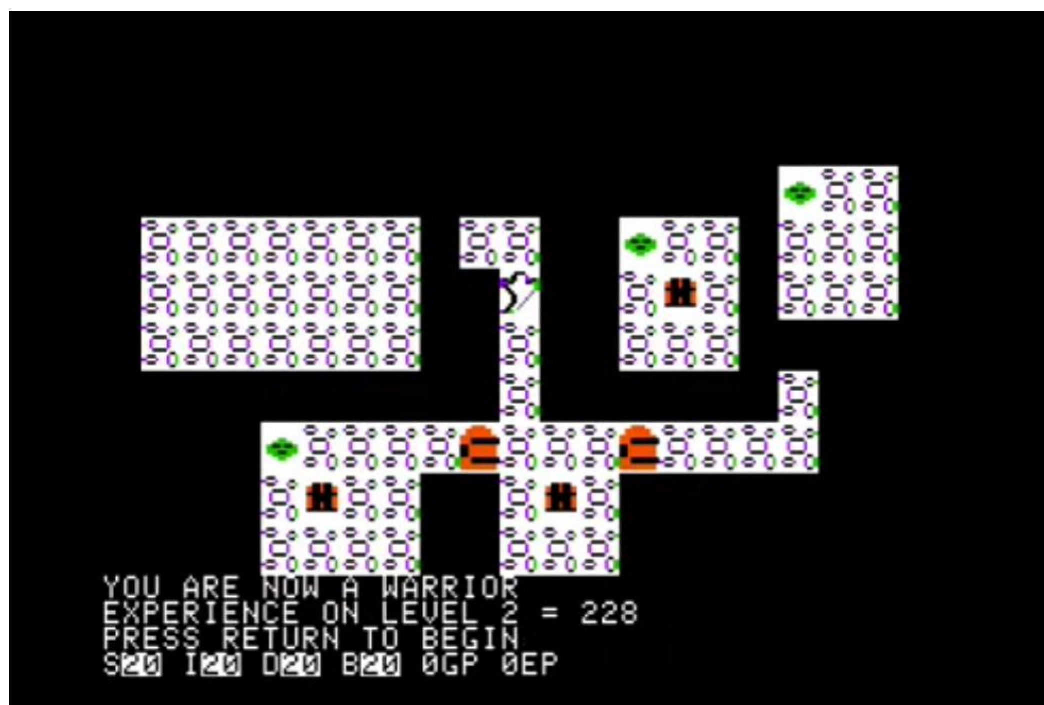


Рисунок 1.2 — Знімок екрана «Beneath Apple Manor»

Джерело: [41]

1.2 Алгоритми процедурної генерації

На сьогодні існує велика кількість різноманітних алгоритмів генерації контенту. Розглянемо кілька з них.

1.2.1 Модель «Cellular automaton».

Клітинний автомат (англ. Cellular automaton) – це модель системи об'єктів «клітин», що живуть на сітці. Клітини перебувають у певному стані, кількість яких є обмеженою. У найпростішій формі системи, клітини можуть перебувати в одному з двох станів, «жива» або «мертва» що в бінарній системі відповідають 1 або 0. Кожна клітина має визначену околицю, що зазвичай є списком сусідніх комірок.

Розробка клітинного автомату зазвичай приписується математикам Station Станіславу Уламу (англ. Stanislaw Ulam) та Джону фон Нейману (англ. John von Neumann), які в 1940-х роках працювали над ним в Лос-Аламоській національній лабораторії в Нью-Мексико. Призначенням даної системи було дослідження фізичних та біологічних процесів, по типу рідин, руху часток у просторі, поведінки клітин, тощо [8, 17]. Створення складних геометричних форм, як результат роботи системи, дозволяє застосовувати дану модель у процедурній генерації. Проста модель з 2-ма станами клітин здатна створювати зображення, що можуть бути використані в ігровому дизайні для генерації загальної форми рівнів чи ландшафту (див. рис. 1.3).

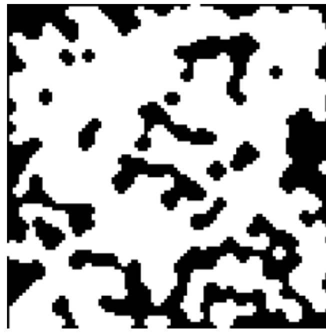


Рисунок 1.3 — Результат роботи клітинного автомату (2 стани клітин)

Джерело: [7]

Збільшення кількості станів, або ж «типів» клітин дає змогу генерувати більш складні зображення, що можуть використовуватись для більш детальної побудови мапи ігрового середовища (див. рис. 1.4) [5].

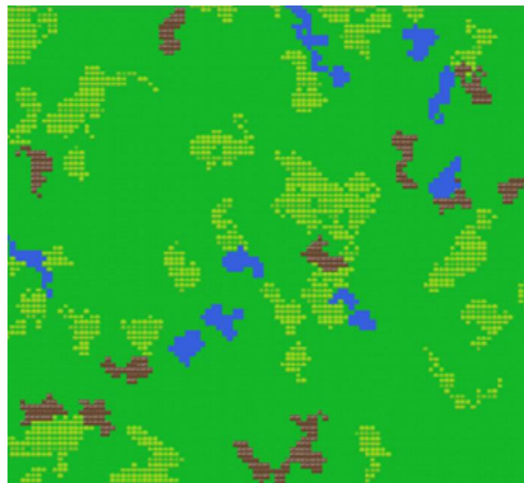


Рисунок 1.4 — Результат роботи клітинного автомату (4 стани клітин)

Джерело: [5]

1.2.2 Алгоритм «Model synthesis».

Model synthesis розроблений комп’ютерним вченим та інженером Полом Мерелом (англ. Paul Merrell) у 2007 році. Алгоритм здатен створювати великі складні дво- та тривимірні форми. Він розроблювався на основі алгоритму синтезу текстур (texture synthesis) [42], який працює за схожим принципом, але здатен генерувати лише двовимірні моделі [21, 23].

Метою алгоритму синтезу моделі (model synthesis) є створення моделі M, яка узгоджується з моделлю E. Тобто кінцевим результатом алгоритму є складніша модель, яка схожа на вхідну, яку алгоритм використовує як основу. Алгоритм приймає на вхід просту 2D або 3D-форму, а потім генерує більшу і складнішу модель, яка нагадує вхідну за формою та локальними особливостями (див. рис. 1.5-1.6).

Генерація відбувається шляхом окремого присвоєння мітки кожній точці сітки. Присвоєння є парою (x, b) точки x та мітки b. Згенерована модель M змінюється з часом по мірі додавання цих міток. Заповнення іде до тих пір доки є точки з пустою міткою. Оскільки алгоритм працює із сіткою його можливості обмежені параметрами цієї сітки [22].

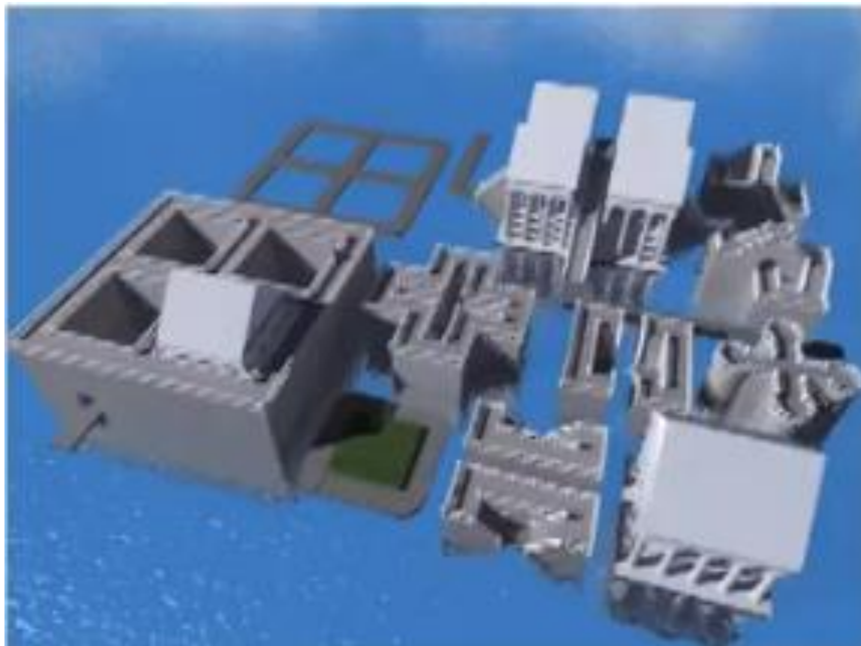


Рисунок 1.5 — Вхідна модель алгоритму Model Synthesis

Джерело: [28]

Рисунок 1.6 — Результат роботи алгоритму Model Synthesis

Джерело: [28]

1.2.3 Алгоритм «Wave Function Collapse».

Wave Function Collapse (WFC) – алгоритм процедурної генерації розроблений у 2016 році Максимом Гаміним (англ. Maxim Gumin) на основі робіт Пола Мерела. Даний метод також створює складні 2D або 3D моделі на основі простих вхідних форм (див. рис. 1.7-1.8).

Алгоритм приймає на вхід набір базових елементів. Кожен з них має набір правил щодо того які елементи можуть стояти поряд. Елементи приймають квадратну чи кубічну форму та мають 4 або 6 граней для дво- та тривимірного простору відповідно. Кожна грань має свою мітку. Якщо грані елементів мають однакові мітки, вони можуть з’єднуватися на цій грані. Таким чином алгоритм проходить по кожному сектору сітки визначаючи ентропію (англ. entropy) – міру невизначеності інформаційної системи. В даному випадку це кількість можливих варіантів в межах одного сектора сітки [13]. При заповненні елемента сітки набір можливих значень на сусідніх з ним секторах звужується. Сектори з найменшою кількістю можливих значень заповнюються першими. Процес

продовжується до тих пір доки не буде заповнена вся сітка. Таким чином принцип роботи WFC має схожість з грою «Судоку» в якій потрібно заповнити поля сітки числами від 0 до 9 без повторів по лініям, стовбцям та секторам 3x3 на які ця сітка поділена [6].

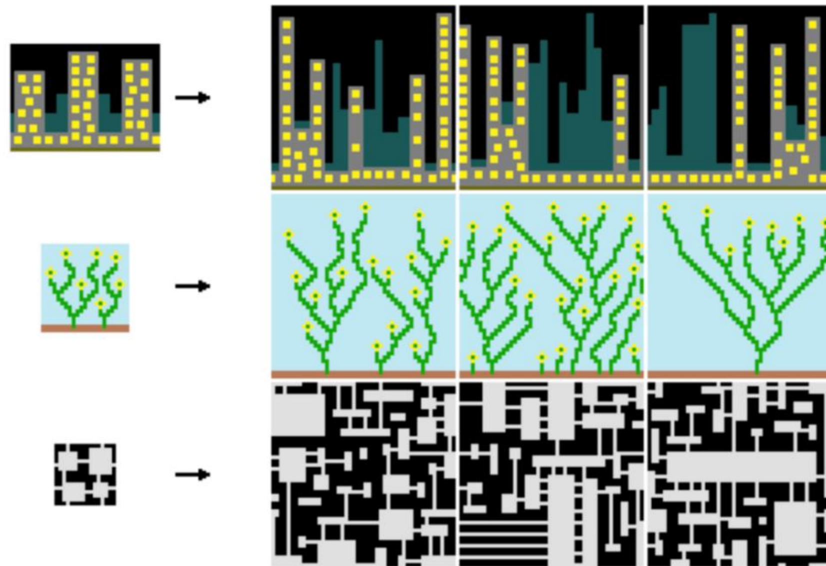


Рисунок 1.7 — Результат роботи алгоритму Wave Function Collapse (двовимірна модель)

Джерело: [16]

Рисунок 1.8 — Результат роботи алгоритму Wave Function Collapse (тривимірна модель)

Джерело: [19]

1.3 Генерація контенту в ігровому дизайні

Процедурна генерація контенту на сьогоднішній день є розповсюдженим та, можливо, одним з головних інструментів ігрового дизайну. Більш широкого застосування вона набула в індустрії відеоігор, проте процедурна генерація в певній формі простежується і в сфері настільних ігор.

Настільні ігри бувають різних жанрів та видів залежно від технологічного забезпечення, що може складатися з карток, гральної дошки, ігрового поля, тайлів (сегментів поля), жетонів, фігурок та гральних костей (дайсів, від англ. «dice») з різною кількістю граней. В останні роки їхня складність значно зросла, збільшилася варіативність ігрових компонентів, правил, які динамічно змінюються під час гри, різноманітних ролей гравців та низки параметрів, які впливають як на ігровий процес так і на баланс [14].

Одним з мотивів для використання PCG в створенні ігор є покращення їх реіграбельності, тобто здатності гри не втрачати варіативність, надавати гравцеві унікальний досвід при кожній новій ігровій сесії. Найоптимальніше застосування алгоритмів процедурної генерації, зважаючи на їх особливості, можна знайти в іграх в яких послідовно будується ігрове поле у формі мапи місцевості. Технологічне забезпечення таких ігор, найчастіше, може включати тайли або картки. Розглянемо кілька прикладів подібних ігор, які включають елементи процедурної генерації контенту.

1.3.1 Настільні рольові ігри

Настільні рольові ігри (TTRPG) для багатьох є популярним хобі через унікальний досвід який вони пропонують гравцям, а саме спільне оповідання історії. Оповідання здебільшого ведеться від імені одного з гравців, який бере на себе роль "Майстра гри". Він виступає в ролі судді, що визначає правила ігрового світу, результати подій та дій інших гравців, кожен з яких грає роль одного персонажа. Складний та специфічний дизайн не дозволяє в повній мірі

відтворити досвід настільних рольових ігор в цифровому форматі, водночас, вони надають унікальний досвід та мають дуже високий рівень варіативності. В даному типі ігор за дизайном необхідна участь людського фактору, тож до них складно застосувати термін саме «процедурної» генерації. Скоріше вони надають гравцям набір правил та інструментів що дозволяють їм власноруч створювати контент, в процесі адаптуючи дані правила згідно власних інтересів [30]. Натомість засоби PCG можна застосувати для окремих аспектів рольової гри.

Однією зі складових більшості TTRPG є дослідження ігрової місцевості та покрокові битви. Для кращої візуалізації та більш інтерактивного досвіду гри, гравці можуть використовувати мапи місцевості, які можуть різнитися від простого ручного рисунку на папері, зібраної модульної сцени, як показано на рисунку 1.9, або згенерованого в програмному додатку середовища. Для останнього способу існує безліч онлайн інструментів, що використовують алгоритми процедурної генерації (див. рис. 1.10) [32]. Випадкові та псевдовипадкові алгоритми також дають спроможність генерувати персонажів, ігрові події, завдання та нагороди.

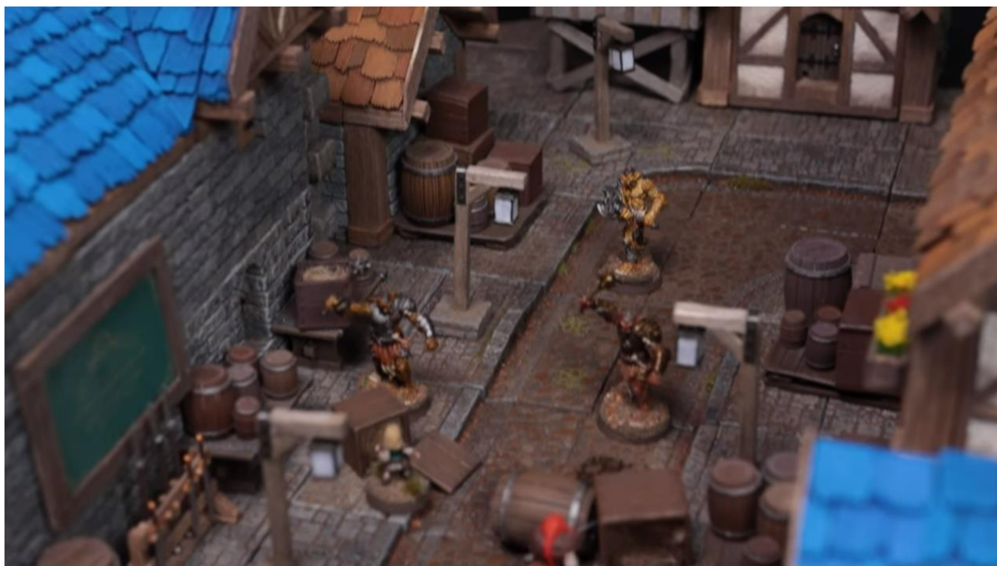


Рисунок 1.9 — Модульна саморобна сцена

Джерело: [33]

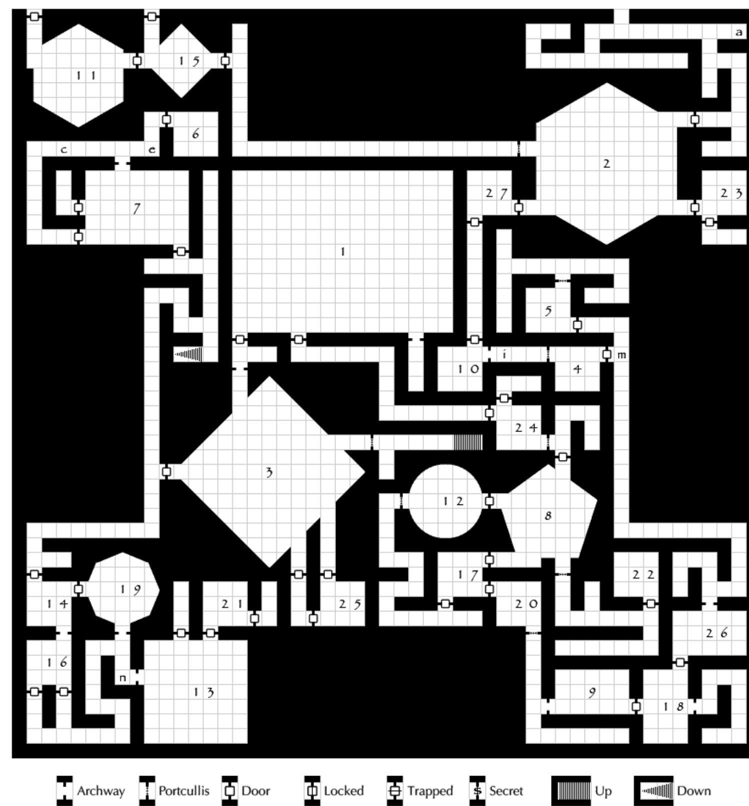


Рисунок 1.10 — Результат роботи онлайн інструменту «donjon»

Джерело: [32]

1.3.2 Гра «Carcassone»

Каркассон (нім. Carcassone) – німецька настільна гра розроблена Клаусом Юргеном Вреде (нім. Klaus-Jürgen Wrede) та вперше видана компанією «Hans im Glück» в Германії у 2000 році.

Суть гри у послідовному будівництві ігрового поля з тайлів (від англ. «tile») та розміщенні фігурок «підданих». На початку гри розміщується стартовий тайл, від якого гравці розбудовують мапу. Нові тайли обов'язково повинні розміщуватись біля граней вже поставлених елементів ігрового поля, як показано на рисунку 1.8.

Рисунок 1.8 — Приклад розміщення тайлів у грі «Carcassone»

Джерело: [40]

При цьому елемент малюнку на самому тайлі теж повинен відповідати його сусіду: дорога має з’єднуватись з дорогою, місто з містом, поле з полем [40]. Такі правила розміщення є прикладом використання алгоритму Wave Function Collapse.

1.3.3 Гра «Saboteur»

Саботер (англ. Saboteur) – настільна гра розроблена бельгійським дизайнером Фредеріком Моерсоеном (англ. Frederic Moyersoen) та випущена у 2005 році німецькою компанією «AMIGO».

Гравці діляться на 2 команди. Суть гри у побудові ігрового поля з карток з зображеннями тонелів та знаходженні картки з «золотом». На початку гри на поле виставляється стартова картка, від якої гравці розбудовують «тонелі» та 3 картки перевернуті лицьовою стороною вниз, розташовані на відстані 7 карток від старту. На двох з них знаходиться «вугілля», остання ж є «золотом» яке одна з команд – «шахтери», повинна знайти, а інша – «саботери», намагаються їм завадити. Поле будується за тим же принципом що і в згаданому раніше Каркассоні – нові карти повинні розміщуватись вздовж граней інших на полі, малюнок на карті має відповідати тому що на сусідній, тобто тунелі мають з’єднуватися. Грані карт теж повинні співпадати: довга біля довгої, коротка біля короткої (див. рис. 1.9) [24].

Дані принципи розміщення карток є ще одним прикладом алгоритму Wave Function Collapse.

Рисунок 1.9 — Приклад розміщення тайлів у грі «Saboteur»

Джерело: [34]

2. ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета дослідження

Спираючись на результати аналізу предметної області можна зробити висновок, що застосування процедурної генерації контенту має своє місце в ігровому дизайні, проте є мало дослідженим у галузі настільних ігор.

Метою роботи є дослідження впливу процедурної генерації на варіативність генеруємого контенту, створення оптимального технологічного забезпечення настільної гри з процедурною генерацією. Дослідження впливу процедурної генерації контенту (PCG) на дизайн ігрових механік, визначення параметрів що найбільше впливають на варіативність ігрового процесу, визначення оптимальних алгоритмів для реалізації окремих ігрових механік, розробка технологічного забезпечення прототипу настільної гри на основі отриманих результатів.

Для досягнення визначеної мети у дослідженні впливу PCG на дизайн ігрових механік та розробці технологічного забезпечення прототипу настільної гри, необхідно вирішити наступні задачі:

Зробити літературний огляд та його аналіз:

- Провести огляд літератури для збору існуючих знань щодо впливу PCG на дизайн ігрових механік та визначення параметрів.
- Проаналізувати існуючі алгоритми процедурної генерації та їхні застосування в ігровій розробці.
- Ознайомитися зі сучасними тенденціями та досягненнями в галузі процедурної генерації контенту в ігровій індустрії.
- Ознайомитися з існуючими прикладами ігор що використовують PCG для генерації контенту.

Провести розробка ігрового прототипу:

- Розробити прототип гри з інтеграцію PCG у її ключові елементи, зокрема побудову ігрового поля.

Реалізація та імплементація оптимальних алгоритмів:

- Створити програмну імплементацію обраних алгоритмів для симуляції ігрової механіки.
- Порівняти результати експериментів та визначити оптимальні алгоритми для реалізації конкретних ігрових механік.

Визначити ключові параметри ігрового процесу:

- Визначити параметри гри, які впливають на варіативність ігрового процесу та дизайн механік.
- Встановити критерії оцінки впливу PCG на ці параметри.

Розробити технологічне забезпечення прототипу настільної гри:

- Використовуючи результати дослідження розробити технологічну базу для створення прототипу настільної гри.
- Розробити просте у відтворенні технологічне забезпечення (ігрові правила, тайли, тощо) для формату настільної гри.

2.2 Вимоги до модулів інформаційної технології

З метою аналізу кількісних показників необхідно створити програмний модулі для симуляції обраних алгоритмів процедурної генерації.

Модулі повинні:

- приймати вхідні дані у вигляді тестових зображень;
- реалізовувати обраний алгоритм процедурної генерації;
- графічно відображати результати роботи;
- зберігати результати у форматі файлів зображення .png;

Додатково створити програмний модуль для графічного порівняння результуючих зображень та дослідження кількісних параметрів їх подібності.

Модуль повинен:

- приймати вхідні дані у вигляді файлів порівнюваних зображень;
- обчислювати параметри подібності зображень;
- графічно відображати результати обчислень;

2.3 Інструменти дослідження

Для реалізації програмних модулів обрано мову програмування Python. Python – об'єктно-орієнтована мова програмування загального призначення, розроблена нідерландським програмістом Гвідо ван Россумом (англ. Guido van Rossum). Дана мова має простий синтаксис, обширну матеріальну базу та документацію, що робить її нескладною у вивченні. Будучи інтерпретованою мовою, Python є дещо повільнішою в порівнянні з мовами C та C++, проте натомість має велику кількість бібліотек, що реалізують методи математичного аналізу, машинного навчання та графічної візуалізації отриманих результатів [31].

