ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Оптимізація розміщення спрощених моделей об'єктів інтер'єру»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Кирило СЕРГІЄНКО
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-61
Кирило СЕРГІЄНКО

Керівник: Оксана ЗОЛОТУХІНА
канд. техн. наук, доц.

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Сергієнку Кирилу Вікторовичу

1. Тема кваліфікаційної роботи: «Оптимізація розміщення спрощених моделей об’єктів інтер’єру»

керівник кваліфікаційної роботи Оксана ЗОЛОТУХІНА, канд. техн. наук, доц.,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «30» жовтня 2025 р. № 467.

2. Строк подання кваліфікаційної роботи «19» грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, інформація про процеси створення інтер’єрів дизайнерами, алгоритми процедурної генерації, методи перевірок на просторовий перетин фігур.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз процесу створення інтер’єрів та алгоритмічні підходи до процедурних генерацій
2. Теоретичні основи розробки системи оптимізації розміщення спрощених моделей об’єктів інтер’єру.

3. Розробка генерації інтер’єру з використанням спрощених моделей об’єктів.
4. Тестування роботи програмного модуля.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Опис правил розміщення об’єктів інтер’єру.
   2. Алгоритм генерації.
   3. Практичний результат.
   4. Результати моделювання.
   5. Тестування впливу кількості обов’язкових об’єктів на результати генерації інтер’єру.
   6. Тестування різних комбінацій наборів обов’язкових та звичайних об’єктів інтер’єру.
   7. Тестування впливу кількості правил обмеження розміщення не результати генерації інтер’єрів.

6. Дата видачі завдання «31» жовтня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	31.10-04.11.2025	
2	Вивчення матеріалів для аналізу процесу створення інтер’єрів дизайнерами	05.11-07.11.2025	
3	Дослідження методів процедурної генерації	08.11-15.11.2025	
4	Аналіз обмеження результатів процедурних генерацій за допомогою обмежуючих правил	16.11-17.11.2025	
6	Створення генерації інтер’єрів з використанням спрощених моделей об’єктів	18.11-11.12.2025	
7	Оформлення роботи: вступ, висновки, реферат	10.12–13.12.2025	
8	Розробка демонстраційних матеріалів	14.12–16.12.2025	
9	Попередній захист роботи	17.12-19.12.2025	

Здобувач вищої освіти _____ Кирило СЕРГІЄНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 71 с., 36 рис., 13 табл., 2 додатки, 20 джерел.

Мета роботи – оптимізація розміщення моделей об’єктів інтер’єру за рахунок використання спрощених геометричних фігур.

Об’єкт дослідження – розміщення спрощених моделей об’єктів інтер’єру.

Предмет дослідження – метод оптимізації розміщення спрощених моделей об’єктів інтер’єру.

В рамках проведеної роботи виконано аналіз сучасних процесів проектування та створення інтер’єрів та визначено рутинний етап, який може бути оптимізований для збільшення продуктивності фахівців в сфері дизайну інтер’єрів. Досліджено методики процедурної генерації існуючих рішень в сфері створення автоматичної генерації інтер’єрів та сукупних напрямків.

Розроблено методику визначення правил та конфігурування генерації інтер’єрів. Визначено основні аспекти оптимізації, що дало змогу спростити моделі об’єктів інтер’єру для досягнення швидкодії обчислення та збільшення гнучкості впливу на результат генерації.

Проведено дослідження результатів за рахунок тестів, кожен з яких фокусується на важливих аспектах програмної системи, для визначення зручності використання та актуальності розробленого ПЗ. Проведено аналіз слабких місць системи і яким чином можна досягти найбільшої ефективності при генерації інтер’єрів з спрощеними моделями об’єктів інтер’єру.

КЛЮЧОВІ СЛОВА: ОПТИМІЗАЦІЯ ПРОЦЕСІВ, ГЕНЕРАЦІЯ ІНТЕР’ЄРІВ, СПРОЩЕННІ МОДЕЛІ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ, ODIN, CLAY, RAYLIB.

ABSTRACT

Text part of the qualification work for obtaining a master's degree: 71 pages, 36 figures, 13 tables, 2 appendix, 20 sources.