Для виконання поставлених задач можуть бути використані наступні бібліотеки:

- Pillow – бібліотека методів роботи з файлами зображень [29];
- Scikit-image – бібліотека алгоритмів обробки зображень [35];
- Numpy – бібліотека методів математичних операцій [25];
- OpenCV – бібліотека операцій з масивами даних [26];
- Matplotlib – бібліотека методів графічного відображення даних [20].

2.4 Вимоги до сценарію створення контенту

Для початку дослідження необхідно створити прототип гри, в основі механіки якої буде закладено алгоритм процедурної генерації. Побудова ігрового поля є оптимальним варіантом для використання PCG, спираючись на існуючі приклади її імплементації у сфері ігрового дизайну.

Ефективний та простий варіант реалізації технологічного забезпечення – квадратні тайли, з яких буде будуватися ігрове поле, утворюючи квадратну двовимірну сітку по якій будуть пересуватися гравці. Кожен одиничний сектор сітки може визначати тип «місцевості», яка буде впливати на можливість гравця до просування. Нехай в кожного гравця є ресурс що витрачається на просування ігровим полем. Таким чином кількість ходів кожного гравця є обмеженою. Тоді наявність секторів що змусять витратити більше ресурсу, відповідно зменшивши пройдену відстань за хід, змусять гравців знаходити оптимальні шляхи для просування. Для прототипу створимо 3 типи місцевості:

- «рівнини» – сектори білого кольору, втрата 1 одиниці ресурсу
- «ліс» – сектори зеленого кольору, втрата 2 одиниць ресурсу
- «гори» – сектори сірого кольору, втрата 3 одиниць ресурсу

Приклад ігрового поля зображено на рисунку 3.1.

Необхідно створити завдання, виконання якого буде умовою перемоги гравця/гравців, та ігрові механіки що утворять ігровий процес (див. рис. 3.2).

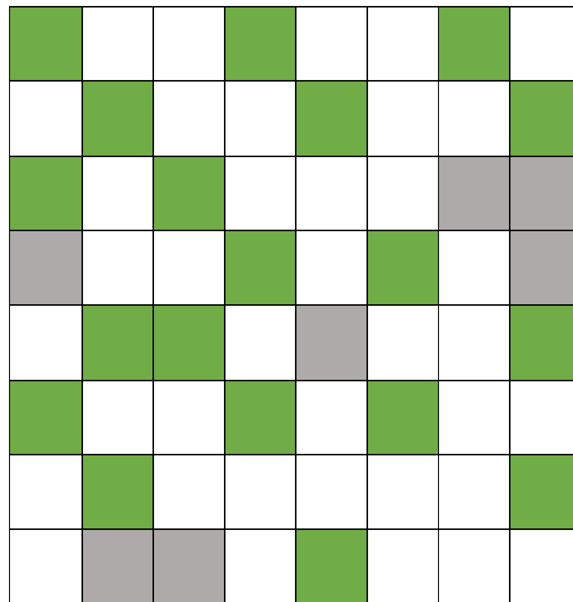


Рисунок 2.1 — Прототип ігрового поля

Джерело: побудовано автором

Нехай умовою перемоги буде «здолати дракона», якого гравцям необхідно знайти на ігровому полі. В процесі його побудови на тайлах будуть з’являтися «точки інтересу», перемістившись на які гравець відкриває «подію» яка на ній знаходиться. Однією з цих подій і буде знахідка «дракона». Для перемоги над ним необхідною умовою буде мати необхідну кількість «спорядження», яке можна отримати за інші види подій на ігровому полі. Нехай тип події гравець дізнається лише після переміщення на сектор, в якому вона знаходиться. Це створить елемент випадковості, що зробить ігровий процес більш варіативним. Події можуть гарантовано винагородити гравця, виступити в ролі перешкоди, чи іншим чином поставити його в менш вигідне становище, що надасть іншим гравцям перевагу. Якщо дати гравцеві можливість обирати активувати «подію» чи ні, це змусить його приймати рішення оцінюючи можливі ризики (наприклад активувати подію в якій негативні наслідки більш

вірогідні ніж успіх, чи спробувати знайти іншу з більшим шансом отримати винагороду).

Для прототипу створимо 3 види подій:

- «Битва з монстром» — перешкода яка сповільнить просування гравця. Маючи достатньо «спорядження» її можна уникнути.
- «Випадкова подія» — гравець може отримати «спорядження» у винагороду або натрапити на «битву з монстром».
- «Підземелля» — послідовність з кількох «випадкових подій», за виконання якої гравець гарантовано отримає винагороду.

Дані ігрові механіки дозволяють реалізувати як кооперативний так і суперницький стилі гри. Принцип розміщення «точок інтересу» на ігровому полі залежить від обраного методу його побудови та вигляду тайлів з яких воно будуватиметься.

3. МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Структурно-функціональне моделювання процесу

Для побудови інформаційної системи створення контенту необхідно визначити ключові зв'язки між нею та її складовими.

IDEF (Integrated Definition) – це графічний метод моделювання процесів всередині системи. Початковим рівнем є IDEF0, який використовується для визначення загального обсягу аналізуємої системи, особливо для функціонального аналізу [39]. Це допоможе визначити які компоненти матимуть вплив на неї та її основні функції (див. рис. 3.1).

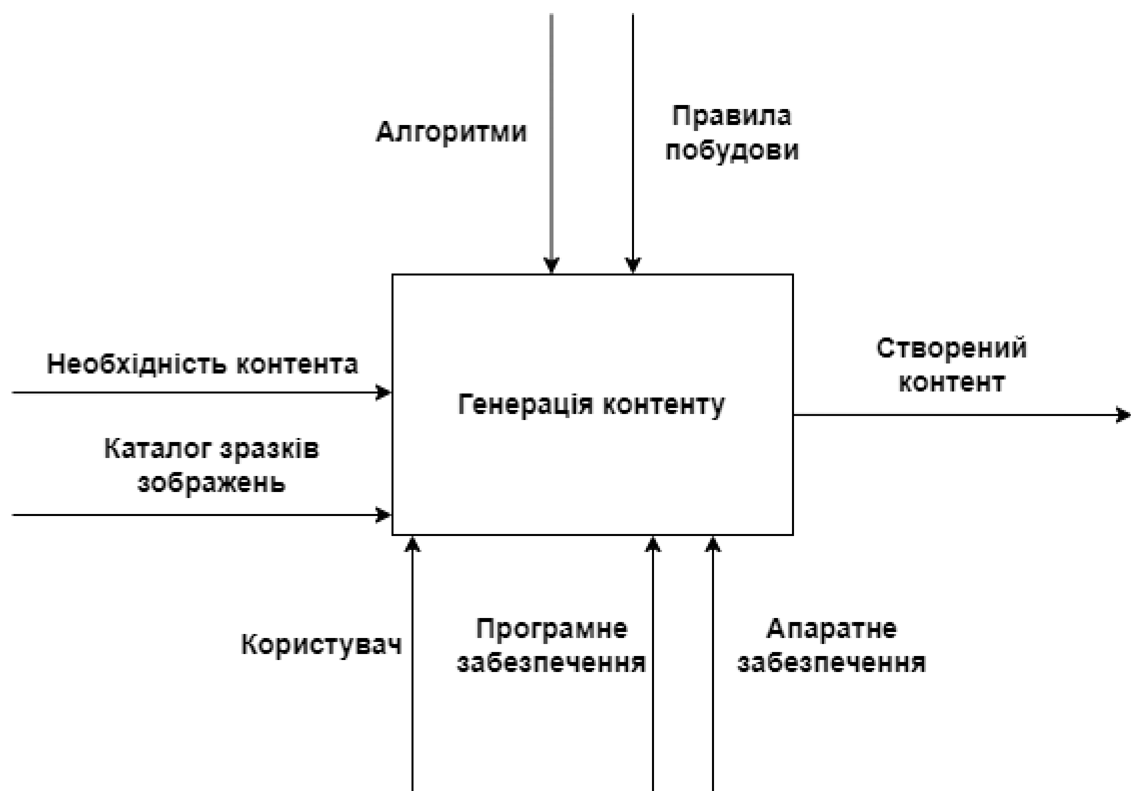


Рисунок 3.1 – Контекстна діаграма інформаційної системи

Джерело: побудовано автором

Додатково необхідно зробити декомпозицію системи на рівні IDEF1 для кращої візуалізації процесів, що в ній відбуватимуться. На даному рівні більш детально відображені основні етапи роботи системи та потоку інформації (див. рис. 3.2).

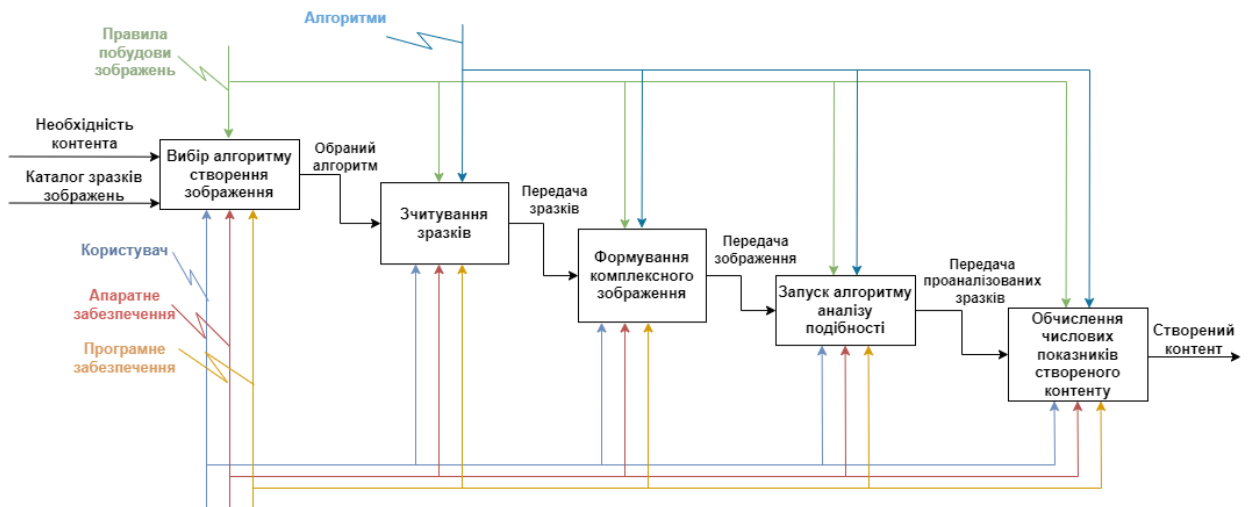


Рисунок 3.2 – Перший рівень декомпозиції

Джерело: побудовано автором

3.2 Моделювання варіантів використання

Для опису очікуваної поведінки системи необхідно побудувати діаграму варіантів використання (Use Case Diagram). Вона детальніше відобразить основні процеси системи з точки зору користувача, який нею буде оперувати. В даній роботі основними процесами інформаційної системи є побудова зображень обраними алгоритмами процедурної генерації і порівняння ознак їх подібності (див. рис. 3.3).

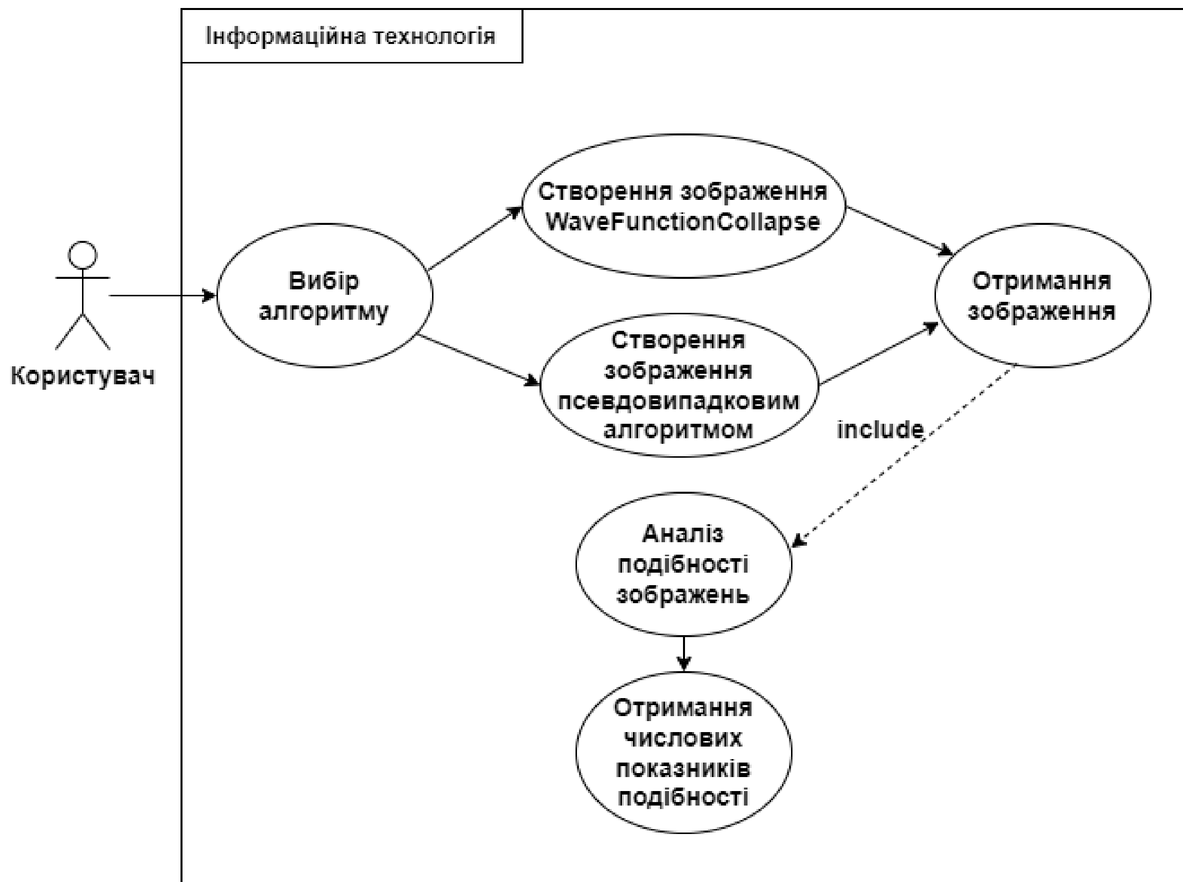


Рисунок 3.3 – Діаграма варіантів використання

Джерело: побудовано автором

3.3 Проектування програмної реалізації алгоритму WFC

Одним з модулів для симуляції побудови ігрового поля буде реалізація алгоритму Wave Function Collapse. Модуль прийматиме на вхід масив зображень, з яких буде будуватися ігрове поле. Кожному елементу масиву буде присвоєно кортеж з міток, які будуть визначати можливих сусідів з кожної з 4 сторін елемента. Далі буде створено порожній масив заданого розміру, кожен елемент якого буде містити значення ентропії відповідного сектору сітки. Після цього обираються сектори з найменшою ентропією, якщо їх більше 1, то випадковим чином обирається один з них. Обраному сектору присвоюється відповідний тайл, якщо можливих варіантів більше 1, то знову випадковим

чином обирається один з них. Значення ентропії в сектора сусідніх до заповненого оновлюються, після чого процес повторюється до того моменту, доки існують пусті сектори. Кінцевий масив перетворюється на зображення, що є результатом роботи алгоритму. Зображення зберігається у форматі .png («portable network graphics»).

Методи для реалізації самого алгоритму та метод запуску роботи алгоритму розділено на окремі класи для спрощення роботи з кодом. Діаграма класів програмного модулю зображена на рисунку 3.4.

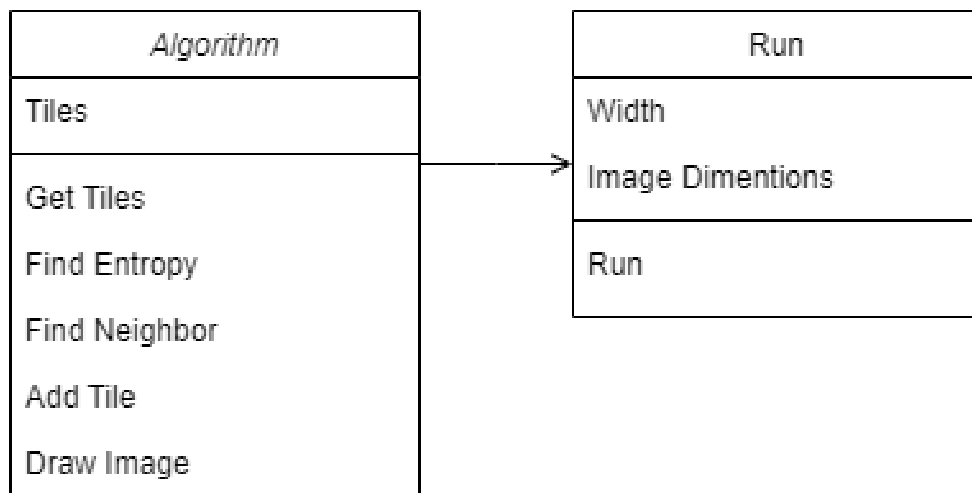


Рисунок 3.4 — Діаграма класів модулю з алгоритмом WFC

Джерело: побудовано автором

3.4 Проектування програмної реалізації псевдовипадкового алгоритму

Якщо прибрати правила розташування тайлів присутні в WFC, додавши фізичні обмеження формату настільної гри, а саме сталу кількість та типи тайлів, та залишивши сталу форму ігрового поля, то процес його побудови можна охарактеризувати як сортування масиву у випадковому порядку. Якщо вважати поле масивом, а тайли – його елементами, з відповідними індексами, до випливає досить очевидна схожість. Даний алгоритм є простішим, але водночас має меншу кількість можливих результатів, що залежить лише від

розміру масиву. Він є простим в реалізації, тому додаткових класів в програмному додатку не потрібно. Діаграма класів програмного модулю зображена на рис. 3.5.

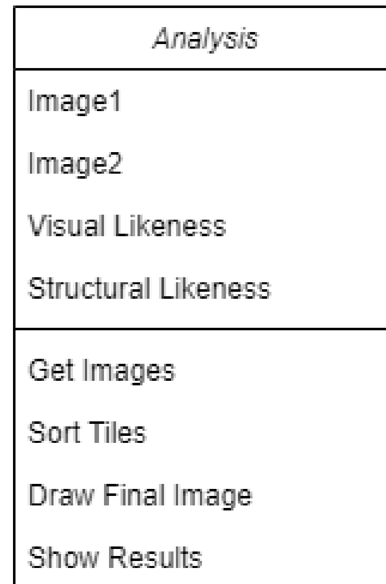


Рисунок 3.5 — Діаграма класів додатку з псевдовипадковим алгоритмом

Джерело: побудовано автором

3.5 Проектування програмної реалізації порівняння зображень

Для визначення різниці результуючих зображень реалізованих алгоритмів проведемо аналіз подібності отриманих зображень. Використаємо 2 критерії для аналізу: подібність візуальних складових та структурну схожість.

Подібність візуальних елементів зображення можливо обчислити за допомогою засобів бібліотеки OpenCV, а саме метод ORB, що знаходить візуальні ознаки зображення та співвідносить їх між собою [27].

Структуру зображень дозволяє проаналізувати метод structural_similarity() бібліотеки Scikit-image. Даний метод порівнює середні значення рівнів чорного та білого пікселів зображень та на їх основі виводить відсоток схожості їх загальної структури [37].

Результатом роботи програмного модулю буде спільне зображення двох порівнюваних, з лініями відповідності візуальних ознак, та числовими значеннями їх візуальної та структурної схожості. Діаграма класів програмного модулю зображена на рис. 3.6

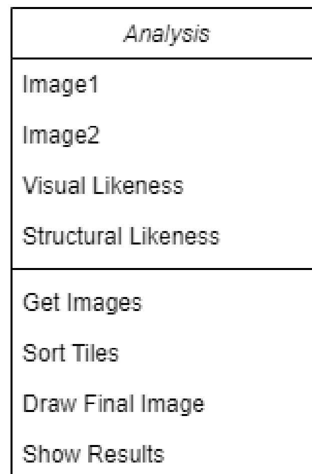


Рисунок 3.6 — Діаграма класів додатку порівняння зображень

Джерело: побудовано автором

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Визначення ігрових параметрів для аналізу

Як було зазначено у пункті 2.4, гра повинна містити механіки що імплементують алгоритми процедурної генерації. Для прототипу були визначені наступні ігрові завдання:

Мета гри – знайти та перемогти «дракона».

Умови досягнення мети – знаходження достатньої кількості «спорядження», знаходження «дракона» на ігровому полі.

Засоби виконання умов – пересування ігровим полем, взаємодія з «точками інтересу».

Зважаючи на ігрові завдання визначено наступний ігровий сценарій процесу:

1. Кожен гравець в свій хід пересувається ігровим полем, при виході за межі існуючого поля гравець докладає новий його сегмент (тайл).
2. При викладенні нового тайла на ньому ж розміщуються «точки інтересу», з якими гравець зможе взаємодіяти.
3. При взаємодії з «точками інтересу» гравець може отримати «спорядження», необхідне для виконання умови перемоги, або натрапити на перешкоду, що сповільнить гравця, надавши перевагу іншим учасникам.
4. Кожен гравець має на початку ходу сталу кількість ігрового ресурсу, який витрачається на пересування та взаємодію з «точками інтересу». Хід гравця закінчується тоді, коли він витратить весь ігровий ресурс, або самостійно вирішить завершити його раніше.

Єдиною ігровою механікою, що використовує процедурну генерацію є побудова ігрового поля. Інші механіки не мають впливу на побудову поля, а навпаки залежать від неї. Тож для їх реалізації необхідно спочатку визначитись з процесом побудови ігрового поля.

Є 2 головних параметри, що впливають на даний процес: вигляд елементів поля (тайлів) та принцип їх поєднання. Необхідно визначити варіанти їх реалізації для проведення подальшого аналізу варіативності.

4.1.1 Вигляд тайлів ігрового поля

Для побудови поля використовуються квадратні тайли, тож саме поле матиме вигляд квадратної сітки. Найменший можливий розмір тайлу, або його внутрішньої розмітки, буде відповідати одиничному сектору поля. Одиничні сектори, як зазначено в пункті 2.4, можуть відповідати 3 типам місцевості, для пересування якими гравець витрачатиме різну кількість ігрового ресурсу (див. рис. 4.1).

Рисунок 4.1 – Типи одиничних секторів ігрового поля

Джерело: побудовано автором

Вигляд місцевості має надавати гравцеві можливість вибору оптимального шляху пересування. Розмірність тайлів 1x1 у даному випадку не реалізовує дану можливість взагалі. Оскільки нові тайли обираються випадково, та викладаються лише при пересуванні за межі поля, у гравця не буде контролю над тим, на яку місцевість він потрапить (див. рис. 4.2).

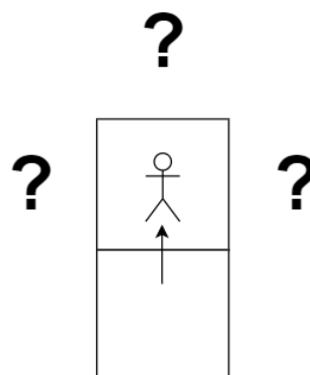


Рисунок 4.2 – Варіанти пересування при розмірності 1x1

Джерело: побудовано автором

Отже необхідно збільшити розмірність тайлів для впровадження елементу тактики при пересуванні полем. На рисунках 4.3-4.4 зображено варіанти пересування при розмірі тайлів 2x2 та 3x3.

При розмірі 2x2 у гравця фактично буде змога просунутись на 1 сектор, після чого знову з’явиться необхідність виставляти новий тайл.

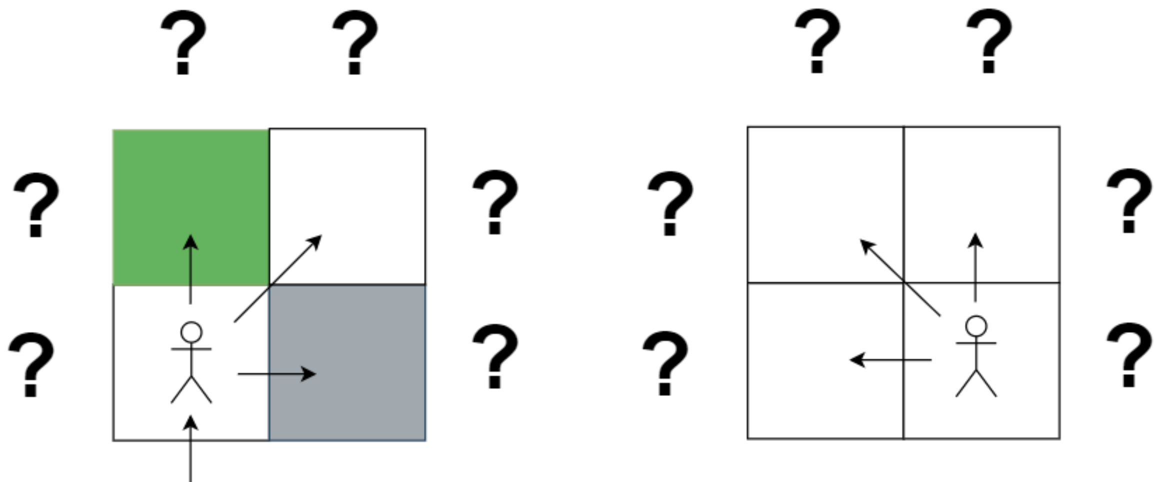


Рисунок 4.3 – Варіанти пересування при розмірності 2x2

Джерело: побудовано автором

Розмір 3x3 дозволяє створити ситуації, де можливо кілька варіантів пересування. З крайніх секторів гравець матиме кілька шляхів незалежно від того, почне він з кутового чи крайнього середнього сектора. З кутового сектора він, в додаток до можливості одразу пересунутись за межу та поставити новий тайл, може обрати просування до протилежного краю поточного тайлу. При цьому даний шлях можливо здійснювати кількома способами які можуть бути більш або менш оптимальними в залежності від «типів місцевості». Тому у гравця може бути до 3 варіантів вибору власного шляху пересування, що задовольняє потребу наявності елементу тактики.

Таким чином будемо вважати 3x3 мінімальною розмірністю, що задовольняє дану потребу.

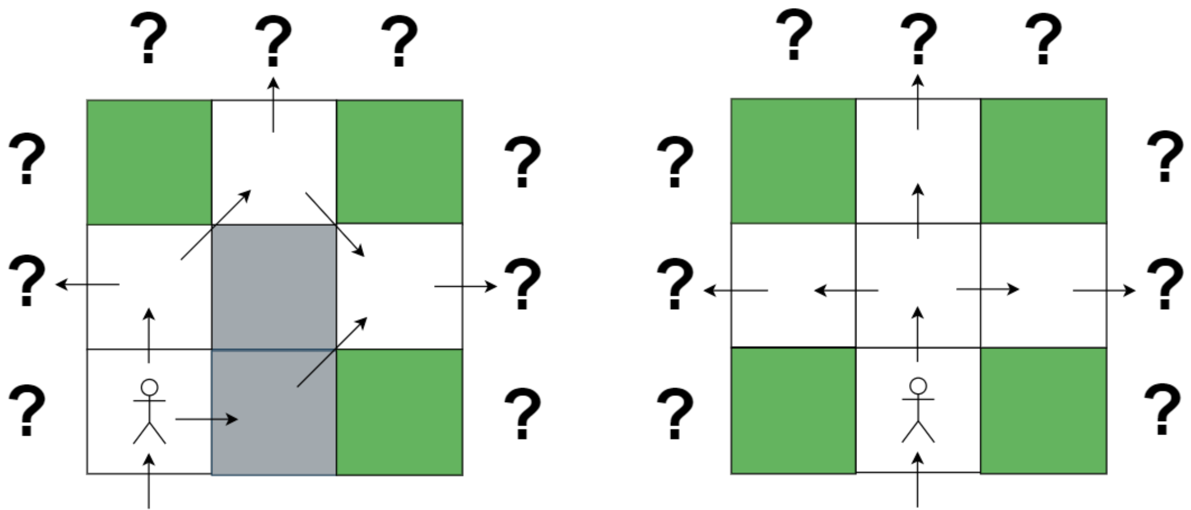


Рисунок 4.4 – Варіанти пересування при розмірності 3х3

Джерело: побудовано автором

4.1.2 Правила побудови ігрового поля

Ігрове поле являє собою двовимірну сітку, тож незалежно від способу побудови розмір тайлів та їх внутрішньої розмітки мають дотримуватись одного масштабу. У створеному прототипі припускається, що всі тайли мають однаковий розмір та внутрішню розмітку, тож дана умова виконується.

Першим способом поєднання квадратних тайлів є з'єднання «ребром до ребра». Даний спосіб може бути застосованим до тайлів з будь якою розмірністю внутрішньої сітки (див. рис. 4.5).

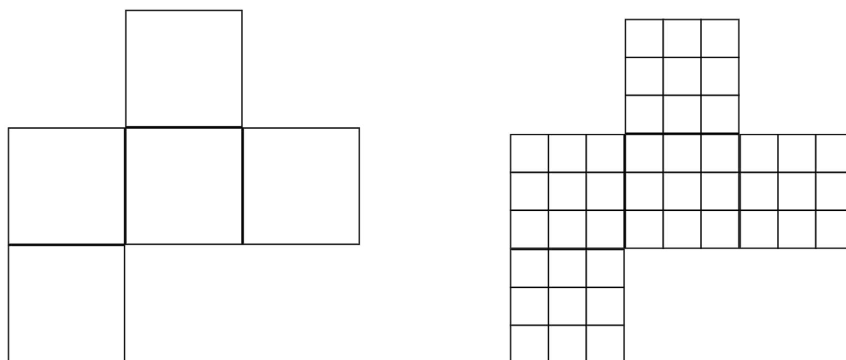


Рисунок 4.5 – Поєднання тайлів «ребром до ребра»

Джерело: побудовано автором

Іншим способом є побудова у довільному порядку зі збереженням сітки. Даний спосіб працює лише для тайлів розмірністю 2х2 і більше, оскільки для розмірності 1х1 він фактично не відрізняється від попереднього. При побудові цим способом ми орієнтуємося не на ціле ребро тайлу а на його внутрішню розмітку (див. рис. 4.6). Таким чином відносно внутрішньої розмітки, тайли правильно формують сітку, але відносно один одного вони можуть бути зсунутими в сторону з'єднуючись лиш частиною ребра.

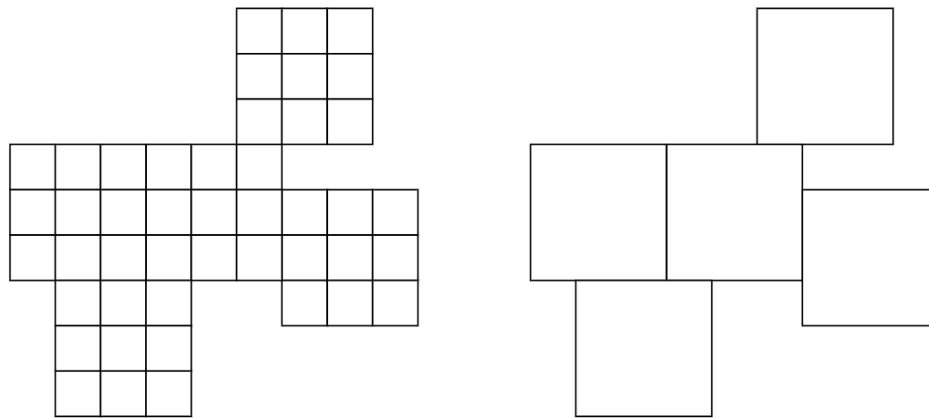


Рисунок 4.6 – Поєднання тайлів методом «довільної фігури»

Джерело: побудовано автором

4.2 Аналіз варіативності побудови ігрового поля

Параметр варіативності побудови ігрового поля є кількістю всіх можливих комбінацій взаємного розташування його складових. При наявності одного тайла можемо визначити кількість варіантів розташування наступного. Спершу розглянемо спосіб «ребром до ребра».

4.2.1 Метод «ребром до ребра»

Кількість варіантів розташування наступного елементу при даному способі дорівнює кількості ребер фігури до якої елемент доєднується. Для

квадратного тайлу дане число рівне 4-ом. На рисунку 4.7 бачимо, що для 2-го елементу маємо 4 варіанти розташування. Незалежно від обраного варіанту отримаємо фігуру що матиме 6 варіантів для наступного елементу.

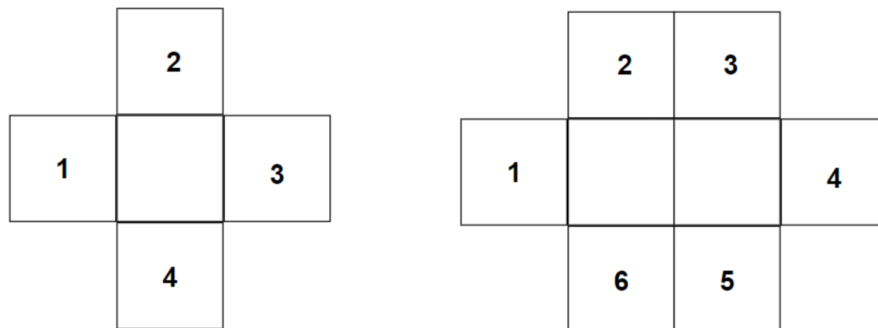


Рисунок 4.7 – Варіанти розташування 2-го елемента «ребром до ребра»

Джерело: побудовано автором

При подальших кроках ситуація змінюється в залежності від побудови фігури. Як бачимо з рисунку 4.8 при розташуванні елементів в ряд для 5-го елементу отримаємо 10 варіантів, проте при розташуванні квадратом маємо 8 варіантів. Дані приклади надають найбільшу та найменшу кількість варіантів розташування наступного елементу відповідно, не залежно від ітерації.

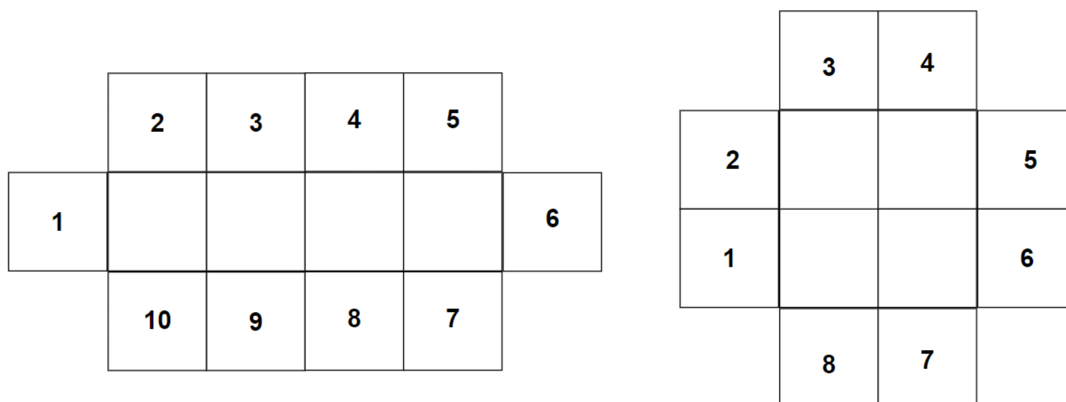


Рисунок 4.8 – Варіанти розташування 5-го елемента «ребром до ребра»

Джерело: побудовано автором

Отже з даних закономірностей можемо вивести наближені формули мінімуму та максимуму можливих варіантів розташування n-ного елементу.

Нехай:

N – кількість можливих варіантів розташування;

n – порядковий номер елемента;

k – кількість ребер.

Тоді для n=2

$$N_{min} = k$$

$$N_{max} = k$$

Для n>2

$$N_{min} = k + (k-3)(n-1)$$

$$N_{max} = k + (k-2)(n-2)$$

Таблиця 4.1 – Кількість варіантів розміщення n-го елемента методом «ребро до ребра»

n	**«Ребром до ребра»**	
	min	**max**
2	4	4
3	6	6
4	7	8
5	8	10
6	9	12
7	10	14
8	11	16
9	12	18
10	13	20
11	14	22
12	15	24
13	16	26
14	17	28
15	18	30
16	19	32
17	20	34
18	21	36
19	22	38
20	23	40
21	24	42
22	25	44
23	26	46
24	27	48
25	28	50

4.2.2 Метод «довільної фігури»

Кількість варіантів розташування наступного елементу при даному способі залежить від внутрішньої розмітки тайлу. На рисунку 4.9 зображені варіанти розташування вздовж ребра тайлу розмірністю 3х3. Можна зробити припущення що при розмірності *x* дана кількість дорівнюватиме *x+(x-1)*.

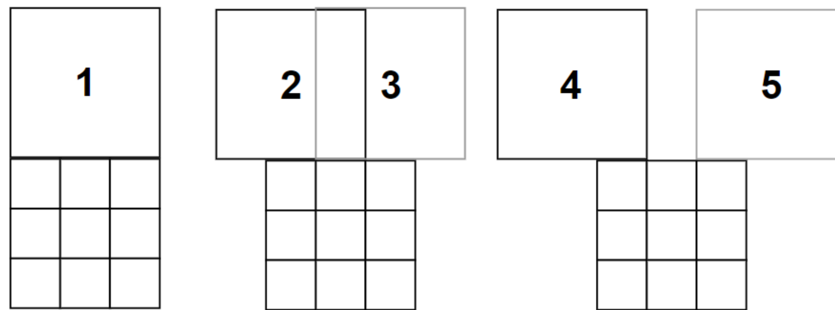


Рисунок 4.9 – Варіанти розташування 2-го елемента вздовж ребра тайла розмірністю 3х3

Джерело: побудовано автором

Тоді при розмірності 4х4 кількість варіантів вздовж одного ребра буде дорівнювати 4+(4-1)=7 (див. рис. 4.10).

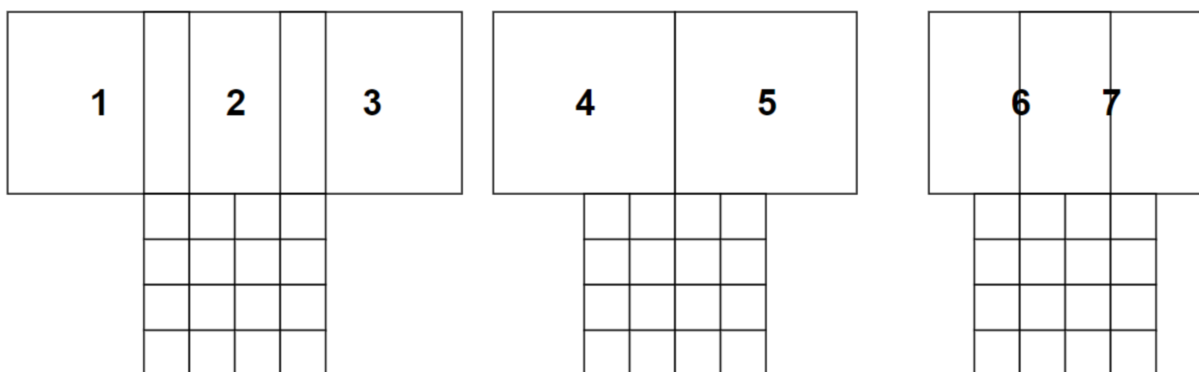


Рисунок 4.10 – Варіанти розташування 2-го елемента вздовж ребра тайла розмірністю 4х4

Джерело: побудовано автором

Знаючи дану закономірність можемо вивести наближені формули кількості можливих варіантів розташування n-го елементу для розмірності x.

Нехай:

N – кількість можливих варіантів розташування;

n – порядковий номер елемента;

k – кількість ребер;

x – розмірність внутрішньої сітки.

Тоді для n=2

$$N = k\big(x + (x - 1)\big)$$

Для n>2

$$N_{min} = (kn - k)(2x - 1) - (2x - 1)(3n - 7{,}5) - (x - 1)(n - 0{,}5)$$

$$N_{max} = (kn - k)(2x - 1) - 4x(n - 2)$$

В таблиці 4.2 приведені розрахунки мінімуму та максимуму можливих варіантів розташування n-го елементу методом «довільної фігури».

В таблиці 4.3 приведені розрахунки, що наближено відображають середню кількість можливих варіантів розташування для обох розглянутих методів. З результатів бачимо, що різниця можливих варіантів при використанні методу «довільної фігури» для розмірності 3x3 перевищує метод «ребро до ребра» майже в 4 рази. Проте з подальшим збільшенням розмірності дана різниця зменшується. Так розмірність 4x4 має лише в 1,4 рази більше варіантів розташування ніж 3x3. Для 5x5 – 1,3 рази від 4x4. Дані таблиці графічно подані у вигляді точкової діаграми, зображеної на рисунку 4.10.

Таблиця 4.2 – Кількість варіантів розміщення n-го елемента методом «довільної фігури»

n	**3x3**		**4x4**		**5x5**	
	min	**max**	**min**	**max**	**min**	**max**
2	20	20	28	28	36	36
3	27,5	28	38	40	48,5	52
4	30,5	36	42	52	53,5	68
5	33,5	44	46	64	58,5	84
6	36,5	52	50	76	63,5	100
7	39,5	60	54	88	68,5	116
8	42,5	68	58	100	73,5	132
9	45,5	76	62	112	78,5	148
10	48,5	84	66	124	83,5	164
11	51,5	92	70	136	88,5	180
12	54,5	100	74	148	93,5	196
13	57,5	108	78	160	98,5	212
14	60,5	116	82	172	103,5	228
15	63,5	124	86	184	108,5	244
16	66,5	132	90	196	113,5	260
17	69,5	140	94	208	118,5	276
18	72,5	148	98	220	123,5	292
19	75,5	156	102	232	128,5	308
20	78,5	164	106	244	133,5	324
21	81,5	172	110	256	138,5	340
22	84,5	180	114	268	143,5	356
23	87,5	188	118	280	148,5	372
24	90,5	196	122	292	153,5	388
25	93,5	204	126	304	158,5	404

Джерело: побудовано автором

Таблиця 4.3 – Середня кількість варіантів розміщення n-го елемента

n	Варіанти побудови поля з n тайлів			
	Ребром до ребра	Довільна фігура		
		3x3	4x4	5x5
2	4	20	28	36
3	6	27,75	39	50,25
4	7,5	33,25	47	60,75
5	9	38,75	55	71,25
6	10,5	44,25	63	81,75
7	12	49,75	71	92,25
8	13,5	55,25	79	102,75
9	15	60,75	87	113,25
10	16,5	66,25	95	123,75
11	18	71,75	103	134,25
12	19,5	77,25	111	144,75
13	21	82,75	119	155,25
14	22,5	88,25	127	165,75
15	24	93,75	135	176,25
16	25,5	99,25	143	186,75
17	27	104,75	151	197,25
18	28,5	110,25	159	207,75
19	30	115,75	167	218,25
20	31,5	121,25	175	228,75
21	33	126,75	183	239,25
22	34,5	132,25	191	249,75
23	36	137,75	199	260,25
24	37,5	143,25	207	270,75
25	39	148,75	215	281,25

Джерело: побудовано автором

В таблиці 4.4 приведені розрахунки загальної кількості можливих варіантів побудови поля з n елементів, що є добутком варіантів розташування n-го елемента.

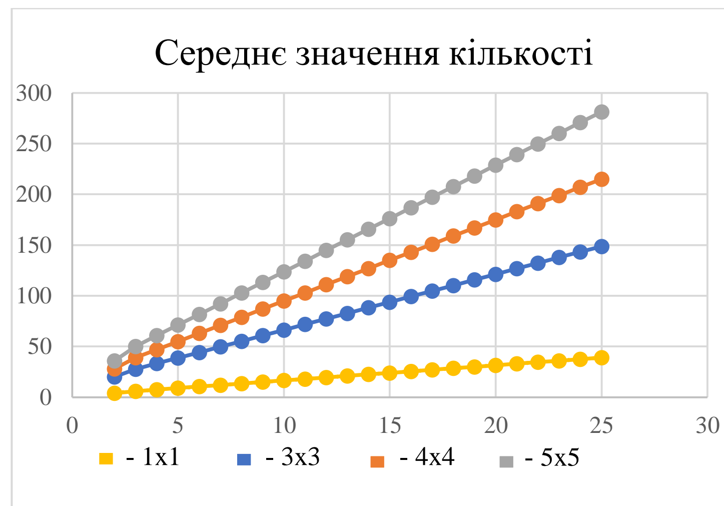


Рисунок 4.10 – Діаграма залежності варіантів розташування n-го елементу від розмірності для методу «довільної фігури»

Джерело: побудовано автором

Таблиця 4.4 – Загальна кількість варіантів побудови поля з n елементів

n	Варіанти побудови поля з n елементів			
	Ребром до ребра	Довільна фігура		
		3x3	4x4	5x5
2	4	20	28	36
3	24	555	1092	1809
4	180	19980	51324	109896,75
5	1620	879120	2822820	7830143,438
6	17010	45714240	177837660	640114226
7	204120	2742854400	12626473860	59050537350
8	2755620	1,86514E+11	9,97491E+11	6,06744E+12
9	41334300	1,41751E+13	8,67818E+13	6,87138E+14
10	682015950	1,19071E+15	8,24427E+15	8,50333E+16
11	12276287100	1,09545E+17	8,49159E+17	1,14157E+19
12	2,39388E+11	1,09545E+19	9,42567E+19	1,65243E+21
13	5,02714E+12	1,18309E+21	1,12165E+22	2,56539E+23
14	1,13111E+14	1,37238E+23	1,4245E+24	4,25214E+25
15	2,71466E+15	1,70175E+25	1,92308E+26	7,49439E+27
16	6,92237E+16	2,24631E+27	2,75E+28	1,39958E+30
17	1,86904E+18	3,14483E+29	4,1525E+30	2,76067E+32
18	5,32676E+19	4,65435E+31	6,60248E+32	5,73528E+34
19	1,59803E+21	7,26079E+33	1,10261E+35	1,25173E+37
20	5,03379E+22	1,19077E+36	1,92957E+37	2,86332E+39
21	1,66115E+24	2,04812E+38	3,53112E+39	6,8505E+41
22	5,73097E+25	3,68662E+40	6,74444E+41	1,71091E+44
23	2,06315E+27	6,93085E+42	1,34214E+44	4,45265E+46
24	7,73681E+28	1,35845E+45	2,77824E+46	1,20555E+49
25	3,01736E+30	2,77123E+47	5,97321E+48	3,39062E+51

При однаковій кількості тайлів та розміру одиничного сектору сітки, збільшення розмірності буде впливати на розмір ігрового поля, що в свою чергу збільшить тривалість ігрової сесії. Так поле з 25 тайлів розмірності 3х3 матиме 225 секторів, поле з тієї ж кількості тайлів, але розмірності 4х4 матиме 400 секторів. Таким чином кінцеве поле буде майже в 1,5 рази більшим, що для формату настільної гри може бути суттєвим за наявності обмеженого місця для проведення гри.

Враховуючи дані показники будемо вважати що для розмірності тайлів 4х4 і більше, збільшення варіативності побудови поля не є достатньо суттєвим, щоб знехтувати наслідками що негативно вплинуть на ігровий процес та дизайн технологічного забезпечення. Отже для потреб прототипу та особливостей ігрового процесу достатнім та оптимальним буде розмірність тайлів 3х3.

4.3 Реалізація програмних модулів

Обидва описаних способи побудови ігрового поля реалізовуються певним алгоритмом процедурної генерації. Алгоритм Wave Function Collapse (WFC) вимагає чітких обмежень того, які елементи поля можуть поєднуватись між собою, в той час, як псевдовипадковий алгоритм, таких обмежень не має. Зважаючи на це, WFC потребуватиме специфічного дизайну набору елементів поля, щоб під час його побудови не виникало ситуації в яких певні елементи неможливо буде поєднати. До того ж кінцевий вигляд ігрового поля може бути одноманітним, та обмежувати тактичність пересування гравця. Якщо поля, побудовані різними алгоритмами не будуть суттєво відрізнятись один від одного, задовольняючи потреби ігрового процесу, тоді оптимальним буде використання простішого в реалізації алгоритму.