The purpose of the work is to optimize the placement of interior object models through the use of simplified geometric shapes.

The object of the study is the placement of simplified models of interior objects.

The subject of the study is a method for optimizing the placement of simplified models of interior objects.

Within the scope of this research, an analysis of contemporary processes of interior design and development was conducted, and a routine stage was identified that can be optimized to increase the productivity of specialists in the field of interior design. Existing approaches and methodologies of procedural generation applied to automated interior generation, as well as related directions, were examined.

A methodology for defining rules and configuring the interior generation process was developed. The key aspects of optimization were identified, which made it possible to simplify interior object models in order to improve computational performance and increase the flexibility of influencing the generation results.

An experimental evaluation of the results was carried out through a series of tests, each focusing on critical aspects of the software system, with the aim of assessing usability and the relevance of the developed software. An analysis of the system’s limitations was performed, along with an investigation of approaches to achieving the highest efficiency in generating interiors using simplified models of interior objects.

The purpose of the work to formulate an algorithm for optimizing the placement of interior objects represented by simplified models.

Object of research – process of interior creation by designers and ways to optimize it.

Subject of research – is a method for optimization the placement of simplified interior objects.

KEYWORDS: PROCESS OPTIMIZATION, INTERIOR GENERATION, SIMPLIFIED MODELS, PROCEDURAL GENERATION, ODIN, CLAY, RAYLIB.

ЗМІСТ

ВСТУП

Проєктування інтер'єрів є складним багатоступеневим процесом, який поєднує творчі, технічні та інженерні аспекти. У сучасній практиці дизайнери інтер'єрів працюють з великою кількістю обмежень, серед яких функціональність простору, ергономіка, відповідність будівельним нормам, естетичні вимоги та побажання замовника. На початкових етапах проєктування особливу роль відіграє створення попередніх планувальних рішень, ескізів та заготовок інтер'єру, які слугують основою для подальшої деталізації та візуалізації.

З розвитком комп'ютерних технологій значна частина робочих процесів у сфері дизайну інтер'єрів була перенесена у на рівень роботи в середовищі програмного забезпечення для тривимірного моделювання. Використання інтрументів трьовимірного моделювання дозволяє дизайнерам наочно оцінювати просторові пропорції, взаємне розташування об'єктів, освітлення та загальну композицію інтер'єру ще до початку фізичної реалізації проєкту. Крім того, тривимірне моделювання суттєво полегшує комунікацію між дизайнером і замовником, оскільки воно надає можливість продемонструвати майбутній інтер'єр у зрозумілому вигляді.

Разом з тим, незважаючи на високий рівень розвитку сучасного програмного забезпечення, значна частина роботи дизайнера на ранніх етапах залишається рутинною. Створення первинних заготовок інтер'єру часто передбачає багаторазове розміщення типових об'єктів, таких як меблі, елементи зберігання або сантехнічне обладнання, з подальшою перевіркою їхніх габаритів, взаємних відстаней та відповідності просторовим обмеженням приміщення. Такі дії потребують часу, але водночас слабо впливають на унікальність дизайнерського рішення, оскільки базуються на загальних правилах і стандартах.

У зв'язку з цим є актуальною задача оптимізації рутинних етапів створення інтер'єрних заготовок шляхом автоматизації. Одним із перспективних підходів

до розв’язання цієї задачі є процедурна генерація інтер’єрів, яка дозволяє алгоритмічно формувати варіанти розміщення об’єктів на основі заданих правил, обмежень і параметрів. Процедурний підхід широко застосовується в суміжних галузях, зокрема в комп’ютерних іграх та системах віртуальної реальності, де він довів свою ефективність для генерації складних сцен із великою кількістю елементів.

Для підвищення продуктивності таких систем доцільним є використання спрощених моделей об’єктів інтер’єру. Замість детальних тривимірних моделей застосовуються абстрактні геометричні представлення, зокрема паралелепіпеди, які з достатньою точністю описують габарити об’єктів. Це дозволяє значно зменшити обчислювальну складність операцій перевірки перетинів та дотримання просторових обмежень, що є критично важливим для швидкої генерації великої кількості варіантів.