Для аналізу даного аспекту створено програмні модулі, що реалізовують алгоритми Wave Function Collapse та псевдовипадковий, а також модуль що порівнює візуальні та структурні ознаки зображень на подібність одне одному. Модулі створені програмними засобами мови Python. Для простоти та узгодженості аналізу алгоритми будують квадратні зображення ігрового поля, що відповідає способу побудови «ребро до ребра».

4.3.1 Модуль Wave Function Collapse

Алгоритм Wave Function Collapse присвоює кожному зразку кортеж з міток, які будуть визначати можливих сусідів з кожної з 4 сторін елемента. Код програмного модулю наведений в Додатках Б-В.

Створимо початковий набір з 5 зразків вхідних зображень для побудови поля алгоритмом WFC (див. рис. 4.11).

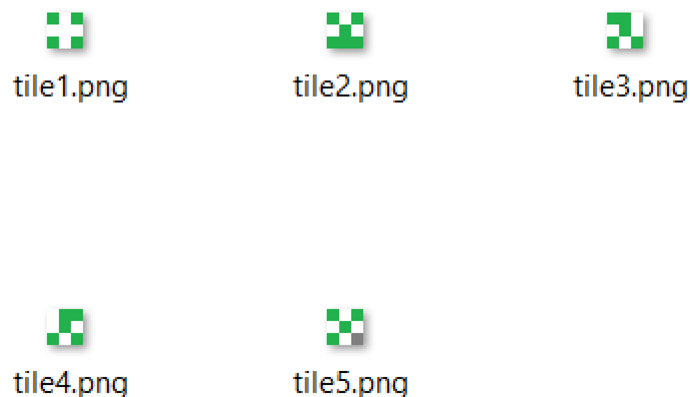


Рисунок 4.11 – Початковий набір зразків зображень (5 зразків)

Джерело: побудовано автором

Для визначення міток зразків зображень введемо наступні позначення:

- A – сектор з типом місцевості «рівнини»;
- B – сектор з типом місцевості «ліс»;
- C – сектор з типом місцевості «гори»;

Таким чином даний набір зображень матиме наступний словник міток:

"tile1.png": ("BAB", "BAB", "BAB", "BAB"),

"tile2.png": ("BAA", "BAB", "BBB", "BAB"),

"tile3.png": ("BBA", "AAB", "BAB", "BAB"),

"tile4.png": ("ABB", "BAB", "BAB", "BAA"),

"tile5.png": ("BAB", "BAC", "CAB", "BAB"),

Створимо комплексне зображення розміром 5x5. В результаті роботи алгоритму WFC отримуємо поле, зображене на рисунку 4.12

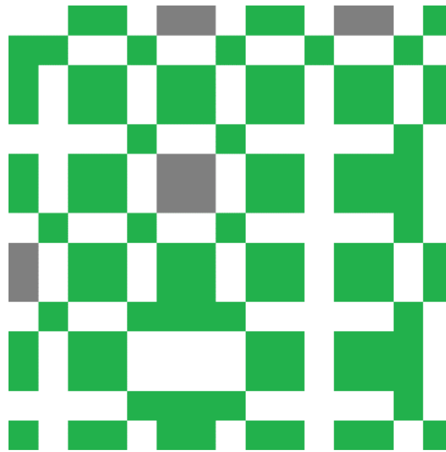


Рисунок 4.12 – Результат роботи алгоритму Wave Function Collapse з набором з 5 зразків

Джерело: побудовано автором

Отримане зображення має багато повторюваних візерунків та є досить одноманітним, що не задовольняє потреб ігрового прототипу. Спробуємо збільшити кількість зразків до 10 (див. рис. 4.13).

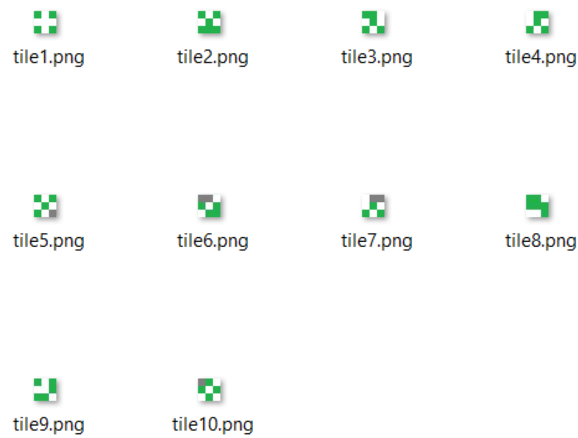


Рисунок 4.13 – Початковий набір зразків зображень (10 зразків)

Джерело: побудовано автором

Зі збільшенням кількості зразків складніше створити набір міток, що включатиме всі можливі варіанти поєднань. Виникають ситуації в яких неможливо завершити зображення, оскільки алгоритм не може знайти правильний зразок. В результаті досі отримуємо одноманітні зображення, які часто не використовують певних зразків.

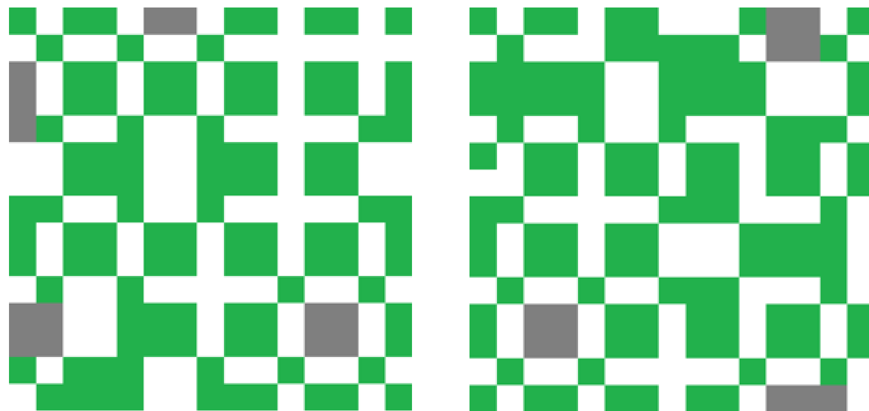


Рисунок 4.14 – Результат роботи алгоритму Wave Function Collapse з набором з 10 зразків

Джерело: побудовано автором

Подальше збільшення кількості зразків не дозволяє згенерувати зображення, оскільки алгоритм щоразу стикається з ситуаціями за яких неможливо завершити зображення за заданими правилами побудови.

Спробуємо спростити правила побудови, дозволивши секторам «рівнин» з'єднуватися з «лісом». Для зменшення повторюваних візерунків змінено дизайн та додано більше зразків (див. рис. 4.15). В результаті отримуємо поле, зображене на рисунку 4.16.

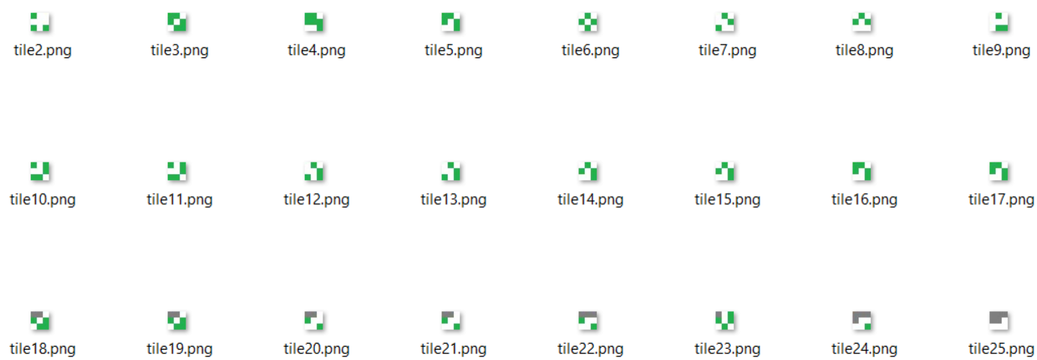


Рисунок 4.15 – Початковий набір зразків зображень (23 зразки)

Джерело: побудовано автором

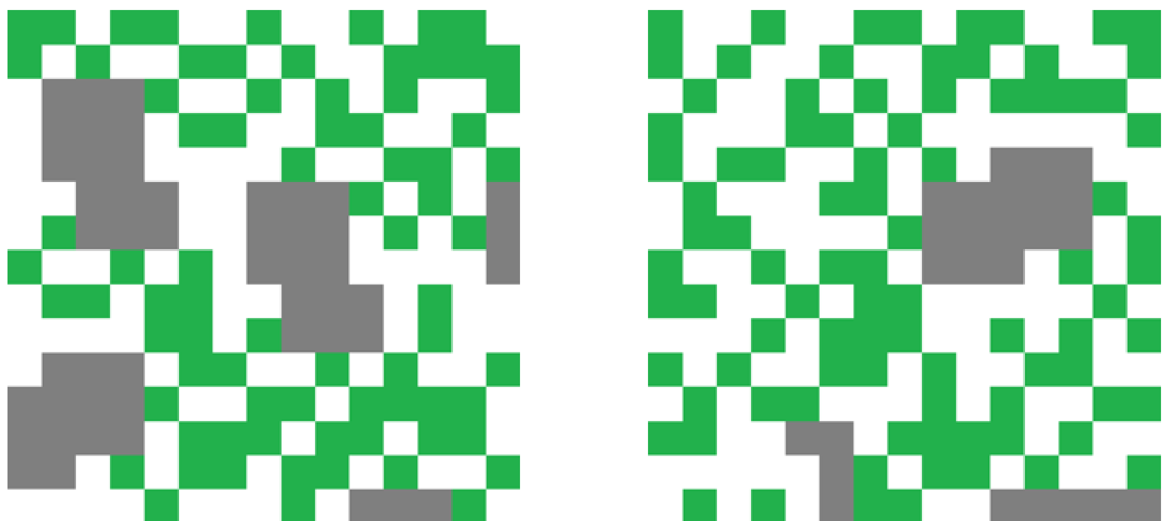


Рисунок 4.16 – Результат роботи алгоритму
Wave Function Collapse з набором з 23 зразків

Джерело: побудовано автором

Як бачимо зображення мають більш варіативну структуру, проте досить часто можна отримати великі ділянки з «горами». Присвоївши всім зразкам однакові мітки, алгоритм здатен створювати поле у вигляді, зображеному на рисунку 4.17. В даному варіанті найкраще збалансоване розміщення та вигляд секторів різної місцевості, що задовольняє потребам ігрового прототипу.

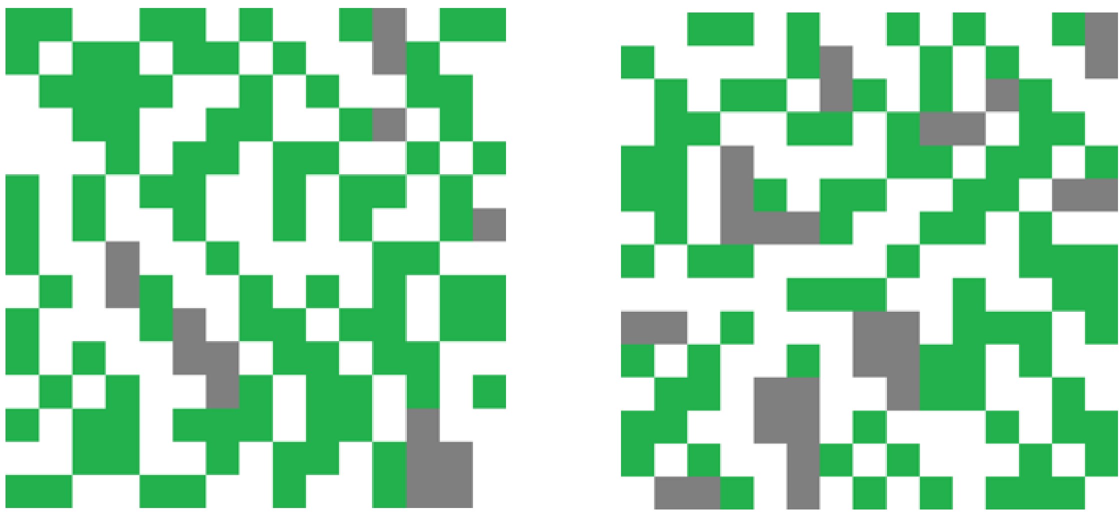


Рисунок 4.17 – Результат роботи алгоритму Wave Function Collapse з однаковими мітками для всіх зразків

Джерело: побудовано автором

4.3.2 Модуль з псевдовипадковим алгоритмом

Псевдовипадковий алгоритм, на відміну від алгоритму Wave Function Collapse, завжди використовує всі вхідні зображення. Фактично, ігрове поле будується випадковим сортуванням масиву цих зображень (див. рис. 4.18).

В результаті роботи алгоритму отримуємо зображення, що також мають збалансоване розміщення типів місцевості ігрового поля (див. рис. 4.19). Помаранчевий сектор на зображенні є частиною тайлу, що викладається першим на початку гри, та від якого будується ігрове поле під час гри. В звичній ігровій ситуації він відображав би центр ігрового поля, проте під час генерації алгоритм може розміщувати його у крайній частині зображення. Код програмного модулю наведений в Додатку Г.

4	18	14	15	22
19	11	23	2	25
8	1	12	24	5
13	21	3	17	10
6	9	16	20	7

Рисунок 4.18 – Принцип роботи псевдовипадкового алгоритму

Джерело: побудовано автором

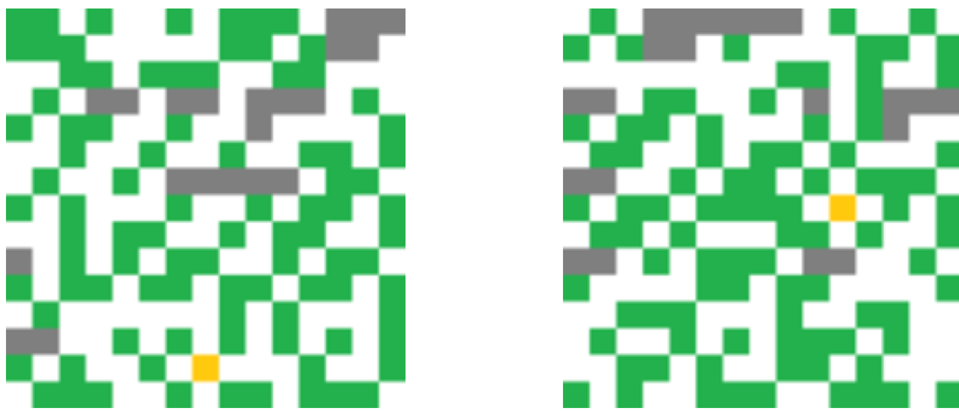


Рисунок 4.19 – Результат роботи псевдовипадкового алгоритму

Джерело: побудовано автором

4.3.3 Модуль аналізу подібності зображень

Для визначення подібності результатів побудови ігрового поля різними алгоритмами створено програмний модуль, що проводить аналіз візуальних ознак та структури вхідних зображень. Код програмного модулю наведений в Додатку Ґ.

Візуальна схожість розраховується методом ORB (Oriented FAST and Rotated BRIEF), що є засобом бібліотеки OpenCV мови Python. Даний метод знаходить ключові візуальні елементи зображень та порівнює їх. Коефіцієнт подібності 0 означає повну невідповідність, 0.5 – часткову відповідність, 1 – повну ідентичність. Задовільним будемо вважати коефіцієнт 1.

Структурна схожість розраховується методом SSIM (Structural Similarity Index), що реалізовується засобами бібліотеки Scikit-image мови Python. Коефіцієнт подібності визначає відсоток схожості загальної структури зображення. Оскільки метод порівнює середні значення рівнів чорного та білого пікселів, зображення необхідно перетворювати на відтінки сірого. Задовільною будемо вважати структурну подібність $\geq 75\%$.

Спочатку порівняємо ігрові поля побудовані кожним алгоритмом окремо. На рисунку 4.20 зображено порівняння двох варіантів ігрового поля створених алгоритмом Wave Function Collapse. Оскільки зображення фактично складаються з пікселів трьох кольорів, то візуальні елементи повторюються досить часто, а отже коефіцієнт ORB завжди виходить рівним 1, тобто поодинокі елементи є ідентичними для обох зображень. Структурна схожість в середньому дає показник 0,84, що дорівнює 84%, отже зображення мають сильну структурну подібність. Для зображень на рисунку 4.20 цей показник рівний 82,6%.

Порівняння зображень створених псевдовипадковим алгоритмом теж дає коефіцієнт ORB рівний 1. Структурна схожість в середньому складає 76%, що на 8% нижче ніж результати алгоритму WFC. Для зображень на рисунку 4.21 цей показник рівний 78,9%. Дана різниця може бути спричиненою тим, що алгоритм Wave Function Collapse може повторно використовувати одні і ті ж зразки декілька разів, а деякі не використати взагалі, тому зображення виходять більш однорідними.

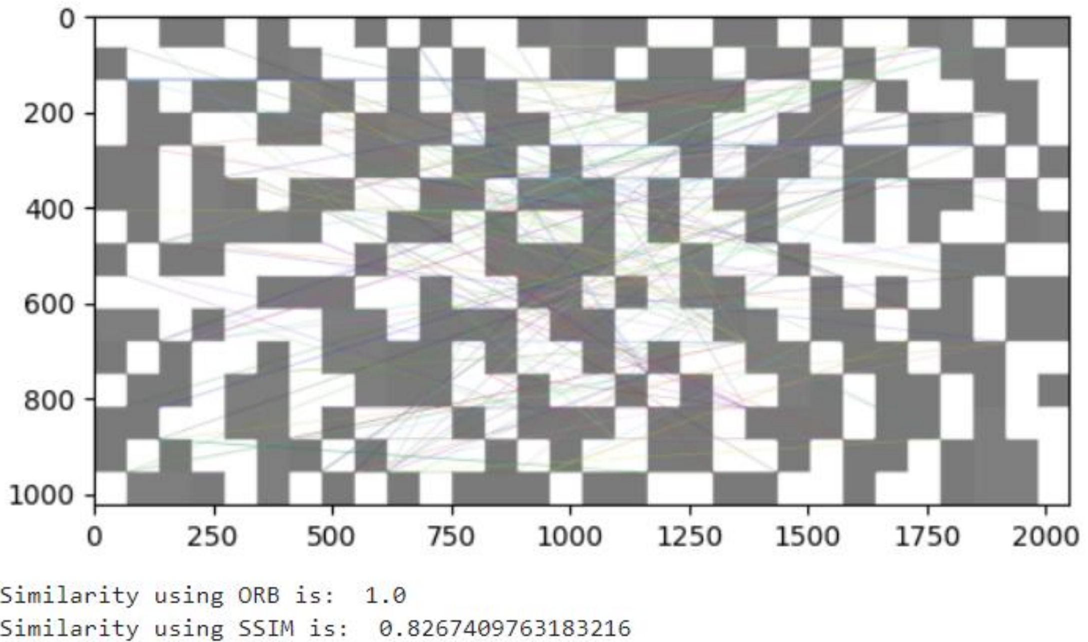


Рисунок 4.20 – Порівняння зображень побудованих алгоритмом WFC

Джерело: побудовано автором

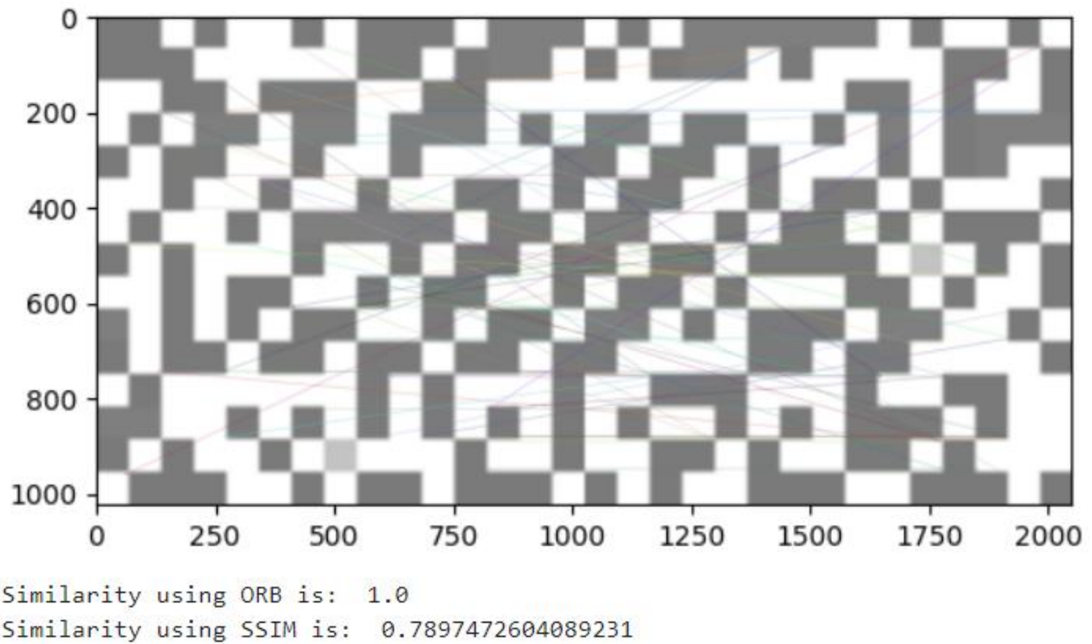


Рисунок 4.21 – Порівняння зображень побудованих псевдовипадковим алгоритмом

Джерело: побудовано автором

Тепер порівняємо між собою зображення створені різними алгоритмами. Коефіцієнт ORB рівний 1. Структурна схожість в середньому складає 75,9%.

Для зображень на рисунку 4.21 цей показник рівний 76,8%. Як бачимо показники відрізняються несуттєво.

В результаті аналізу випливає висновок, що при генерації ігрового поля псевдовипадковим алгоритмом та алгоритмом Wave Function Collapse, використовуючи однаковий набір вхідних зразків, отримуємо зображення що мають сильну візуальну та структурну подібність, що несуттєво відрізняється від вибору алгоритму.

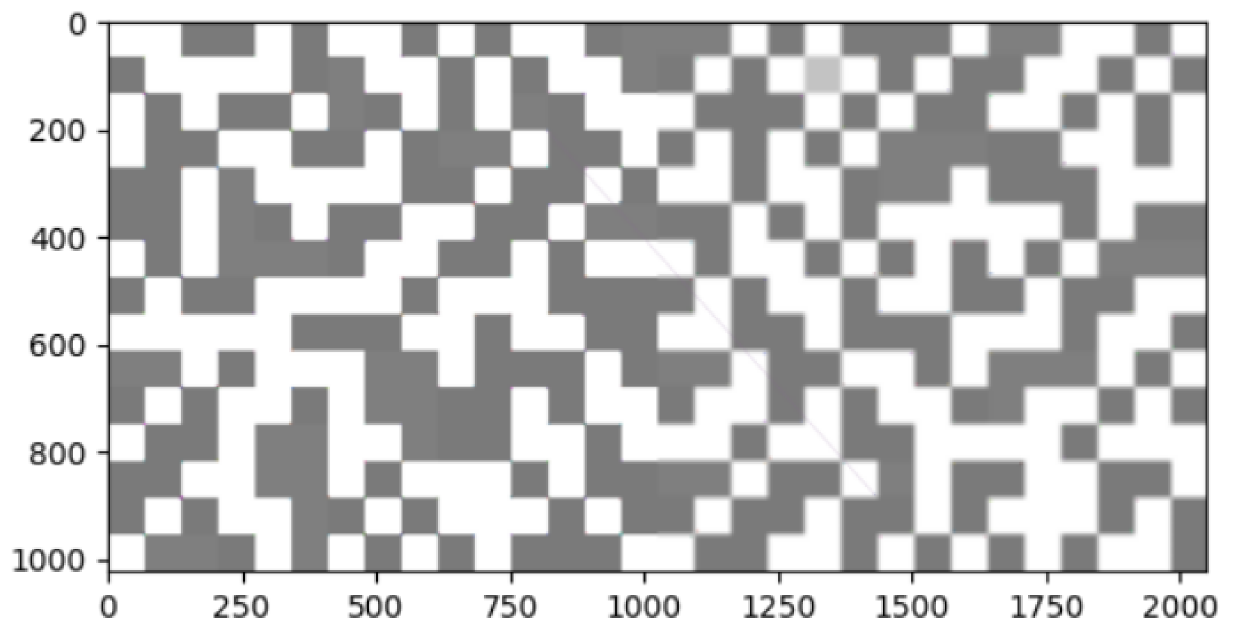


Рисунок 4.21 – Порівняння зображень побудованих псевдовипадковим алгоритмом та алгоритмом WFC

Джерело: побудовано автором

4.4 Завершення ігрового прототипу

Проаналізувавши варіанти побудови ігрового поля зроблено наступні висновки:

1. Метод поєднання «довільної фігури» для тайлів з внутрішньою розміткою значно збільшує варіативність побудови ігрового поля.
2. Кількість елементів для побудови поля впливає на варіативність сильніше ніж метод їх поєднання та розмірність їх внутрішньої розмітки.
3. Алгоритм Wave Function Collapse найкраще працює при наявності невеликої кількості типів з'єднання елементів.
4. При використанні однакового набору зразків/елементів для побудови псевдовипадковий алгоритм та алгоритм Wave Function Collapse створюють зображення, що загалом мають сильну структурну та візуальну схожість.