Метою кваліфікаційної роботи є оптимізація розміщення моделей об’єктів інтер’єру за рахунок використання спрощених геометричних фігур та розробка програмного забезпечення, що надає функціонал генерування інтер’єрів таким чином.

Об’єктом дослідження виступає розміщення спрощених моделей об’єктів інтер’єру.

Предметом дослідження є метод оптимізації розміщення спрощених моделей об’єктів інтер’єру для спрощення рутинних етапів роботи дизайнерів.

Для досягнення поставленої мети потрібно виконати наступні завдання:

- дослідити процеси створення інтер’єрів для визначення найважливіших аспектів прототипування інтер’єрів на ранніх стадіях розвитку інтер’єрів
- проаналізувати сучасні наукові та технічні рішення, які стосуються алгоритмічної та процедурної генерації
- дослідити підходи використання конфігурування правил, як гнучких інструментів обмеження генерацій користувачем

- розробити власний стек технологій для досягнення максимального контролю над процесами та отриманням максимальну кількість ресурсів для оптимізації функціоналу
- визначити набір значень та даних, потрібних для конфігурації генерації інтер'єрів
- розробити власний алгоритм генерації інтер'єрів, на базі вказаної конфігурації та правил обмеження розміщення спрощених моделей об'єктів інтер'єру

Особливу увагу приділити перевірці правильності роботи логіки, можливості знаходження оптимальних параметрів конфігурації для можливості досягнення потрібних результатів в згенерованих інтер'єрах.

Наукова новизна роботи полягає в створенні системи для надання користувачу всього потрібного інструментарію та можливості гнучкого налаштування системи для досягнення потреб в генерації інтер'єрів з використанням спрощених моделей об'єктів інтер'єру на етапі прототипування. Запропонований підхід дає змогу швидко перевіряти ідеї, варіативність результату розширює діапазон доступних рішень.

Практична значущість дослідження визначається можливістю використання програмної системи «Furnugen» дизайнерами нових інтер'єрів або для прототипування варіацій перестановок в рамках існуючих інтер'єрів. При додатковому доопрацюванні, програмний модуль може бути відкритий для використання на широкий загал, розширюючи можливості для використання навіть за межами роботи з інтер'єрами, переходячи в сферу загального моделювання в тривимірному просторі.

Структура роботи складається зі вступу, чотирьох основних розділів роботи, висновків, переліку посилань та додатків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд та аналіз літературних джерел за темою наукового дослідження

В результаті дослідження було виявлено, що процес створення інтер'єрів можна поділити на декілька етапів [27, 28]:

1. Визначення бажань та потреб клієнта. На цьому етапі часто дизайнер допомагає клієнту зрозуміти, що конкретно подобається, а що ні. Які основні переваги дизайну, в чому найбільший пріоритет?

2. Далі на базі зібраної інформації про замовлення дизайнер створює початкові концепти інтер'єру.

3. З запропонованих варіантів клієнт обирає той концепт, який йому найбільше до вподоби і з яким буде проводитися подальша робота. Дизайнер наповнює вибраний концепт конкретними об'єктами інтер'єру, матеріалами та іншим. Цей результат передається клієнту для перевірки. При незгоді з якимись елементами, дизайнер перероблює концепт інтер'єру і надає на перевірку клієнту знову. Таким ітеративним чином інтер'єр вдосконалюється до моменту, поки бажання клієнта не будуть повністю задоволені.

4. Створення документації про інтер'єр і його деталі. Далі дизайнер або передає готову документацію клієнту і на цьому співпраця закінчується, або дизайнер також приймає активну участь у перевірці відтворення деталей інтер'єру.

Для кращої інтерпретації інтер'єру та його правильному розумінню створюється модель, яка повністю або частково показує, як виглядатиме фінальний результат. Наявність моделі інтер'єру надає можливість краще "відчути" фінальний дизайн, як професіоналу, так і звичайній людині. Ще з давніх часів люди планували архітектурні споруди за допомогою спрощених моделей з

глини, за допомогою малюнків та схем. Сучасні дизайнери використовують ілюстрації та тривимірні моделі для візуалізації фінального вигляду [5].