На основі цього можемо визначити параметри та особливості реалізації механіки побудови ігрового поля, які краще пасують концепції власного ігрового прототипу:

1. Поле будується з тайлів розмірністю 3х3.
2. Тайли з'єднуються відносно їх внутрішньої розмітки методом «довільної фігури».
3. Тайли можна з'єднувати довільно, не залежно від типу з'єднуваних секторів.

Побудова поля починається зі стартового тайлу, центральний сектор якого має символ «багаття». Кількість використовуваних тайлів та інших ігрових елементів залежить від кількості гравців (в прототипі від 2 до 4 гравців). Таким чином гру можна за бажанням досить легко масштабувати під різну кількість гравців. Нехай на одного гравця припадатиме 6 тайлів ігрового поля. Розподіл кількості тайлів зображено в таблиці 4.5.

Таким чином для прототипу доопрацьовано інші ігрові механіки, та створено технологічне забезпечення, у вигляді роздруківки тайлів, жетонів точок інтересу та інших ігрових елементів для зручного відтворення у форматі настільної гри та подальшого тестування ігрового процесу. Приклад вигляду ігрового поля зображено на рисунку 4.22. Технологічне забезпечення гри наведене в Додатку Д.

Таблиця 4.5 – Залежність кількості ігрових елементів від кількості гравців

Типи ігрових елементів	**Кількість гравців**		
	2 гравці	**3 гравці**	**4 гравці**
Тайли	13	19	25
Точки інтересу	23	33	43

Джерело: побудовано автором

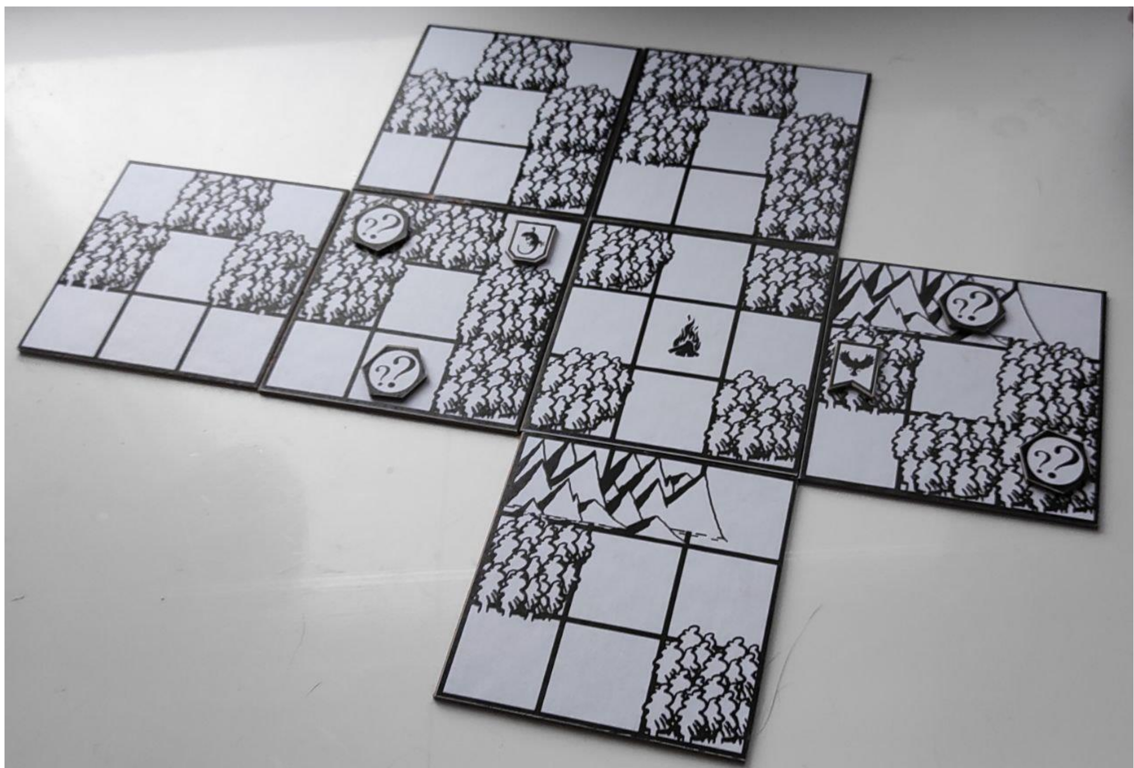


Рисунок 4.22 – Ігрове поле створеного прототипу

Джерело: побудовано автором

ВИСНОВКИ

У ході виконання кваліфікаційної роботи проведені дослідження джерел за тематикою процедурної генерації контенту та особливостей дизайну десктопних ігор. Виявлено, що необхідно створити інформаційну технологію генерації контенту, визначено об'єкт та предмет дослідження.

Після цього були окреслені задачі для виконання проекту. За результатами проведених досліджень обрано інструментарій створення програмного забезпечення та сформовані основні завдання для реалізації.

Виконано структурно функціональне моделювання, створено WBS та OBS діаграми, які дозволяють побудувати відповідну структуру робіт. Також створено календарний план, матрицю відповідальності і визначено ризики проекту які наведено в Додатку А.

Розроблено інформаційну технологію генерації контенту для десктопної гри, яка включає в себе аналіз використання алгоритмів Wave Function Collapse та псевдовипадкового алгоритму з подальшим використанням алгоритму аналізу візуальної та структурної подібності.

Проведено аналіз параметрів та способів побудови ігрового поля, в процесі якого було виведено формули розрахунку кількості варіантів його побудови в залежності від розмірності його елементів та методу їх з'єднання. На основі результатів використання створеної інформаційної технології обрано оптимальні параметри та особливості реалізації механіки побудови ігрового поля, які імплементовано в допрацьований власний прототип десктопної гри. Розроблено технологічне забезпечення створеного прототипу гри, наведене в Додатку Д.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Заруцька В., Жиленко М. Дидактичні настільні ігри як інструмент підвищення ефективності викладання у вищій школі. *Молодий вчений*. 2021. Т. 4, № 92. С. 46–49. URL: https://doi.org/10.32839/2304-5809/2021-4-92-10 (дата звернення: 30.09.2023).
2. Карпій О. П. Ринок настільних ігор україни: сучасний стан та перспективи розвитку. *Менеджмент та підприємництво в Україні: етапи становлення та проблеми розвитку*. 2022. Т. 2, № 8. С. 248–256.
3. Лях Т. Настільні ігри як інтерактивний метод просвітницько-профілактичної роботи. *Соціальна робота в Україні: теорія та практика.* 2009. №4. С. 37-45.
4. Abbott D. Modding tabletop games for education. *Games and Learning Alliance: 7th International Conference*, GALA 2018, Palermo, Italy, December 5–7, 2018, Proceedings 7. Springer International Publishing, 2019. P. 318–329
5. Bauer M. Using cellular automata to create a random map generator. *finalparsec.com*. URL: https://www.finalparsec.com/Blog/ViewPost/nauticus-map-generation (date of access: 30.11.2023).
6. Boris. Wave function collapse explained. *boristhebrave.com*. URL: https://www.boristhebrave.com/2020/04/13/wave-function-collapse-explained/ (date of access: 22.10.2023).
7. Bronson Z. Procedural generation with cellular automata. *bronsonzgeb.com*. URL: https://bronsonzgeb.com/index.php/2022/01/30/procedural-generation-with-cellular-automata/ (date of access: 22.10.2023).
8. Chopard, Bastien, Droz M. Cellular automata modeling of physical systems. Cambridge University Press, 2009. URL: https://doi.org/10.1017/CBO9780511549755 (date of access: 30.11.2023).

9. Crain H., Carpenter D., Martens C. Evaluating a Casual Procedural Generation Tool for Tabletop Role-Playing Game Maps. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. Roma, Italy, 2022. P. 1-6.
10. Depaulis T. Board games before ur?. *Board game studies journal*. 2020. Vol. 14, no. 1. P. 127–144.
11. Engelstein G., Shalev I. Building blocks of tabletop game design: an encyclopedia of mechanisms. CRC Press, 2022. 626 p.
12. Epix, Inc. Rogue. Pixel Games UK, 1985. URL: https://store.steampowered.com/app/1443430/Rogue/ (date of access: 30.11.2023).
13. Frank, Eric Smith D. Measurement invariance, entropy, and probability. *Entropy*. 2009. Vol. 12, no. 3. P. 289–303.
14. Gaina, Raluca. TAG: a tabletop games framework. 2020.
15. Ham E. Tabletop game design for video game designers. CRC Press, 2015. 348 p.
16. Heaton R. The wavefunction collapse algorithm explained very clearly. *robertheaton.com*. URL: https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/ (date of access: 24.10.2023).
17. Kari, Jarkko. Cellular automata. 2022.
18. Koster R. Theory of fun for game design. 2nd ed. O'Reilly Media, Inc., 2013. 279 p.
19. Martin Donald. Superpositions, sudoku, the wave function collapse algorithm, 2020. *youtube.com*. URL: https://www.youtube.com/watch?v=2SuvO4Gi7uY (date of access: 30.11.2023).
20. Matplotlib: visualization with python. *matplotlib.org*. URL: https://matplotlib.org/ (date of access: 30.11.2023).

21. Merrell P., Manocha D. "Model synthesis: a general procedural modeling algorithm." IEEE transactions on visualization and computer graphics, Chapel Hill, 2010, P. 715-728.
22. Merrell P. Model synthesis : doctoral dissertation. Chapel Hill, 2009. 171 p.
23. Merrell P. Model synthesis. *paulmerrell.com*. URL: https://paulmerrell.org/model-synthesis/ (date of access: 30.10.2023).
24. Moyersoen F. Saboteur. AMIGO Spiel. 2 p. (Preprint). URL: https://world-of-board-games.com.sg/docs/Saboteur-Amigo.pdf (date of access: 02.11.2023).
25. NumPy documentation. *numpy.org*. URL: https://numpy.org/doc/stable/index.html (date of access: 25.10.2023).
26. OpenCV modules. *docs.opencv.org*. URL: https://docs.opencv.org/4.x/index.html (date of access: 25.10.2023).
27. ORB (Oriented FAST and Rotated BRIEF). *opencv.org*. URL: https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html (date of access: 28.10.2023).
28. Paul Merrell Research. Model synthesis algorithm, 2021. *youtube.com*. URL: https://www.youtube.com/watch?v=A2ODauA1a0M&t=28s (date of access: 30.10.2023).
29. Pillow. *pypi.org*. URL: https://pypi.org/project/Pillow/ (date of access: 27.10.2023).
30. Puchal C. H. To create a game master: a decalogue for procedural generation of interactive stories : bachelors thesis. 2021. 176 p. URL: https://doi.org/10.13140/RG.2.2.15031.85922 (date of access: 29.09.2023).
31. Python in a nutshell / A. Martelli et al. O'Reilly Media, Inc., 2023. 738 p.
32. Random dungeon generator. *donjon.bin.sh*. URL: https://donjon.bin.sh/fantasy/dungeon/ (date of access: 30.11.2023).
33. RP Archive. City tiles are better than battlemats - and i'll show you why!, 2021. *youtube.com*.

URL: https://www.youtube.com/watch?v=K_vR2wQlzFo&t=87s (date of access: 30.11.2023).

34. Saboteur (AR/EN). *unwindboardgames.com*. URL: https://unwindboardgames.com/products/saboteur-1 (date of access: 29.11.2023).

35. Scikit-image's documentation. *scikit-image.org*. URL: https://scikit-image.org/docs/stable/ (date of access: 26.10.2023).

36. Smith G. An analog history of procedural content generation. *International conference on foundations of digital games*. Boston, 2015.

37. Structural similarity index. *scikit-image.org*. URL: https://scikit-image.org/docs/dev/auto_examples/transform/plot_ssim.html (date of access: 24.10.2023).

38. Tabletop games designed to promote computational thinking / F. Poole et al. *Computer science education*. 2021. Vol. 32, no. 4. P. 449–475. URL: https://doi.org/10.1080/08993408.2021.1947642 (date of access: 29.09.2023).

39. The complete guide to understand IDEF diagram. *edrawmax.com*. URL: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html (date of access: 05.12.2023).

40. Wrede K.-J. Carcassonne. Z-Man Games. 6 p. (Preprint). URL: https://images.zmangames.com/filer_public/d5/20/d5208d61-8583-478b-a06d-b49fc9cd7aaa/zm7810_carcassonne_rules.pdf (date of access: 28.11.2023).

41. Ultimate History Of Videogames. Beneath apple manor (1978) - first top-down free-roaming RPG, 2018. *youtube.com*. URL: https://www.youtube.com/watch?v=b-AHgmV5pbo (date of access: 30.11.2023).

42. Zhiguo H., Yipin Y. Texture Synthesis: generating arbitrarily large textures from image patches. P. 1–8.

ДОДАТОК А

А.1 Ідентифікація мети IT-проекту

Метою проекту є дослідження впливу процедурної генерації на варіативність генеруємого контенту, створення оптимального технологічного забезпечення настільної гри з процедурною генерацією.

Для більш точного проектування деталізуємо мету за допомогою SMART-методу (див. табл. А.1).

Таблиця А.1 – Деталізація мети проекту методом SMART

Specific (деталізована)	Розробити ігрові механіки прототипу настільної гри з використанням процедурної генерації контенту (PCG)
Measurable (вимірювана)	Отримати N показник варіативності ігрової механіки при різних варіантах імплементації
Achievable (досяжна)	Існує велика кількість онлайн ресурсів, літератури для аналізу предметної області, а також безкоштовні інструменти для проведення обчислень та моделювання, тому виконання завдання не матиме високої складності
Relevant (реалістична)	Дослідження впливу технології PCG на ігровий процес дозволить створити інструменти та методики для швидкого та більш дієвого дизайну ігрових механік та технологічного забезпечення настільних ігор
Time-framed (обмежена в часі)	З врахуванням пошуку необхідних матеріалів поставлені задачі можливо виконати впродовж кількох тижнів

А.2 Планування змісту структури робіт інформаційної системи

Для розподілу обсягів роботи на етапи була розроблена WBS (Work Breakdown Structure) структура проекту. На рисунку А.1 зображена WBS діаграма, яка має ієрархічну структуру, завдяки якій простіше зрозуміти процес виконання задач для реалізації проекту.

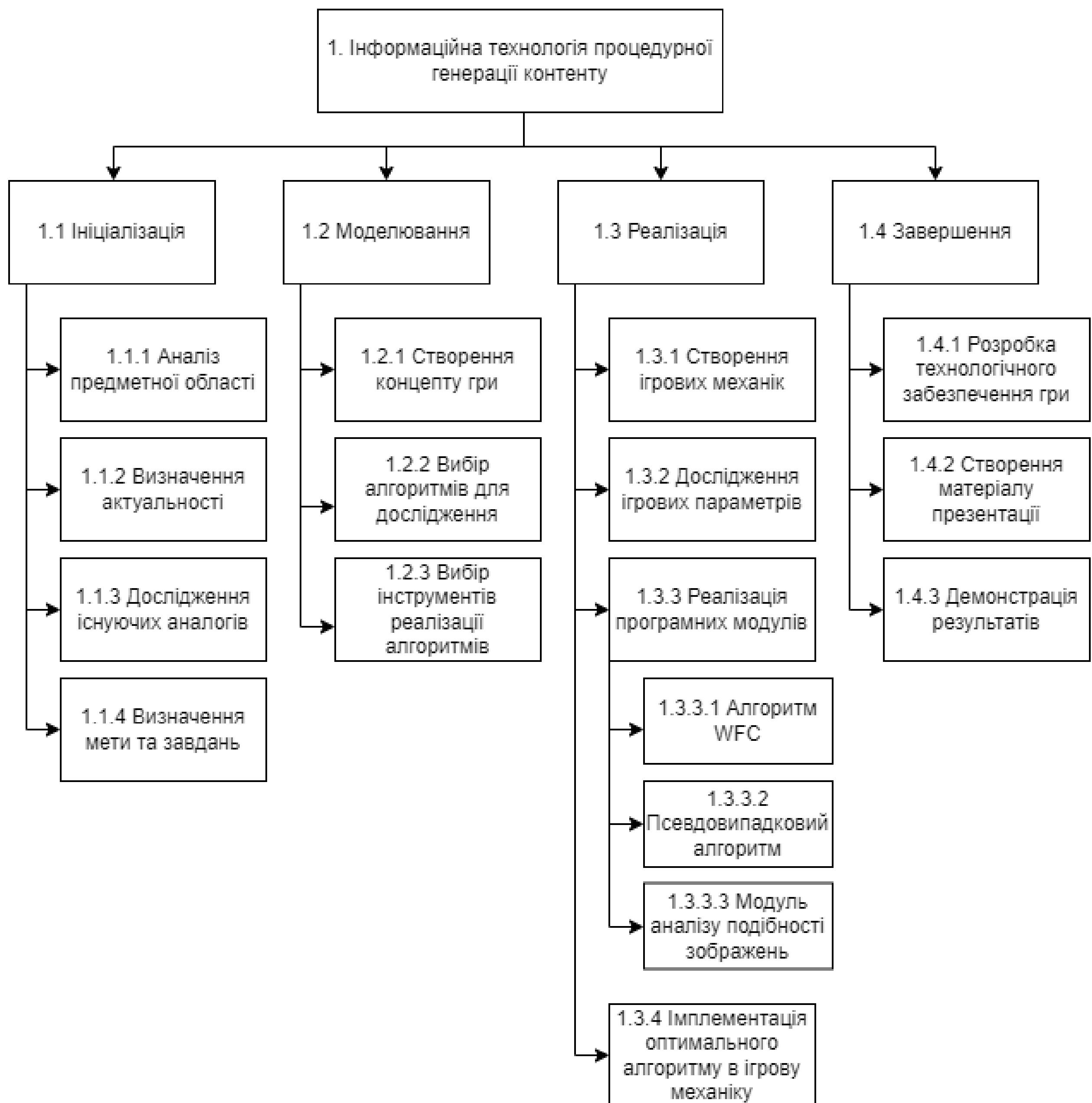


Рисунок А.1 – WBS структура проекту

Окрім WBS структури, необхідно створити OBS (Organization Breakdown Structure) структуру виконавців (див. рис. А.2).

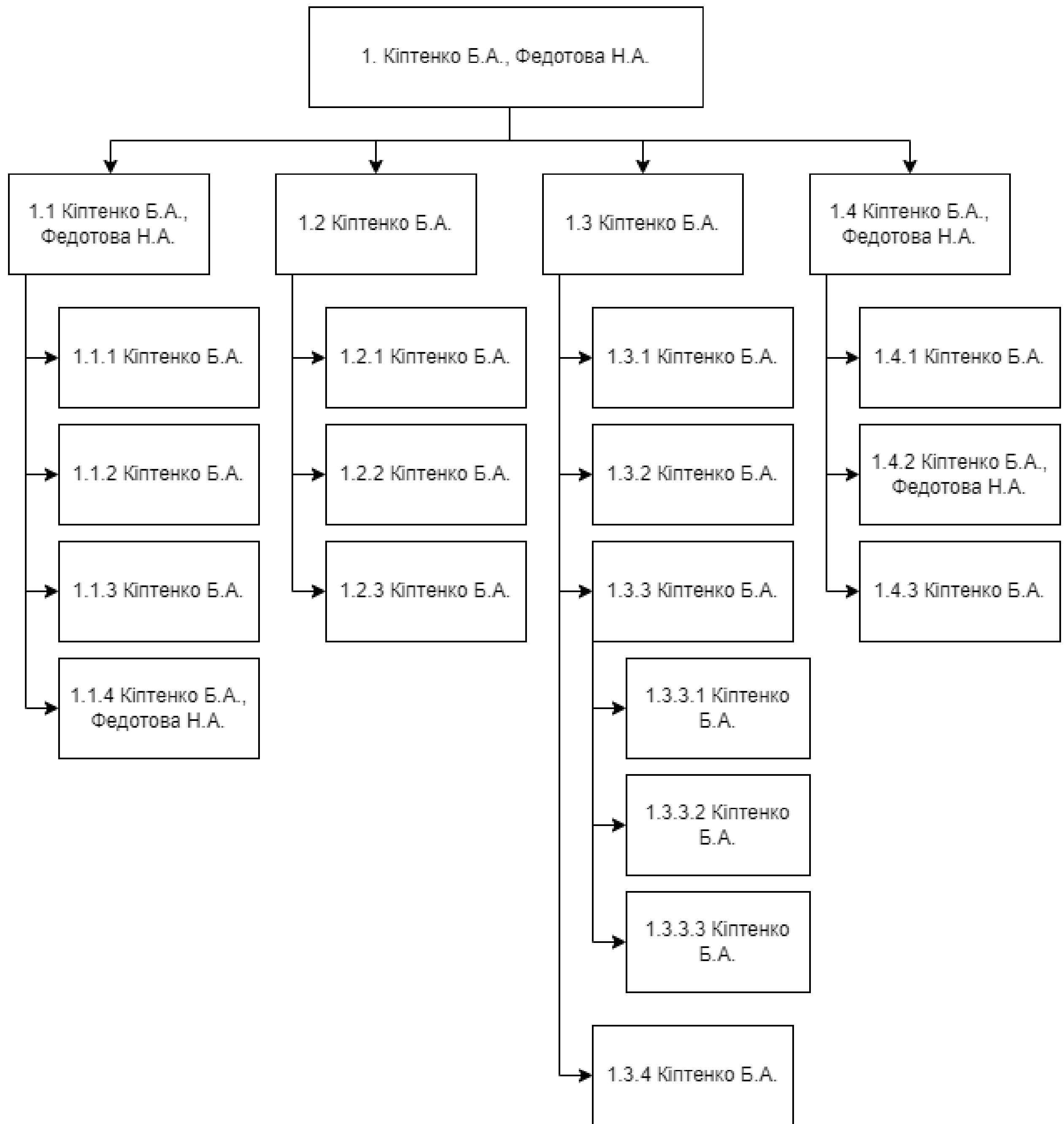


Рисунок А.2 – OBS структура проекту

А.3 Побудова календарного графіку виконання інформаційної системи

Для оцінки часу необхідного для виконання роботи було створено діаграму Ганта. Вона необхідна для відображення завдань та часу запланованого для їх виконання. Вона допомагає відстежувати дати початку та тривалості кожного етапу проекту. Побудована діаграма Ганта зображена на рисунку А.3.

А.4 Планування ризиків проекту

Для швидшого вирішення можливих проблем необхідно провести оцінку ризиків. Оцінку проведемо за наступними критеріями: ймовірність виникнення, вплив та тип ризику (див. табл. А.2).