В результаті дослідження було виявлено, що процес створення інтер'єрів можна поділити на декілька етапів [27, 28]:

1) Визначення бажань та потреб клієнта. На цьому етапі часто дизайнер допомагає клієнту зрозуміти, що конкретно подобається, а що ні. Які основні переваги дизайну, в чому найбільший пріоритет?
2) Далі на базі зібраної інформації про замовлення дизайнер створює початкові концепти інтер'єру.
3) З запропонованих варіантів клієнт обирає той концепт, який йому найбільше до вподоби і з яким буде проводитися подальша робота. Дизайнер наповнює вибраний концепт конкретними об'єктами інтер'єру, матеріалами та іншим. Цей результат передається клієнту для перевірки. При незгоді з якимись елементами, дизайнер перероблює концепт інтер'єру і надає на перевірку клієнту знову. Таким ітеративним чином інтер'єр вдосконалюється до моменту, поки бажання клієнта не будуть повністю задоволені.
4) Створення документації про інтер'єр і його деталі. Далі дизайнер або передає готову документацію клієнту і на цьому співпраця закінчується, або дизайнер також приймає активну участь у перевірці відтворення деталей інтер'єру.

Для кращої інтерпретації інтер'єру та його правильному розумінню створюється модель, яка повністю або частково показує, як виглядатиме фінальний результат. Наявність моделі інтер'єру надає можливість краще "відчути" фінальний дизайн, як професіоналу, так і звичайній людині. Ще з давніх часів люди планували архітектурні споруди за допомогою спрощених моделей з глини, за допомогою малюнків та схем. Сучасні дизайнери використовують ілюстрації та тривимірні моделі для візуалізації фінального вигляду [5].

Наскільки важливо мати креслення майбутньої споруди для розуміння її технічних деталей, настільки ж важливо мати візуальне відображення

майбутнього інтер'єру для розуміння його ергономічності та зручності використання елементів в ньому. Через цю важливість сучасні дизайнери все в більших і більших масштабах опановують тривимірне моделювання, використання технологій віртуальної та доповненої реальності [5, 6].

Важливим аспектом створення якісного інтер'єру є як художня, так і ергономічна складова. Попри те, що використання технологій для створення дизайну та інтер'єрів постійно збільшується, їх розвиток ще не дозволяє створювати якісні інтер'єри без професійних дизайнерів. Ці інструменти лиш покращують і спрощують процеси створення дизайну для спеціалістів [1, 9]. В цьому дослідженні основний фокус надаватиметься саме покращенню процесу створення тривимірних концептів.

В сфері створення та проектування тривимірних візуалізацій інтер'єрів вже існує вдосталь інструментів, які надають великий асортимент моделей об'єктів інтер'єру, спрощенні процеси створення рендерів, збереження пресетів і інших полегшень створення інтер'єрів. Але всі ці рішення є мануальними, тобто надають або ширший набір елементів, які може використати дизайнер, або облегшують рутинні завдання, та основну роботу все одно виконує дизайнер власноруч.

В поточному дослідженні увагу буде зосереджено саме на підходах до генерації тривимірних об'єктів для створення інтер'єру. Використання такого функціоналу може сильно спростити рутинний процес підбору розміщення об'єктів інтер'єру для створення базових концептів, що дасть змогу більше часу приділити саме креативній частині створення дизайну - вибору стилю, відповідної фурнітури, візуальних складових фінального дизайну, тощо.

Аналіз вже існуючих підходів і методик до генерації тривимірного контенту показує, що ця проблема не є новою. Для кожної сфери можуть використовуватися різні підходи до генерації об'єктів, деякі з них споріднені за своїми реалізацією, інші ж є абсолютно різними. Проблеми створення алгоритмів для процедурної генерації постають при потребі створення віртуальних світів - архітектурні та дизайнерські концепти, створення світів в відеоіграх, наповнення

віртуальних та доповнених реальностей. Також розвиваються підходи до генерації самих тривимірних об'єктів.[10, 15]

Хоч алгоритми, що створюють рельєф місцевості чи розподіл кімнат, відрізняються від тих, які розміщують тривимірні об'єкти в деякому вказаному просторі, тим не менш, фундаментальні блоки таких алгоритмів часто перетинаються.

Такими блоками є:

- використання «шумів» для обмеження генерації [12];
- випадкові розподілення [13];
- використання рекурсивного підходу [14];
- графова граматика [7];
- симуляційні алгоритми [2].

Всі ці підходи і алгоритми можуть бути успішно використанні для вирішення задач розміщення спрощених об'єктів інтер'єру. але найбільш ефективним є використання комбінацій цих підходів для досягнення більш "реалістичного" та доступного для глибокої конфігурації результату генерації [3, 8, 16].

1.2 Аналіз існуючих алгоритмів та підходів до автоматизації процесу створення інтер’єрів

Генерація в тривимірному просторі може різнитися за конкретною реалізацією, але основним аспектом завжди є обмеження та правила, в рамках яких цей алгоритм генерує результат.

В роботі [4] було задіяно генерацію на базі конфігурованих правил. Основна задача алгоритму - генерація будинків, які статистично реалізовують вказані правила генерації.

Результуючі будівлі є комплексними об'єктами: їх елементи, такі як вікна, двері, колір фасаду, дах, і т. д., кожен з яких має набір характеризуючих

параметрів. Процес створення будівель поділяється на дві фази: початкове налаштування та генерація.

В фазі налаштування дизайнер виділяє територіальний регіон, відносно якого вказуватимуться правила. Далі вказуються присутність тих чи інших елементів, їх параметри, шанс використання того чи іншого значення, вказування текстур, тощо.

В наступній фазі алгоритм на базі регіонів та правил в ньому генерує будинки, фасади яких статистично відповідають вказаним правилам.

Попри те, що цей алгоритм генерує будівлі, а не інтер'єри, підхід до генерації на базі правил є ефективним рішенням. Незважаючи на те, що є потреба в початковому мануальному налаштуванні, це дає більше контролю над результатами генерації.

Алгоритм [20] реалізовує розміщення об'єктів інтер'єру за рахунок детального опису властивостей меблів та інших предметів в кімнаті, на базі чого проводиться генерація з максимальною реалізацією вказаних правил розміщення та обмежень.

Як і в попередньому розглянутому алгоритмі, для генерації проводиться дві фази: фаза вирахування правил розміщення та сама генерація інтер'єру.

Програмний модуль розроблений з можливістю автоматичного аналізу наданого інтер'єру як прикладу. Далі визначені правила та властивості для об'єктів можна змінити чи доповнити за допомогою інтерфейсу в мануальному режимі.

Відносно кожного об'єкта вказуються: поверхні взаємодії, з визначенням «задньої» сторони, визначення центру та допустимого оберту об'єкта, області доступу до об'єкта відносно середньостатистичного людського розміру, область та кут перегляду та інші параметри, які потрібні для оптимізації.

Також у об'єктів можуть бути вказані взаємодії відносно стін чи інших об'єктів інтер'єру, взаємодію розміщення одного об'єкта на/в іншому та парні взаємодії між об'єктами.

Алгоритм генерації реалізований за принципом алгоритму Метрополіса - Гастінгса [11]. На початкових генераціях зміни відносно правил та можливого

зміщення об'єктів є дуже гнучкими, з ітеративним зменшенням допустимих змін, допоки загальні значення реалізації правил об'єктами не досягне стабільного стану. Тобто такого, в якому об'єкти інтер'єру розміщенні найбільш ефективним чином відносно вказаних правил.

В роботі [18] був проведений глибинний аналіз існуючих підходів до генерації інтер'єрів, такі як робота [18] та інші недавні реалізації. Також в роботі наведено важливі проблеми, які пов'язані з реалізаціями алгоритмів, а саме:

- алгоритми, що засновані на оптимізаційній моделі, такі як [18] - з збільшенням кількості елементів в кімнаті зростають потреби в ресурсах, а також час виконання;
- алгоритми, що засновані на обмеженнях - або потребують занадто багато людського втручання, або недостатньо гнучкі для використання з нестандартними схемами зон проживання;
- алгоритми, що отримують інформацію з прикладів інтер'єрів - не враховують схеми інтер'єрів, основна форма яких не є прямокутною чи має більше однією двері.