В свою чергу кожен критерій включає в себе градацію по значущості:

Ймовірність виникнення:

1. Низька ймовірність
2. Середня ймовірність
3. Висока ймовірність

Ступінь впливу:

1. Низький
2. Середній
3. Високий

Рівень ризику:

1. Прийнятні
2. Виправдні
3. Недопустимі

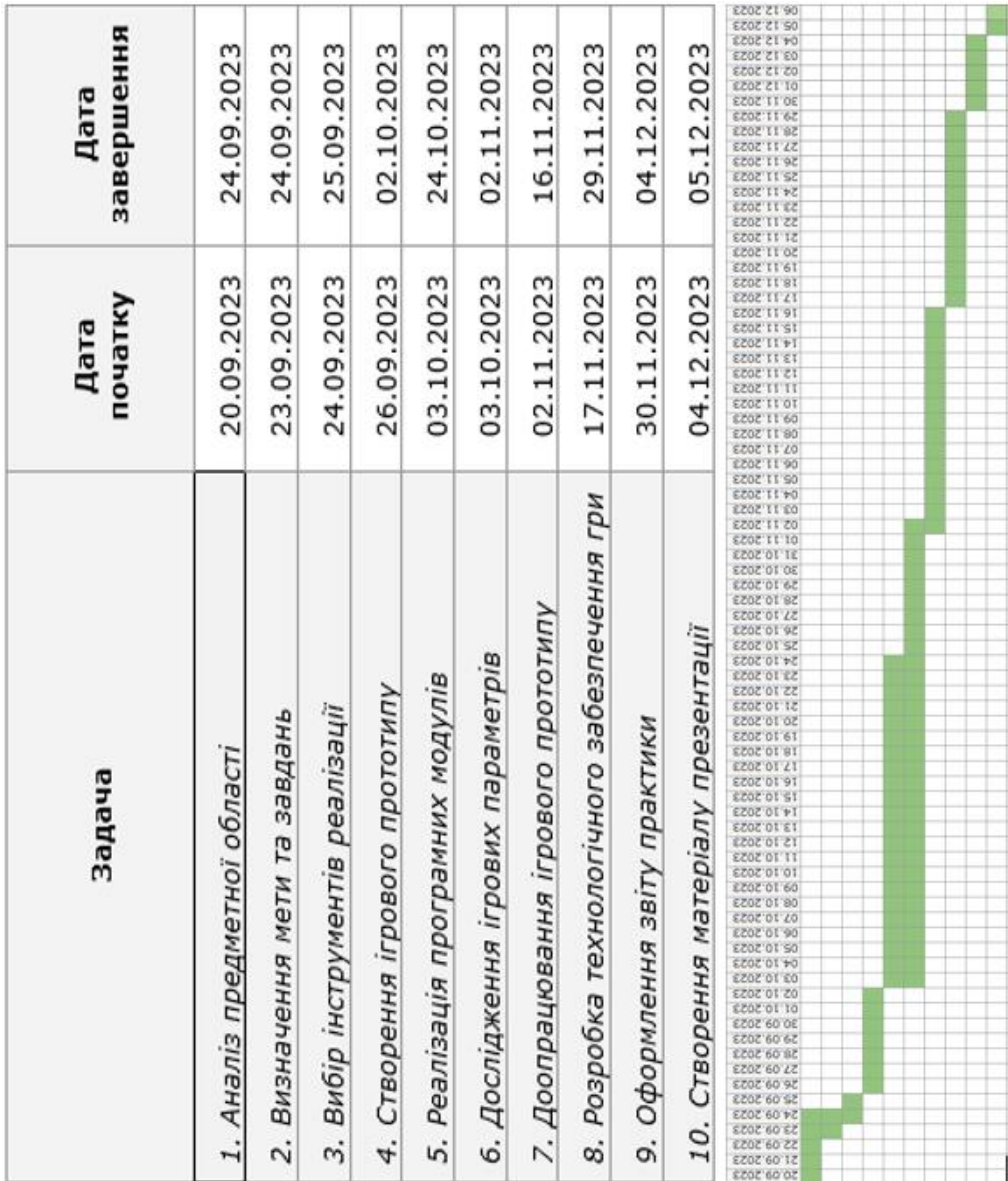
Таблиця А.2 – Шкала оцінювання ризиків за ймовірністю виникнення та величиною впливу

Оцінка	Ймовірність виникнення	Вплив ризику	Рівень ризику
	Низька	Низький	Прийнятні
	Середня	Середній	Виправдані
	Висока	Високий	Недопустимі

Визначивши шкалу оцінювання критеріїв, побудуємо матрицю відповідності для проведення оцінки можливих ризиків (див. табл. А.3).

Таблиця А.3 – Матриця оцінки ризиків

Ризик	Оцінка		
	Ймовірність виникнення	Вплив ризику	Тип ризику
Недоліки планування робіт			
Поява додаткових завдань/вимог			
Недотримання календарного плану			
Відсутність інтернет-зв'язку			
Відсутність електроенергії			
Технічні несправності			
Сбої роботи програмних засобів			



Задача	Дата початку	Дата завершення
1. Аналіз предметної області	20.09.2023	24.09.2023
2. Визначення мети та завдань	23.09.2023	24.09.2023
3. Вибір інструментів реалізації	24.09.2023	25.09.2023
4. Створення ігрового прототипу	26.09.2023	02.10.2023
5. Реалізація програмних модулів	03.10.2023	24.10.2023
6. Дослідження ігрових параметрів	03.10.2023	02.11.2023
7. Доопрацювання ігрового прототипу	02.11.2023	16.11.2023
8. Розробка технологічного забезпечення гри	17.11.2023	29.11.2023
9. Оформлення звіту практики	30.11.2023	04.12.2023
10. Створення матеріалу презентації	04.12.2023	05.12.2023

Рисунок А.3 – Діаграма Ганта проекту

ДОДАТОК Б
ПРОГРАМНИЙ МОДУЛЬ АЛГОРИТМУ WAVE FUNCTION COLLAPSE

Скрипт __init__.py

```
"""
A python implimentation of the Wave Function Collapse algorithm
"""
from collections import namedtuple


def run(x: int, y: int, pixel_size: tuple):
    """
    The main function for running the algorithm
    x           : The amount of tiles in the x direction
    y           : The amount of tiles in the y direction
    pixel_size  : amount of pixels for 1 row of the image, created using tuples
    tiles       : dictionary of tile samples
    """

    global tiles

    from random import shuffle

    assert tiles # Makes sure that tiles have been passed
    board = [ [None for j in range(x)] for i in range(y) ] # This is where all the values are held
    filled_set = set() # A Set with the (x, y) coordinates of every tile that exists

    def find_possible_neighbors(j, i) -> set:
        """
        Function for returning squares near i & j in appropriate range
        j : x value
        i : y value
        """
        outset = {
            (j, i - 1),
            (j + 1, i),
```

```
        (j, i + 1),
        (j - 1, i)
    }
    return { entry for entry in outset if (0 <= entry[0] < x) and (0 <= entry[1] < y) } # returns entries in
range


def find_entropy(j, i):
    """
    function for finding the entropy of a potential tile

    """

    # tile to top
    if (j, i - 1) in filled_set:
        ref = board[i - 1][j]
        if ref is not None:
            ref = ref.bottom[::-1]
            localset = set()
            for tile in tiles:
                if tiles[tile].top == ref: localset.add(tile)
    else:
        localset = set()
        for tile in tiles:
            localset.add(tile)
    outset = localset

    # tile to right
    if (j + 1, i) in filled_set:
        ref = board[i][j + 1]
        if ref is not None:
            ref = ref.left[::-1]
            localset = set()
            for tile in tiles:
                if tiles[tile].right == ref: localset.add(tile)
    else:
        localset = set()
        for tile in tiles:
            localset.add(tile)
```

```
        outset &= localset

        # tile to bottom
        if (j, i + 1) in filled_set:
            ref = board[i + 1][j]
            if ref is not None:
                ref = ref.top[::-1]
                localset = set()
                for tile in tiles:
                    if tiles[tile].bottom == ref: localset.add(tile)
        else:
            localset = set()
            for tile in tiles:
                localset.add(tile)
        outset &= localset

        # tile to left
        if (j - 1, i) in filled_set:
            ref = board[i][j - 1]
            if ref is not None:
                ref = ref.right[::-1]
                localset = set()
                for tile in tiles:
                    if tiles[tile].left == ref: localset.add(tile)
        else:
            localset = set()
            for tile in tiles:
                localset.add(tile)
        outset &= localset

        return (len(outset), (j, i), outset)

    def draw_image(filename: str = "image"):
        """
        Function for drawing the generated data to file
        """
        from PIL import Image
```

```
        data = []

        for i in board:
            for j in range(len(board[0][0].image)):
                data.append([])
                for k in i:
                    data[-1] = [*data[-1], *[l for l in k.image[j]]]

        im = Image.new("RGBA", pixel_size, (255, 255, 255, 255))
        data = [ [ data[int(len(data) * (i / pixel_size[1]))][int(len(data[1]) * (j / pixel_size[0]))] for j in range(pixel_size[0])] for i in range(pixel_size[1]) ]
        data2 = []
        for i in data:
            for j in i:
                data2.append(j)
        im.putdata(data2)
        im.save(f"{filename}.png")

    while x * y > len(filled_set):
        to_go_through = set() # A set of squares to go to
        for tile in filled_set: # Adds potential neighbors for each filled_set
            to_go_through |= find_possible_neighbors(*tile)
        to_go_through -= filled_set # Removes filled tiles from potential squares
        if len(to_go_through) == 0:
            to_go_through = {(0, 0)} # If it doesn't want to go through tiles, just go to the top left

        entropies = [ find_entropy(*tile) for tile in to_go_through ] # Gets entropies for each square to go through
        entropies.sort() # sorts by smallest entropy (least possibilities)
        entropies = [ tile for tile in entropies if tile[0] == entropies[0][0] ] # gets all the tiles with equivalent entropy values
        shuffle(entropies) # shuffles values
        entropies = entropies[0] # picks a random value
        xy = entropies[1] # sets xy coordinate of chosen entropy (set in `find_entropy()` function)
        entropies = list(entropies[2]) # sets the possible tiles that could go at that entropy value
        if len(entropies) == 0:
            return False # Returns if no possible square is found
        shuffle(entropies) # shuffles the tiles that could go to xy
```

```
        board[xy[1]][xy[0]] = tiles[entropies[0]] # Sets the board at the xy value to be a chosen tile
        filled_set.add(xy) # adds the added tile to the `filled_set` set

    draw_image() # Saves the generated image
    return True

def rotate_tile(tile, n: int, depth: int = 0):
    """
    Function for rotating a tile
    """


    if depth >= n:
        return tile


    (x, y) = (len(tile.image[0]), len(tile.image))


    adjs_args = [ tile[(i - 1) % 4] for i in range(4) ] # Gets a list of rotated edges


    color_data = zip(*tile.image) # Transposes List
    color_data = [ list(i[::-1]) for i in color_data ] # flips the data over the y axis


    return rotate_tile(tile = adjs(*adjs_args, color_data), n = n, depth = depth + 1)

def get_tiles(tiles_path: str = "img/wfc3f"):
    """
    Function that just returns the dictionary of tiles,  str = "path_to_images"
    """


    import os
    from glob import glob
    from PIL import Image


    basedir = os.path.abspath(".")
    os.chdir(tiles_path)


    tiles = {}
```

```
#map lookup table (5 samples | wfc3)
# file_lookup_table = {
#       "tile1.png": ("BAB", "BAB", "BAB", "BAB"),
#       "tile2.png": ("BAA", "BAB", "BBB", "BAB"),
#       "tile3.png": ("BBA", "AAB", "BAB", "BAB"),
#       "tile4.png": ("ABB", "BAB", "BAB", "BAA"),
#       "tile5.png": ("BAB", "BAC", "CAB", "BAB"),
# }

#map lookup table (10 samples | wfc3)
# file_lookup_table = {
#       "tile1.png": ("BAB", "BAB", "BAB", "BAB"),
#       "tile2.png": ("BAA", "BAB", "BBB", "BAB"),
#       "tile3.png": ("BBA", "AAB", "BAB", "BAB"),
#       "tile4.png": ("ABB", "BAB", "BAB", "BAA"),
#       "tile5.png": ("BAB", "BAC", "CAB", "BAB"),
#       "tile6.png": ("CCA", "ABB", "BBA", "ABC"),
#       "tile7.png": ("ACC", "CAB", "BAB", "BAA"),
#       "tile8.png": ("BBA", "ABB", "BAA", "ABB"),
#       "tile9.png": ("BAB", "BBA", "ABB", "BAB"),
#       "tile10.png": ("CBA", "ABA", "ABA", "ABC"),
# }

# #map lookup table (forest to forest, mountains to mountains | wfc3f)
# file_lookup_table = {
#       "tile0.png": ("AAA", "AAA", "AAA", "AAA"),
#       #"tile1.png": ("BAB", "BAB", "BAB", "BAB"), #tile with campfire
#       "tile2.png": ("BAA", "AAB", "BAB", "BAB"),
#       "tile3.png": ("BBA", "ABB", "BBA", "ABB"),
#       "tile4.png": ("BBA", "ABB", "BAA", "ABB"),
#       "tile5.png": ("BBA", "ABB", "BAA", "ABB"),
#       "tile6.png": ("ABA", "ABA", "ABA", "ABA"),
#       "tile7.png": ("ABA", "ABA", "AAB", "BAA"),
#       "tile8.png": ("ABA", "ABA", "AAA", "ABA"),
#       "tile9.png": ("ABA", "AAB", "BBA", "AAA"),
#       "tile10.png": ("BAB", "BBA", "ABB", "BAB"),
#       "tile11.png": ("BAB", "BBA", "ABB", "BAB"),
```

```
#	"tile12.png": ("ABA", "ABB", "BAB", "BAA"),
#	"tile13.png": ("ABA", "ABB", "BAB", "BAA"),
#	"tile14.png": ("ABA", "ABB", "BAA", "ABA"),
#	"tile15.png": ("ABA", "ABB", "BAA", "ABA"),
#	"tile16.png": ("BBA", "ABB", "BAA", "ABB"),
#	"tile17.png": ("BBA", "ABB", "BAA", "ABB"),
#	"tile18.png": ("CCA", "ABB", "BBA", "ABC"), #tile with mountains
#	"tile19.png": ("CCA", "ABB", "BBA", "ABC"), #tile with mountains
#	"tile20.png": ("ACC", "AAB", "BAA", "CBA"), #tile with mountains
#	#"tile21.png": ("CCA", "AAB", "BAA", "ABC"), #tile with mountains
#	#"tile22.png": ("CCC", "CAB", "BAA", "ABC"), #tile with mountains
#	#"tile23.png": ("CAB", "BBA", "ABA", "ABC"), #tile with mountains
#	#"tile24.png": ("CCC", "CAB", "BAA", "ACC"), #tile with mountains
#	#"tile25.png": ("CCC", "CAA", "AAA", "ACC"), #tile with mountains
# }

#map lookup table (only mountains together | wfc3f)
file_lookup_table = {
	#"tile0.png": ("AAA", "AAA", "AAA", "AAA"),
	#"tile1.png": ("AAA", "AAA", "AAA", "AAA"), #tile with campfire
	"tile2.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile3.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile4.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile5.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile6.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile7.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile8.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile9.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile10.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile11.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile12.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile13.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile14.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile15.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile16.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile17.png": ("AAA", "AAA", "AAA", "AAA"),
	"tile18.png": ("CCA", "AAA", "AAA", "AAC"), #tile with mountains
```

```
    "tile19.png": ("CCA", "AAA", "AAA", "AAC"), #tile with mountains
    "tile20.png": ("CCA", "AAA", "AAA", "AAC"), #tile with mountains
    "tile21.png": ("CCA", "AAA", "AAA", "AAC"), #tile with mountains
    "tile22.png": ("CCC", "CAA", "AAA", "AAC"), #tile with mountains
    "tile23.png": ("CAA", "AAA", "AAA", "AAC"), #tile with mountains
    "tile24.png": ("CCC", "CAA", "AAA", "ACC"), #tile with mountains
    "tile25.png": ("CCC", "CAA", "AAA", "ACC"), #tile with mountains
}

# #map lookup table (free placement | wfc3f)
# file_lookup_table = {
#   #"tile0.png": ("AAA", "AAA", "AAA", "AAA"),
#   #"tile1.png": ("AAA", "AAA", "AAA", "AAA"), #tile with campfire
#   "tile2.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile3.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile4.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile5.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile6.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile7.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile8.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile9.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile10.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile11.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile12.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile13.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile14.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile15.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile16.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile17.png": ("AAA", "AAA", "AAA", "AAA"),
#   "tile18.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile19.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile20.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile21.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile22.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile23.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile24.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
#   "tile25.png": ("AAA", "AAA", "AAA", "AAA"), #tile with mountains
```

```
        # }

        for filename in glob("*"):

            im = Image.open(filename).getdata()
            temp_image = []

            # Assumes tiles are squares and are all the same size
            im_size = len(im)
            im_size = (int(im_size ** .5), int(im_size ** .5))

            for y in range(im_size[1]):
                temp_image.append([])
                for x in range(im_size[0]):
                    try:
                        temp_image[-1].append(im[x + y * im_size[1]])
                    except:
                        print(len(im), x, y, im_size[1])

            if filename in file_lookup_table: sides = file_lookup_table[filename]
            else: sides = (0, 0, 0, 0)
            tiles[filename] = adjs(*sides, temp_image)

        os.chdir(basedir)

        return tiles

    def get_all_rotations(tiles):

        """
        Function for getting all possible rotations of each tile
        """
        tile_keys = [i for i in tiles]

        # Initialises the Tiles by getting all the rotations
        for i in tile_keys: # Doesn't use just tiles here because dict can't change size while being itterated on
            # Goes through each tile that exists
```

```
            if i not in {"tile0.png", "1.png"}:
                for j in range(1, 4):
                    # Goes through all 3 rotations (a rotation of 0 and 4 is the same so it needs to check 3 more)
                    new_tile = rotate_tile(tiles[i], j) # Gets a new potential tile to add
                    tiles[f"{i}##{j}"] = new_tile
    return tiles


def get_all_adjacencies(tiles):
    """
    Function that sets all the adjacencies of all the tiles in the `tiles` dictionary based on image data
    """
    new_tiles = {}

    for tile in tiles:
        sides = []
        sides.append(tiles[tile].image[0]) # top
        sides.append( [tiles[tile].image[i][-1] for i in range(len(tiles[tile].image))] ) # right
        sides.append(tiles[tile].image[-1]) # bottom
        sides.append( [tiles[tile].image[i][0] for i in range(len(tiles[tile].image))] ) # left
        sides = [ str(i) for i in sides ]
        new_tiles[tile] = adjs(*sides, tiles[tile].image)
    return new_tiles

# Setting up Tiles
adjs = namedtuple("Adjacencies", "top right bottom left image")
tiles = get_tiles()
tiles = get_all_rotations(tiles)
# tiles = get_all_adjacencies(tiles)
```

ДОДАТОК В
ПРОГРАМНИЙ МОДУЛЬ АЛГОРИТМУ WAVE FUNCTION COLLAPSE

Скрипт run.py

```
"""
A file for running WFC algorithm
"""
from WFC import run


def forever_for():
	i = 0
	while True:
		yield i
		i += 1


xy = 5				# The amount of tiles in the x and y directions
image_dimensions = 1024		# The dimensions of the image generated


for i in forever_for():
	if run(x = xy, y = xy, pixel_size = (image_dimensions, image_dimensions)): break
	print(f" - No Solution in Attempt #{i}"); i+=1 #message for if the image failed to generate


input("\n ~ FINISHED")
```

ДОДАТОК Г
ПРОГРАМНИЙ МОДУЛЬ ПСЕВДОВИПАДКОВОГО АЛГОРИТМУ

Скрипт pseudorandom.py

```
from PIL import Image
from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
import cv2
import sys

"""
    Function for generating an image using a pseudorandom algorithm
"""
def generate_image(width, height):
    list_img_path = Path("img/wfc3f").glob("*.png")
    list_img = [str(p) for p in list_img_path]
    # list_img =
['img/wfc3f/tile1.png','img/wfc3f/tile2.png','img/wfc3f/tile3.png','img/wfc3f/tile4.png','img/wfc3f/tile5.png',
    #
'img/wfc3f/tile6.png','img/wfc3f/tile7.png','img/wfc3f/tile8.png','img/wfc3f/tile9.png','img/wfc3f/tile10.png',
    #
'img/wfc3f/tile11.png','img/wfc3f/tile12.png','img/wfc3f/tile13.png','img/wfc3f/tile14.png','img/wfc3f/tile15.png',
    #
'img/wfc3f/tile16.png','img/wfc3f/tile17.png','img/wfc3f/tile18.png','img/wfc3f/tile19.png','img/wfc3f/tile20.png',
    #
'img/wfc3f/tile21.png','img/wfc3f/tile22.png','img/wfc3f/tile23.png','img/wfc3f/tile24.png','img/wfc3f/tile25.png']
    new_images = np.random.shuffle(list_img)
    sliced = np.array_split(list_img, 5)
    row1 = sliced[0]
    row2 = sliced[1]
    row3 = sliced[2]
    row4 = sliced[3]
    row5 = sliced[4]
    #print(row5)
    img1 = [Image.open(x) for x in row1]
    img2 = [Image.open(x) for x in row2]
```

```
img3 = [Image.open(x) for x in row3]
img4 = [Image.open(x) for x in row4]
img5 = [Image.open(x) for x in row5]
widths, heights = zip(*(i.size for i in img1))
total_width = sum(widths)
max_height = max(heights)

row1_img = Image.new('RGB', (total_width, max_height))
row2_img = Image.new('RGB', (total_width, max_height))
row3_img = Image.new('RGB', (total_width, max_height))
row4_img = Image.new('RGB', (total_width, max_height))
row5_img = Image.new('RGB', (total_width, max_height))

x_offset = 0
for im in img1:
  row1_img.paste(im, (x_offset,0))
  x_offset += im.size[0]
x_offset = 0
for im in img2:
  row2_img.paste(im, (x_offset,0))
  x_offset += im.size[0]
x_offset = 0
for im in img3:
  row3_img.paste(im, (x_offset,0))
  x_offset += im.size[0]
x_offset = 0
for im in img4:
  row4_img.paste(im, (x_offset,0))
  x_offset += im.size[0]
x_offset = 0
for im in img5:
  row5_img.paste(im, (x_offset,0))
  x_offset += im.size[0]

arrImg1 = np.array(row1_img)
arrImg2 = np.array(row2_img)
arrImg3 = np.array(row3_img)
```

```
    arrImg4 = np.array(row4_img)
    arrImg5 = np.array(row5_img)
    #print(row1_img)
    #print(row2_img)
    cvImg1 = cv2.cvtColor(arrImg1, cv2.COLOR_RGB2BGR)
    cvImg2 = cv2.cvtColor(arrImg2, cv2.COLOR_RGB2BGR)
    cvImg3 = cv2.cvtColor(arrImg3, cv2.COLOR_RGB2BGR)
    cvImg4 = cv2.cvtColor(arrImg4, cv2.COLOR_RGB2BGR)
    cvImg5 = cv2.cvtColor(arrImg5, cv2.COLOR_RGB2BGR)

    final_img = cv2.vconcat([cvImg1, cvImg2, cvImg3, cvImg4, cvImg5])
    final_img = cv2.resize(final_img, (width, height))
    return final_img

#name of a saved image
img_name = "map4.png"
#generate image of set size
image = generate_image(2048, 2048)
plt.imshow(image)
plt.show()
#save image as a .png file
cv2.imwrite(img_name, image)
```