В їх реалізації є 4 правила розміщення об'єктів - парне розміщення, розміщення біля стіни, розміщення по сітці та розміщення навколо іншого об'єкта.

Для парного розміщення об'єкти мають слідувати обмеженням у відстані та оберті відносно одне одного. При розміщенні біля стіни об'єкт має бути «притиснутий» до стіни або кутку кімнати.

Розміщення по сітці вказує групі об'єктів сітку, відносно якої вони мають бути розміщенні та вирівняні. При розміщенні навколо іншого об'єкта залежні об'єкти мають притримуватися радіального розміщення відного «центрального» об'єкта і підтримувати відповідний кут оберту.

Алгоритм дає змогу користувачу вказати план приміщення, на базі якого алгоритм згенерує тривимірний інтер'єр відносно вказаного типу приміщення - вітальня, спальня, класна кімната чи ресторан.

Користувач може маніпулювати з готовим інтер'єром, повертаючи, переміщуючи, додаючи чи видаляючи об'єкти інтер'єру. Далі перемістити місце погляду в потрібне місце і зберегти зображення інтер'єру.

Реалізація алгоритму вирішує за якими правилами розміщуються елементи відносно вказаного типу приміщення.

2 РОЗРОБКА ОПТИМІЗАЦІЇ РОЗМІЩЕННЯ СПРОЩЕНИХ МОДЕЛЕЙ ОБ'ЄКТІВ ІНТЕР'ЄРУ

2.1 Опис оптимізації процесу створення інтер'єру в трьохвимірному просторі

Для оптимізації розміщення спрощених моделей об'єктів інтер'єру було вибрано два основних аспекти створення інтер'єрів: алгоритмічну складність перевірок можливості розміщення об'єкта в просторі генерації та гнучкість впливу на правила розміщення об'єктів генерації.

Для облегшення алгоритмічної складності перевірок на можливість додавання об'єкта було спрощено підхід до репрезентації об'єктів — замість повноцінних трьохвимірних фігур було вирішено використовувати прості паралелепіпеди, що є межами реальної фігури об'єкта інтер'єру. Така методика дозволяє значно спростити перевірки перетину вже наявних фігур інтер'єру та визначення доступності розміщення відносно вказаного набору правил для об'єкта.

Для покращення гнучкості впливу користувачем на правила створення інтер'єру було надано можливість прямої зміни та додавання правил за потреби. Користувач має повний доступ до конфігурації інформації про об'єкти інтер'єру, які можуть бути використані в генерації, які з об'єктів мають бути обов'язково присутні в результуючому інтер'єрі, відстані між об'єктами, тощо.

2.2 Граматика правил розміщення спрощених моделей об'єктів інтер'єру

Для формалізації опису правил, що обмежують можливості розміщення об'єктів для генерації інтер'єру, було описано граматику структури доступних

правил. В поточній реалізації правило поділяється на 2 види: звичайне правило і правило обов’язкового об’єкта.

Звичайне правило має наступну загальну форму:

<Суб’єкт> <Відношення> { <Відстань> <Об’єкт> | <Розмір> <Вісь> <Об'єкт> | <Розмір> <Вісь> <Скаляр>}

Даний тип правила описує обмеження, що накладаються на можливе розміщення конкретного об’єкта інтер’єру відносно інших об’єктів або числових параметрів.

Суб’єкт визначається як елемент масиву сконфігурованих об’єктів інтер’єру, який перебуває на етапі розміщення. Саме для цього об’єкта перевіряється виконання відповідного правила. Таким чином, суб’єкт є активною сутністю правила, позиція або розмір якої підлягає обмеженню.

Об’єкт у контексті правила є вже розміщеним у сцені елементом інтер’єру. Він виступає просторовою або розмірною точкою відліку, відносно якої формулюється обмеження для суб’єкта.

Такий підхід забезпечує ітеративний характер генерації, коли нові об’єкти враховують конфігурацію вже створеного середовища.

Відношення задає тип порівняння між значенням характеристики суб’єкта та відповідним значенням специфікатора. До множини допустимих відношень належать оператори рівності та нерівності:

==, <=, >=.

Відстань описує відстань між можливою позицією суб’єкта та позицією відповідного об’єкта, виміряна в метрах. Обмеження на відстань дають змогу описувати, наприклад, мінімальні проходи, зони доступності або необхідну близькість функціонально пов’язаних елементів інтер’єру.

Розмір характеризує габаритні параметри суб’єкта (ширину, глибину, та висоту) у співвідношенні з іншим об’єктом або числовим значенням. Це дозволяє

задавати пропорційні обмеження, наприклад, щоб один об’єкт не перевищував певної частки розміру іншого.

Вісь задає напрям вимірювання відповідного розміру та належить до множини ортогональних осей системи координат генерації.

Скаляр є числовою константою, яка використовується для задання абсолютних значень обмежень, незалежних від інших об’єктів сцени.

Правило обов’язкового об’єкта складається тільки з одного параметра:

<Суб’єкт>

2.3 Складність розміщення спрощеної моделі об’єкта відносно вказаного правила розміщення

В таблиці 2.1 наведено формули складності розміщення об’єкта враховуючи особливості перевірки правила при генерації об’єкта, а також перевірки на перетин з уже доданими до інтер’єру об’єктами.

Таблиця 2.1

Визначення мінімальної та максимальної складностей правил розміщення

Правило	Мінімальна складність	Максимальна складність
Обов’язкового об’єкта	O(N)	O(tn * N)
Звичайне відносно іншого об’єкта	O(oN * N)	O(oN * tn * N * gn)
Звичайне відносно константи	O(N)	O(tn * N * gn)

В таблиці 2.2 наведено опис змінних, що використовуються в Таблиці 2.1.

Таблиця 2.2

Опис змінних з Таблиці 2.1

Змінна	Опис
N	Кількість доданих об’єктів інтер’єру на момент перевірки
oN	Кількість доданих об’єктів інтер’єру, відносно яких вказане обмеження
tn	Сконфігурована кількість спроб для розміщення об’єкта
gn	Сконфігурована кількість поколінь спроб розміщення об’єкта

З наведеного порівняння бачимо, що найменша можлива складність перевірок демонструють правило обов’язкового об’єкта та звичайне правило відносно константи за найоптимальніших умов, а найбільш складним правилом є звичайне правило відносно іншого об’єкта за найгірших умов.

2.4 Опис конфігурації об’єктів інтер’єру

Користувач має доступ до конфігурації всіх властивостей об’єкта, так як самі об’єкти не є фіксованими і повністю задаються користувачем.

Параметрами конфігурації об’єкта є:

1. Назва — текстове значення, в якому вказується унікальна назва об’єкта.
2. Тег — текстова позначка, яка вказує на тип об’єкта. В конфігурації генерації може бути вказано декілька об’єктів, які розділяють один і той же тег. В такому випадку всі такі об’єкти підходять для перевірки по звичайному правилу, якщо цей тег був вказаний як суб’єкт або об’єкт.
3. Розмір — значення відповідного розміру об’єкта в генерації. Вказується структура з значень ширини, висоти та глибини.
4. Сторона дотикання в інтер’єрі — описує, яким чином об’єкт інтер’єру прилягає до кімнати. Описується типом даних виду «перелік», який може набувати значень «Стеля», «Підлога», «Стіна» та «Куток».

5. Кількість спроб для розміщення — вказує скільки додаткових спроб буде зроблено алгоритмом для розміщення об’єкта до інтер’єру у випадку невдалої першої спроби.
6. Найбільша кількість екземплярів — вказує найбільшу кількість об’єктів яку можна додати до генерації.

Відповідальність за знаходження діапазону значень параметрів конфігурації об’єктів покладається на користувача разом з повноцінним доступом параметрів для досягнення оптимальних налаштувань генерації.

В залежності від того, який у об’єкта розмір, як багато інших об’єктів сконфігуровані з таким же параметром «Сторона дотикання», як багато правил обмежують розміщення цього об’єкта, як багато об’єктів, за позицією та/або розміром обмежується розміщення цього об’єкта — об’єкт може більш або менш успішно бути доданий до конфігурації.