ДОДАТОК Д
ПРОГРАМНИЙ МОДУЛЬ АНАЛІЗУ ПОДІБНОСТІ ЗОБРАЖЕНЬ

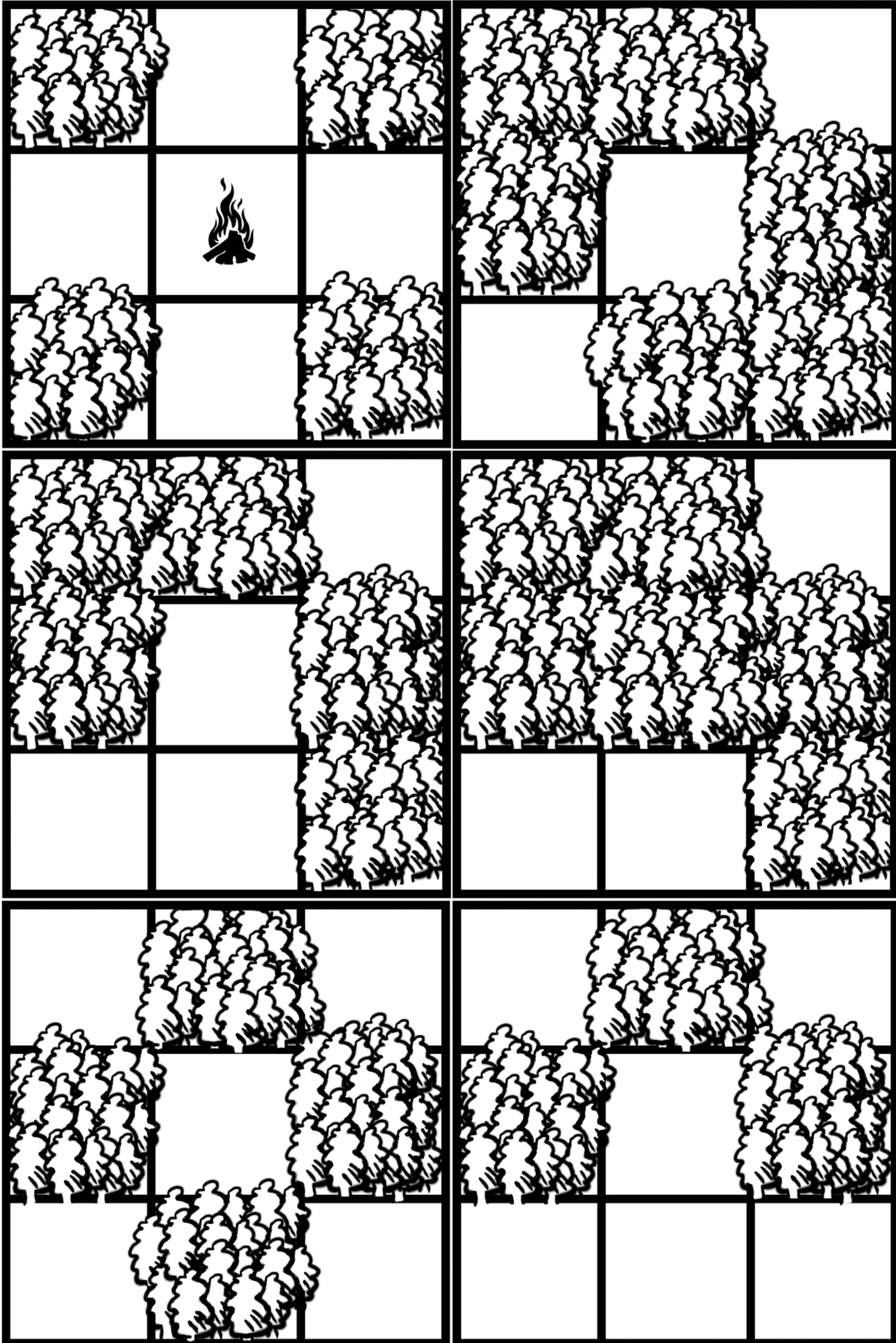
Скрипт img_compare.py

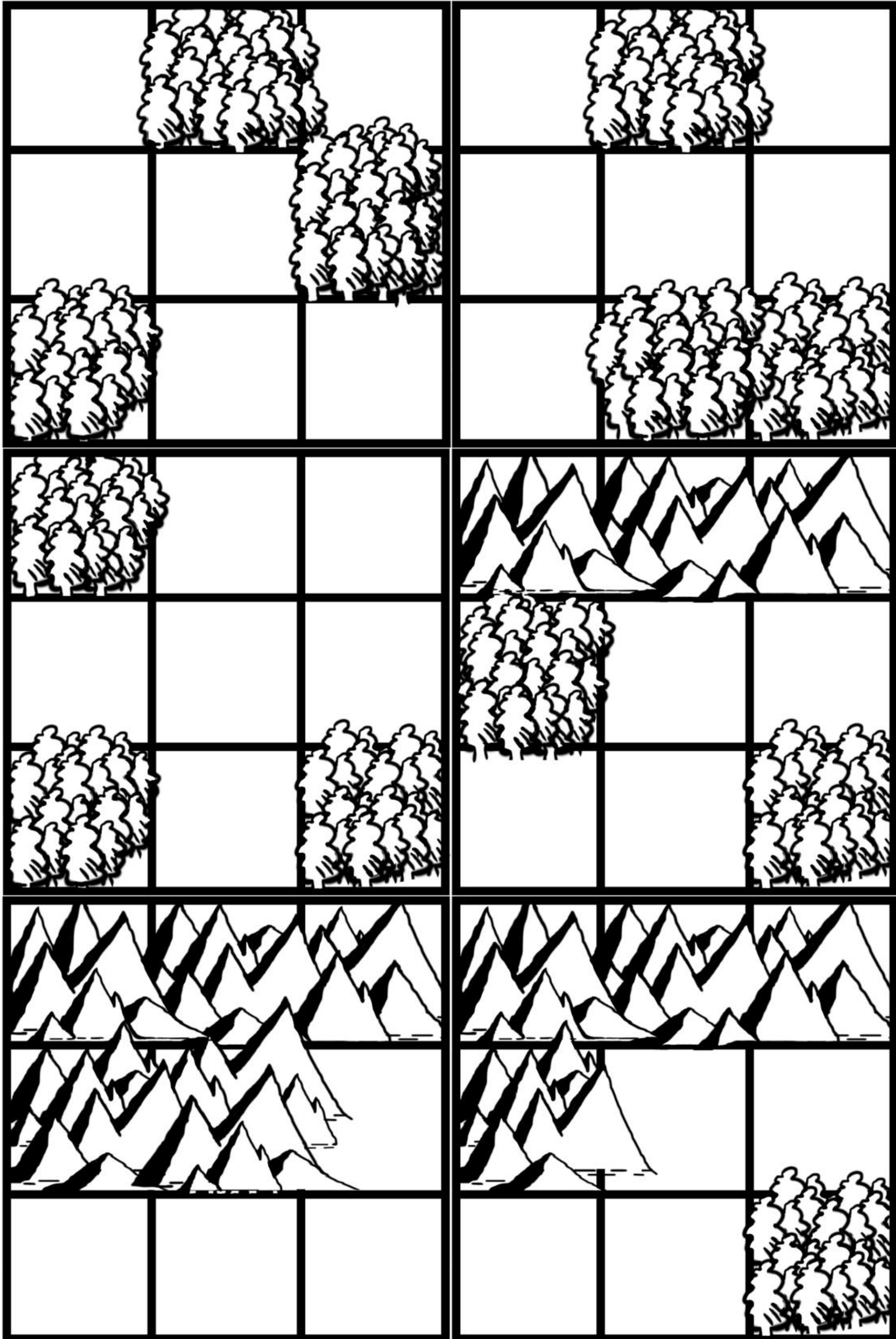
```
from skimage.metrics import structural_similarity
import cv2
"""
  Function for evaluating visual features similarity with ORB
"""
def orb_sim(img1, img2):
  orb = cv2.ORB_create()

  # detect keypoints and descriptors
  kp_a, desc_a = orb.detectAndCompute(img1, None)
  kp_b, desc_b = orb.detectAndCompute(img2, None)

  # define the bruteforce matcher object
  bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
  bf_knn = cv2.BFMatcher(cv2.NORM_HAMMING)
  #perform matches
  matches_knn = bf_knn.knnMatch(desc_a, desc_b, k=2)
  matches = bf.match(desc_a, desc_b)

  good = []
  for m,n in matches_knn:
    if m.distance < 0.75*n.distance:
      good.append([m])

  final_image = cv2.drawMatchesKnn(img1, kp_a, img2, kp_b, good, None)
  final_image = cv2.resize(final_image, (2048, 1024))
  plt.imshow(final_image),plt.show()

  #look for similar regions with distance < 50. Goes from 0 to 100 so pick a number between.
  similar_regions = [i for i in matches if i.distance < 50]
  if len(matches) == 0:
```

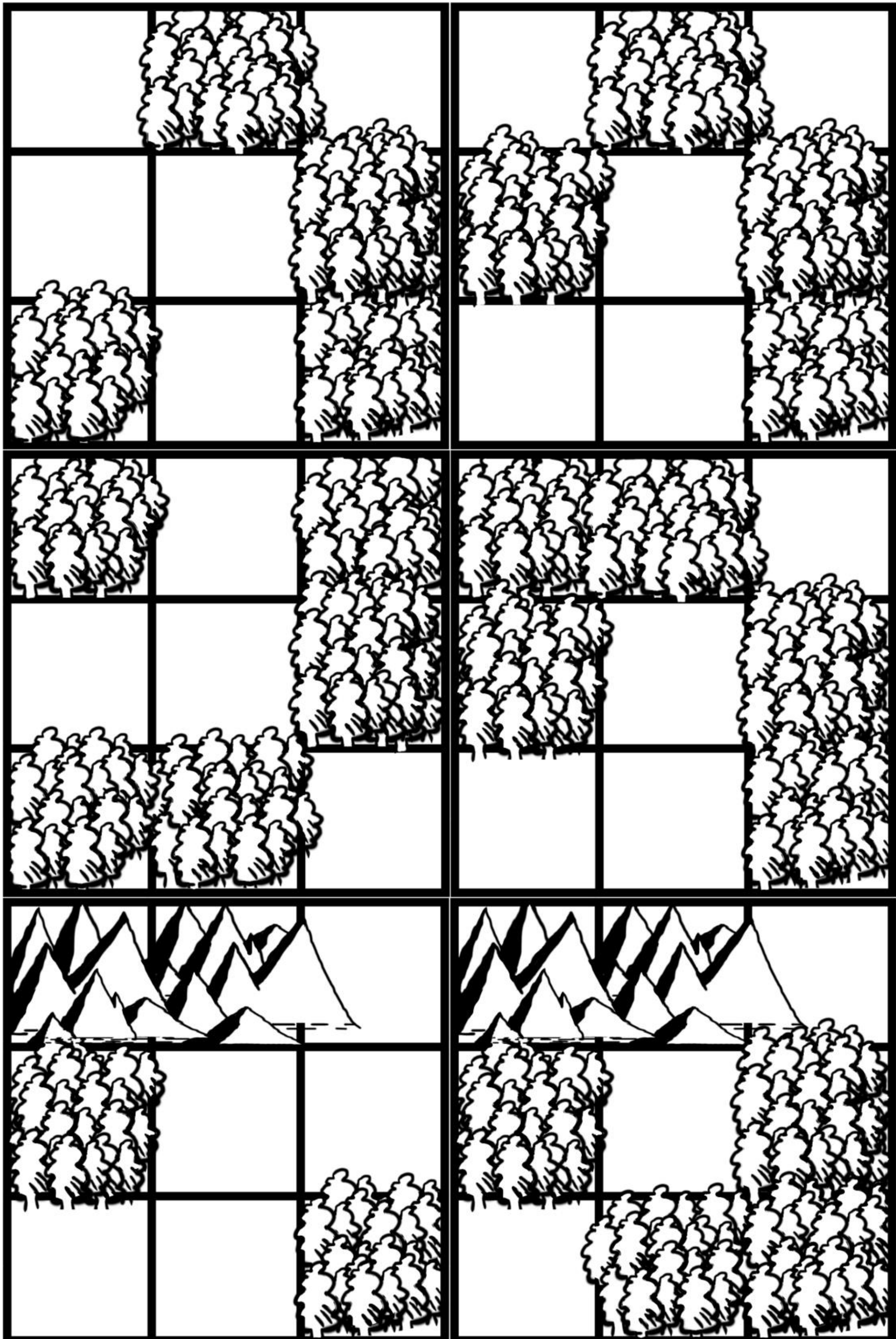
```
    return 0
  return len(similar_regions) / len(matches)


"""
    Function for evaluating structural similarity with SSIM
"""
def structural_sim(img1, img2):


  sim, diff = structural_similarity(img1, img2, full=True)
  return sim


"""
    Target images
"""
# #input image names for pseudorandom generated maps
# img_name1 = 'map3.png'
# img_name2 = 'map4.png'

# #input image names for WFC generated maps
# img_name1 = 'map_wfc1.png'
# img_name2 = 'map_wfc2.png'

# #input image names for both WFC and pseudorandom maps
# img_name1 = 'map_wfc2.png'
# img_name2 = 'map4.png'

#input image names for both WFC and pseudorandom maps
img_name1 = 'map_wfc_f1.png'
img_name2 = 'map_wfc1.png'

img00 = cv2.imread(img_name1, 0)
img01 = cv2.imread(img_name2, 0)

img00_plot = cv2.imread(img_name1)
img01_plot = cv2.imread(img_name1)
img00_plot = cv2.resize(img00, (2048, 2048))
img01_plot = cv2.resize(img00, (2048, 2048))
```

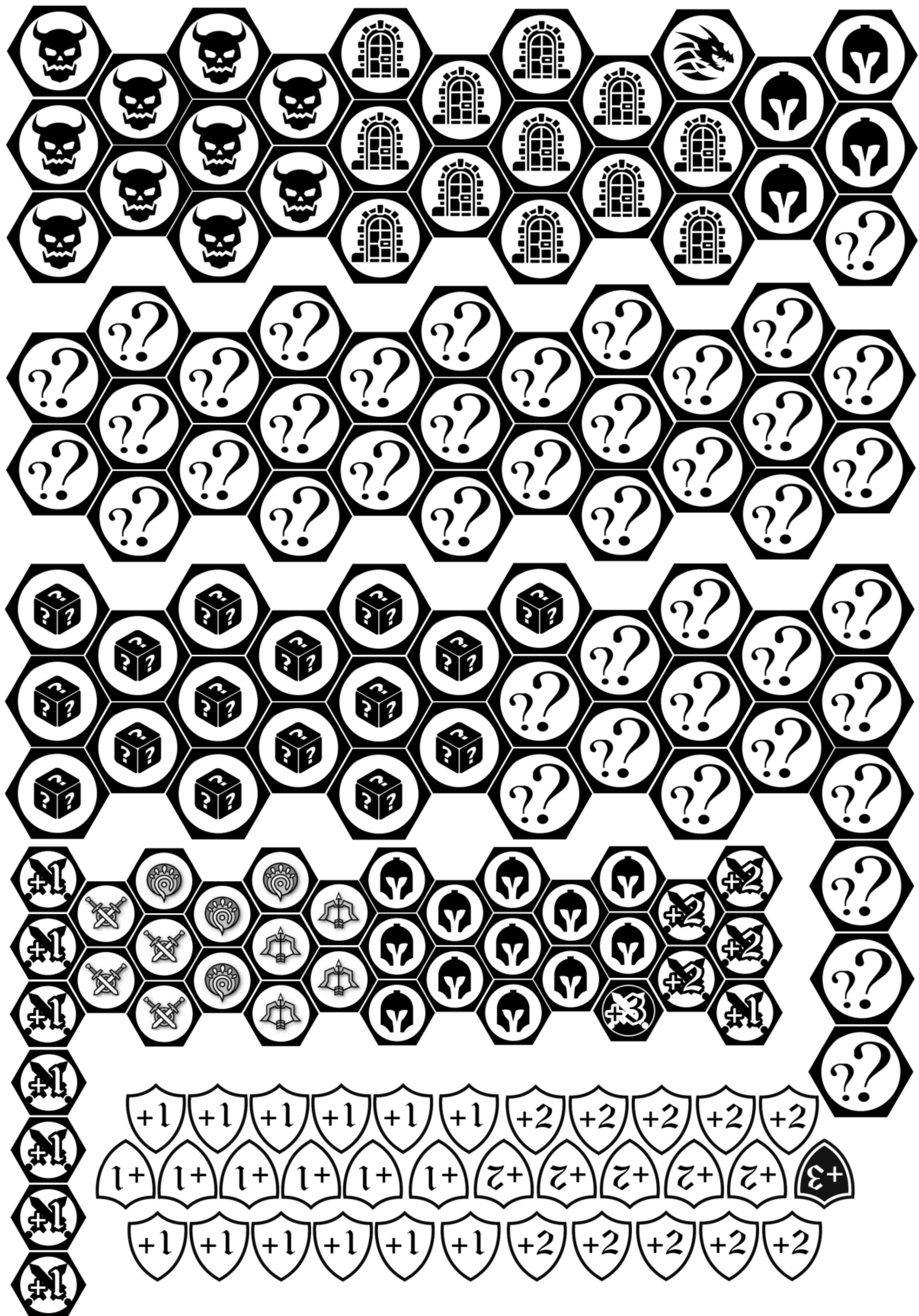
```
img00 = cv2.resize(img00, (2048, 2048))
img01 = cv2.resize(img01, (2048, 2048))

orb_similarity = orb_sim(img00, img01)  #1.0 means identical. Lower = not similar
print("Similarity using ORB is: ", orb_similarity)

ssim = structural_sim(img00, img01) #shows percentage of structural similarity
print("Similarity using SSIM is: ", ssim)
```

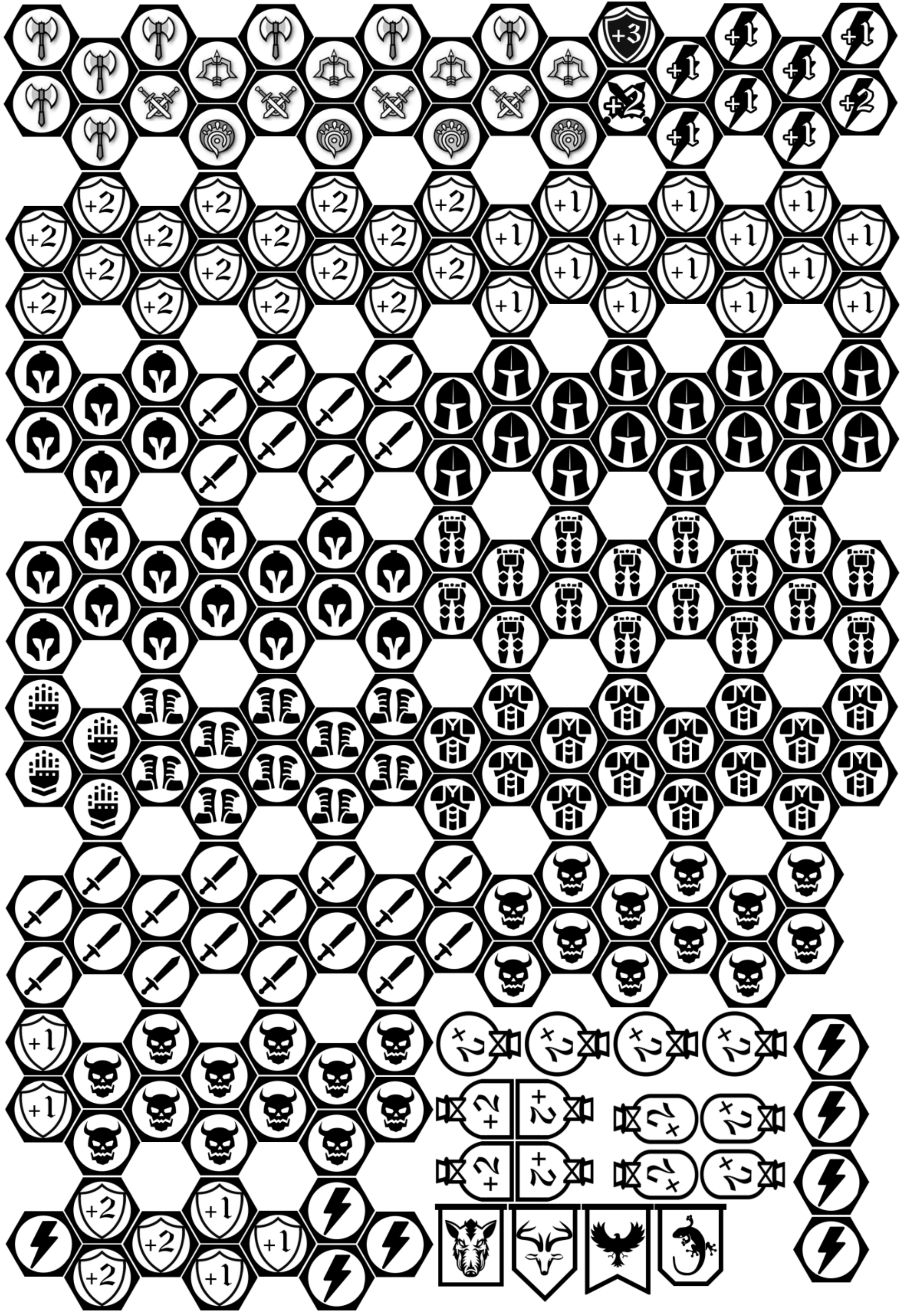
ДОДАТОК Е
ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОТОТИПУ ГРИ

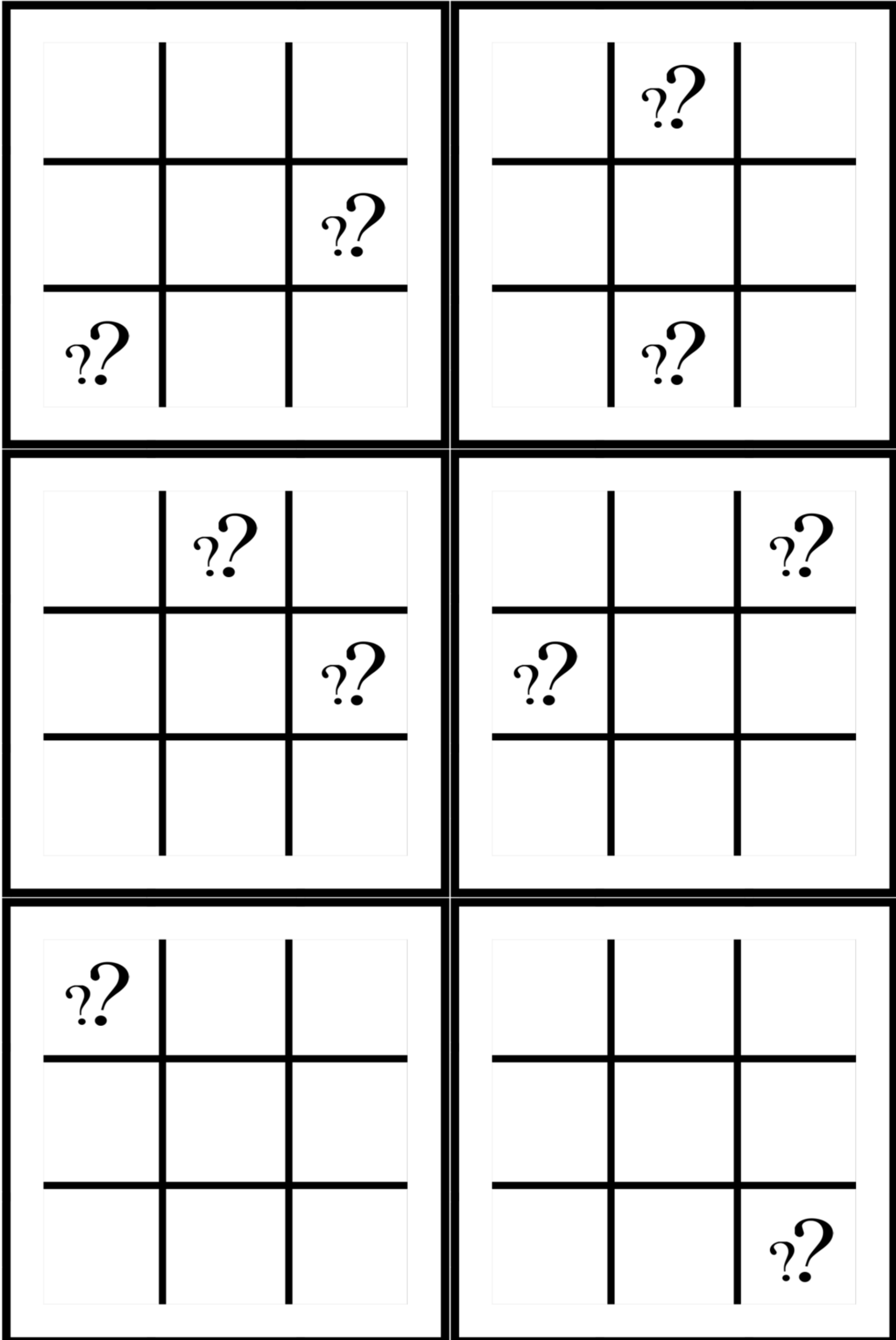


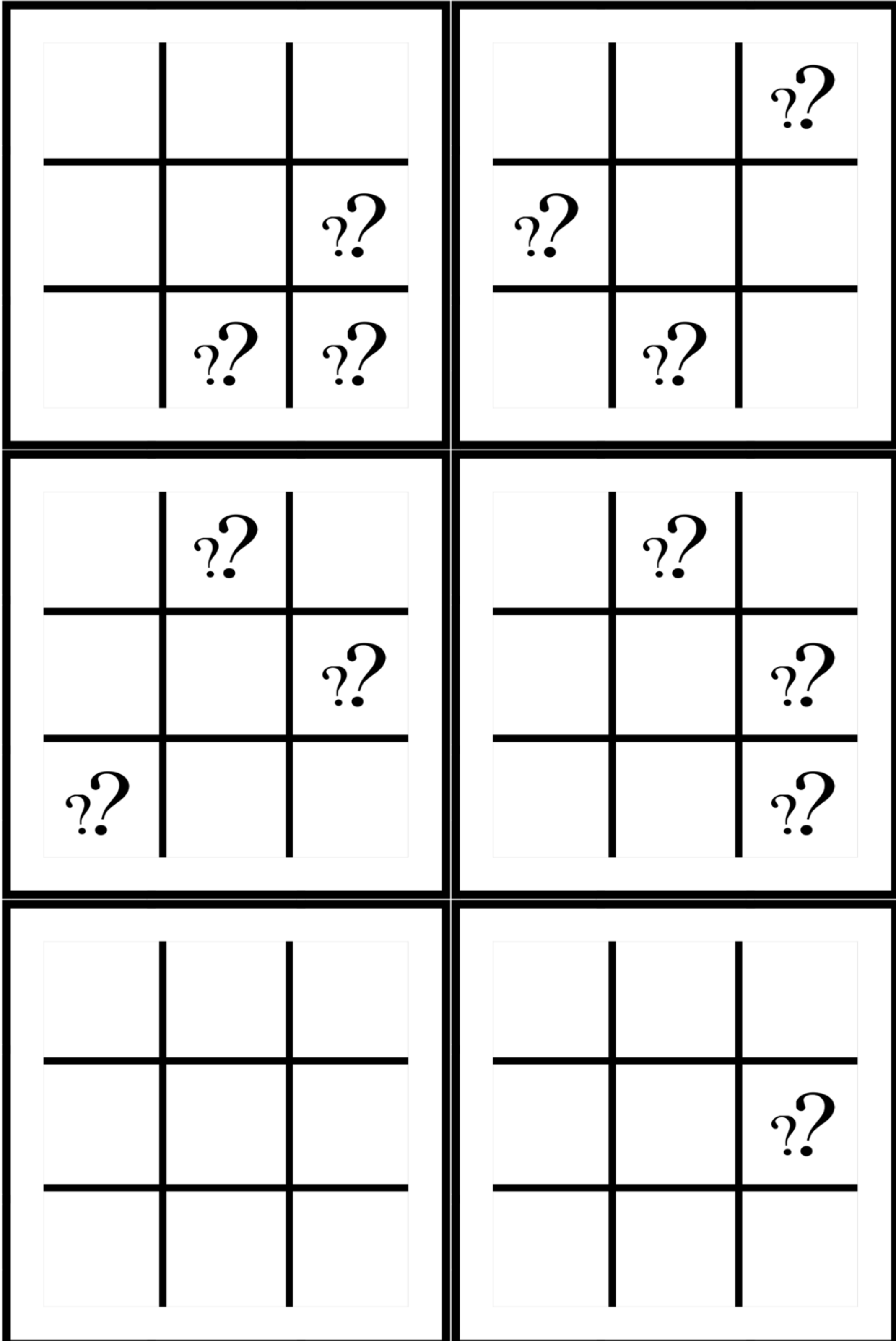


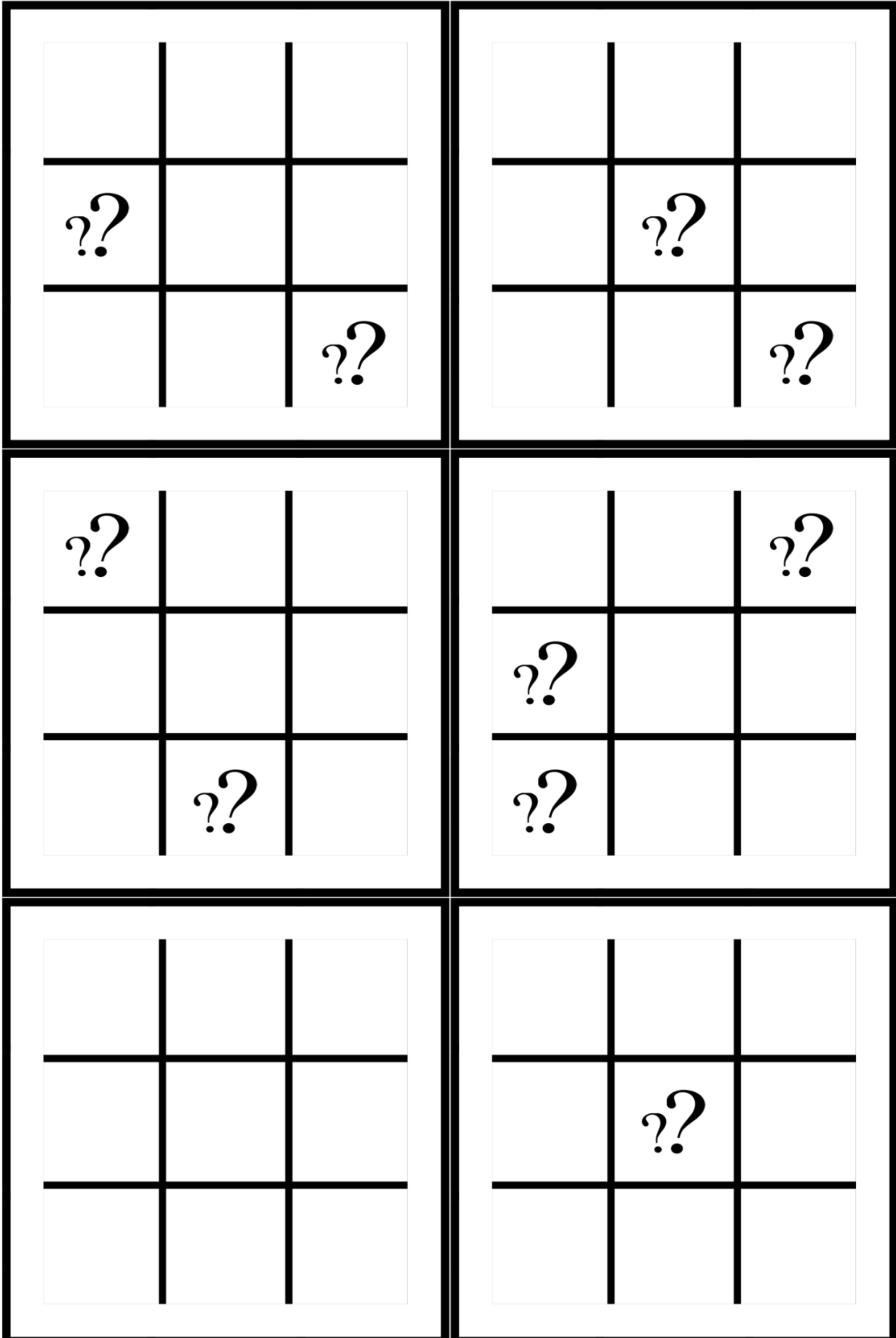


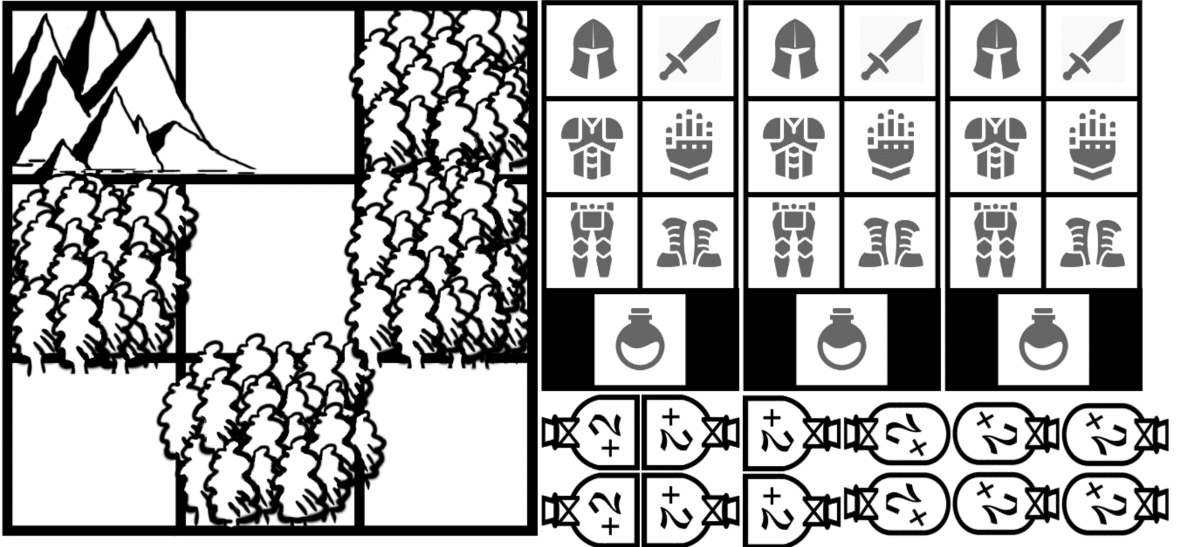










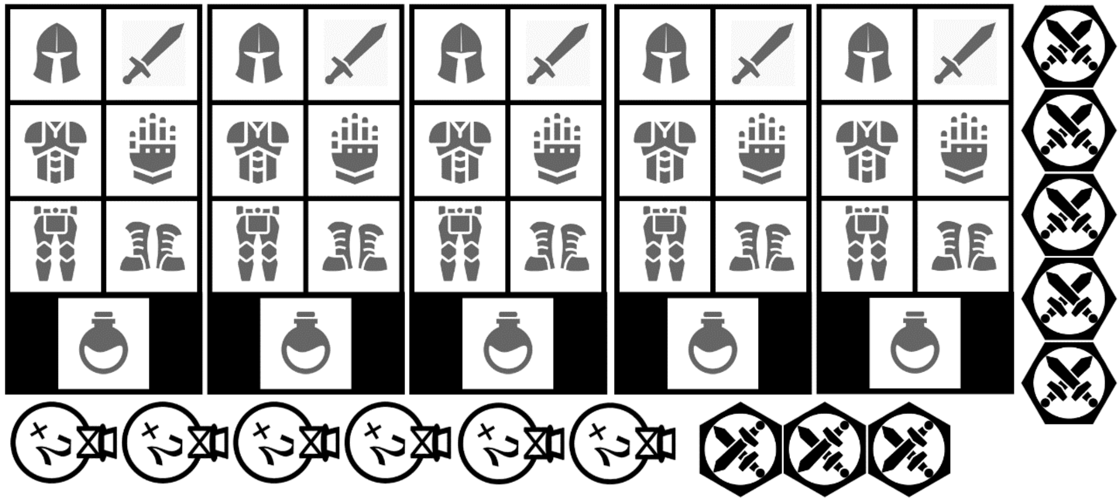


0 1 2 3 4 5 6 7 8 9 10 11 12 13

0 1 2 3 4 5 6 7 8 9 10 11 12 13

0 1 2 3 4 5 6 7 8 9 10 11 12 13

0 1 2 3 4 5 6 7 8 9 10 11 12 13



ДОДАТОК Є
ІГРОВІ ПРАВИЛА

Мета гри

Першим знайти та здолати дракона

Початок гри

У грі може приймати участь від 2 до 4 гравців. Кожен гравець обирає «герб» своєї гільдії, який позначатиме його положення на ігровому полі. Оберіть порядок ходу зручним для вас способом (кидок грального кубика, камінь-ножиці-папір, тощо).

Виставіть початковий регіон (тайл) з позначкою "багаття". Саме звідси буде починатися хід гравців.

Візьміть Лічильник (Counter), значок обраного героя, Лічильники для спорядження та показників персонажа.

Підготуйте окремо регіони (тайли місцевості), картки розвідки та перемішайте їх.

Кожен гравець починає гру маючи 1 героя. Кожен герой має за *замовчуванням 2 міці та 10 очок витривалості (ОВ)*. Дані показники використовуються при зустрічі з *монстрами* та відображають його можливість їх здолати (монстри також мають показники *міці* та *витривалості*).

Послідовність ходу

У свій хід гравець може здійснити наступні дії:

1. **Пересування ігровим полем**

   На початку ходу кожен гравець починає з 5 очками енергії (ОЕ) Енергія витрачається на пересування по ігровому полю. Поле має різні типи місцевості, пересування якими витрачає різну кількість енергії (Рівнини – 1 ОЕ / Ліс – 2 ОЕ / Гори – 3 ОЕ).

Гравець вибирає напрямок відносно свого положення і витрачає енергію згідно місцевості, на якій він знаходився в момент здійснення ходу. При пересуванні по діагоналі витрачається вдвічі більше ОЕ .

При виході за межі існуючого регіону (тайлу), візьміть з колоди новий регіон (тайл) та покладіть як вам заманеться відповідно до напрямку вашого руху. Після чого візьміть з колоди розвідки *картку розвідки* (КР) і покладіть випадкові жетони «точок інтересу» на тайл зображенням «знаку питання» догори на місця вказані на картці.

2. Взаємодія з «точками інтересу»

Коли гравець пересувається на сектор з «точкою інтересу» він може витратити 1 ОЕ для взаємодії з нею. «Точки інтересу» можуть містити одну з наступних подій:

Зустріч з монстром

Коли гравець розкриває жетон (токен) *монстра,* він негайно повинен вступити з ним у взаємодію. Монстр обирається зі списку шляхом кидка грального кубика к12. Від нього можна втекти, у випадку втечі, ваш герой втрачає *очки витравалості* рівні показнику *міці* монстра.

Випадкова подія

Коли гравець розкриває жетон (токен) *події*, він кидає кубик к100, щоб визначити з чим саме йому доведеться зустрітися. В події може знаходитись все що завгодно, згідно таблиці подій: пастки, монстри, скарби, новий герой, тощо.

Підземелля

Коли гравець розкриває жетон (токен) *підземелля*, він тягне Картку Підземелля, на якій вказана кількість кімнат, можливі події в кімнатах та нагорода за проходження підземелля. Гравець вирішує починати проходження підземелля чи ні. При проходженні підземелля гравець витрачає 1 *енергію* для відкриття кімнати, після чого кидає кубик (к6) для визначення події що очікує його в наступній кімнаті. Можливі події та значення кубика що їх визначає вказані на Картці Підземелля. Гравець не

може піти у відпочинок під час проходження підземелля. Гравець може завершити хід знаходячись в підземеллі, прогрес його проходження при цьому не зникає. Інші гравці не можуть зайти в підземелля, якщо в ньому знаходиться інший гравець.

Герой

Коли гравець розкриває жетон (токен) *героя,* він додає нового героя до своєї групи.

Дракон

Коли гравець розкриває жетон (токен) *дракона,* він робить вибір взаємодіяти з ним чи ні. Взаємодія відбувається за тими ж правилами що і *Зустріч з монстром.* Гравець що першим здолає *дракона* стає переможцем.

3. Відпочинок

Гравець витрачає цілий хід, для повного відновлення *витривалості* всіх *героїв*, додатково отримує + 1 ОЕ на наступний хід. Відпочинок гравець може здійснити ЛИШЕ НА ПОЧАТКУ ХОДУ до виконання будь якої іншої дії.

У випадку невдалої зустрічі з монстром, гравець автоматично здійснює відпочинок і пропускає наступний хід.

Спорядження

При проходженні *Підземелля* чи завершенні певних подій гравець може отримати *Скарби*, які можуть включати в себе *спорядження* та *припаси*. *Спорядження* дає постійний бонус до *міці* або *витривалості* героя, *припаси* можуть дати тимчасовий ефект при використанні (відновити *витривалість*, *енергію*, збільшити *міць* на наступний раунд, тощо). Кожен герой може мати одночасно 1 спорядження кожного типу. Слідкувати за спорядженням можна використовуючи Лічильники для спорядження та відповідні жетони (токени).

Взаємодія з монстром

Взаємодія з монстром відбувається за принципом раундів. Кожного раунду у обраного героя гравця та монстра віднімається *витравалість* відповідно до показника *міці* супротивника. Якщо витривалість монстра падає до 0, гравець перемагає, та може продовжувати хід. Якщо витривалість героя гравця падає до 0, він може обрати наступного, якщо вони в нього є, інакше повинен завершити свій хід та здійснити відпочинок. Перед початком кожного раунду гравець може використати 1 *припас* для будь якого свого героя.

Формула розрахунку показнику міці гравця

Базова *міць* всіх героїв у групі + Сумарна *міць* спорядження всіх героїв гравця + додаткові ефекти

Приклад

Якщо герой гравця має сумарно 3 *міці* та 12 *витривалості*, а монстр має 2 *міці* на 7 *витривалості*, то після першого раунду герой гравця матиме 12-2=10 *витривалості*, а монстр 7-3=4 *витривалості*. Гравець переможе після 3 раундів, таким чином у його героя залишиться 12-2-2-2=6 *витривалості*.

ТАБЛИЦІ РОЗРАХУНКУ ІГРОВИХ ЕЛЕМЕНТІВ

Таблиця Є.1.1 – Розподіл Карток Розвідки

Кількість подій	Кількість гравців		
	2 гравці	3 гравці	4 гравці
Події х3	4	5	6
Події х2	4	7	10
Події х1	3	4	5
Події х0	1	2	3
Сума	12	18	24

Таблиця Є.1.2 – Розподіл жетонів Точок інтересу

Тип події	Кількість гравців		
	2 гравці	3 гравці	4 гравці
Дракон	1	1	1
Герої	2	3	4
Підземелля	6	9	12
Події	8	12	16
Монстри	6	8	10
Сума	23	33	43

Таблиця Є.1.3 – Розподіл Скарбів

Тип предмету	Кількість гравців		
	2 гравці	3 гравці	4 гравці
Спорядження	22	33	44
Припаси	10	15	20
Артефакти	1	2	3
Всього предметів	34	50	67

СПИСОК ПРЕДМЕТІВ

Таблиця Є.2.1 – Спорядження (Звичайне)

№	Спорядження	Додатковий ефект	Значення к8
1	Зброя	+1 міці героя	1-2
2	Шолом	+1 витривалості героя	3-4
3	Обладунок	+1 витривалості героя	5-6
4	Броньоване взуття	+1 витривалості героя	7-8

Таблиця Є.2.2 – Спорядження (Рідкісне)

№	Спорядження	Додатковий ефект	Значення к10
1	Зброя	+2 міці героя	1-2
2	Шолом	+2 витривалості героя	3-4
3	Обладунок	+2 витривалості героя	5-6
4	Броньоване взуття	+2 витривалості героя	7-8
5	Взуття швидкості	+1 енергія	9-10

Таблиця Є.2.3 – Припаси (Звичайні)

№	Спорядження	Додатковий ефект	Значення к6
1	Еліксир зцілення	+2 відновлення витривалості героя	1-2
2	Еліксир бадьорості	+2 відновлення енергії	3-4
3	Еліксир сили	+2 міці героя до початку наступного раунду	5-6

Таблиця Є.2.4 – Припаси (Рідкісні)

№	Спорядження	Додатковий ефект	При кидка к6
1	Факел	Дає можливість не витрачати енергію при проходженні 1 підземелля	1-2
2	Камінь переміщення	Дозволяє перемістити значок гільдії на вибраний сектор ігрового поля	3-4
3	Плащ непомітності	Дозволяє пропустити 1 кімнату підземелля	5-6

Таблиця Є.2.5 – Легендарні артефакти

№	Спорядження	Додатковий ефект	При кидка к8
1	Щит	+3 витривалості героя	1-2
2	Лук	+3 міці героя	3-4
3	Посох зцілення	При використанні, +3 відновлення витривалості обраного героя	5-6
4	Чоботи швидкості	+3 енергії	7-8

СПИСОК ПІДЗЕМЕЛЛЯ

Таблиця Є.3.1 – Типи підземелля

Типи підземелля			
2 кімнати	3 кімнати	4 кімнати	5 кімнати
Після повторному випаданні значення при кидку кубика к6, подія заміняється на монстра.			
1-монстр, 2-5 подія, 6-скарб	1-монстр, 2-пастка, 3-5 подія, 6 скарб	1-монстр, 2-3 пастка, 4-5 подія, 6 скарб	1-2-монстр, 3-пастка, 4-5 подія, 6 скарб
Гарантована винагорода при проходженні підземелля			
1-2 Випадкове звич. спорядження, та припас 3-4 Два випадкових звич. спорядження 5-6 Випадкова звич. зброя, звич. броня	1-2 Випадкове звич. спорядження 3-4 Звич. припас на вибір 5-6 Випадкове рідкісне спорядження	1-2 Звич. спорядження на вибір 3-4 Звич. припас на вибір 5-6 Випадкове рідкісне спорядження типу зброя або взуття	1-2 Два випадкових рідкісних спорядження 3-4 Випадкове рідкісне спорядження та рідкісний припас 5-6 Випадковий артефакт

СПИСОК ПАСТОК

У випадковій події на ігровому полі можуть трапитися всі типи пасток. При проходженні підземелля на 2 і 3 кімнати – лише пастки 1 рівня, на 3-4 кімнати – всі типи пасток

Таблиця Є.4.1 – Типи пасток

№	1 рівень	2 рівень
1	Втрата 1 витривалості кожного героя	Втрата 2 витривалості кожного героя
2	Втрата 1 енергії	Втрата 2 енергії
3	Нічого	Втрата одного припасу
4		Втрата спорядження на вибір

СПИСОК МОНСТРІВ

Таблиця Є.5.1 – Слабкі монстри

Монстр	Міць	Витривалість
Кабан	1	4
Кокатрис	1	7
Вовк	2	7
Гігантський кажан	2	9

Таблиця Є.5.2 – Середні монстри

Монстр	Міць	Витривалість
Варг	3	9
Медвідь	3	11
Василиск	4	11
Гаргулья	4	13

Таблиця Є.5.3 – Сильні монстри

Монстр	Міць	Витривалість
Вурдалак	5	13
Трент	6	15
Тролль	7	15
Віндіго	8	17

СПИСОК ПОДІЙ

Таблиця Є.6.1 – Список випадкових подій

№	Подія	Ефект	Кубик к100	Кількість повторів
1	Герой	До вашого гільдії додається герой	1-8	4
2	Ух, монстр!!	Зустріч зі середнім монстром	9-12	4
3	Чарівне джерело	Ви відновлюєте всім героям 2 витривалості	13-16	2
4	Пункт спостереження	Ви можете розвідати обраний вами регіон та дізнатися що знаходиться за жетонами точок інтересу	17-20	2
5	Дрібниця, але приємно	Ви знаходите звичайний припас на ваш вибір	21-24	2
6	Брухт	Ви знаходите звичайне спорядження	25-29	2
7	Сумка з еліксирами	Ви можете вибрати два звичайних припаси на ваш вибір	30-34	2
8	Знахідка	Ви можете вибрати один рідкісний припас	35-39	2
9	Схрон	Ви знаходите рідкісне спорядження на ваш вибір	40-44	2
10	Пощастило	Ви знаходите випадковий рідкісний предмет	45-49	2
11	Час ховатися!!!	Зустріч зі сильним монстром	50-54	2
12	Сумка з еліксирами	Ви можете вибрати два звичайних припаси на вибір	55-59	2
13	Обережно!	Спрацьовує пастка	60-72	6
14	Неприємність	Зустріч зі слабким монстром	73-85	6