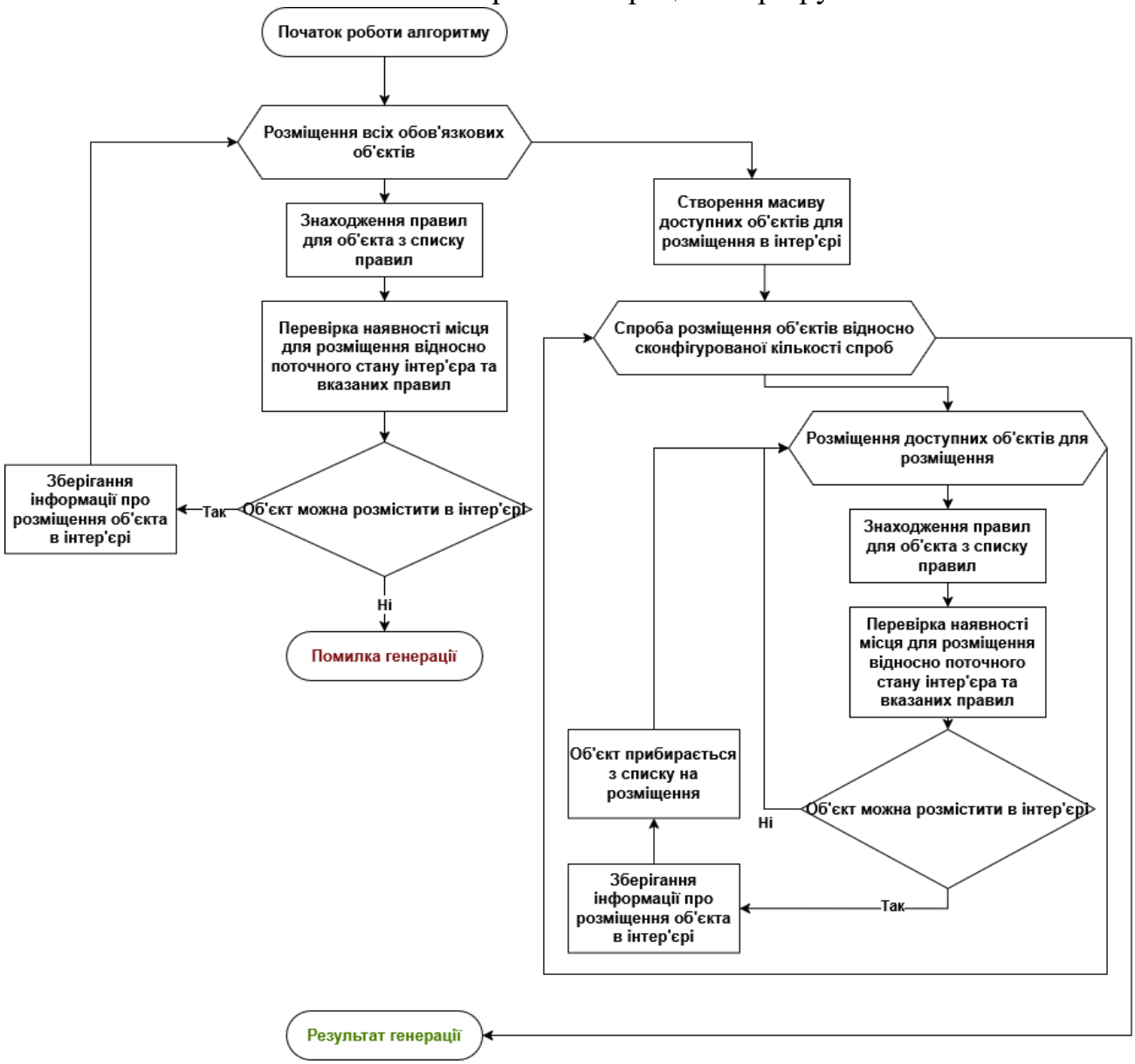
Для збільшення шансу на успішне додавання об’єкта до згенерованого інтер’єру може бути вказане вище значення параметра «Кількість спроб для розміщення».

2.5 Алгоритм генерації інтер’єру

На рисунку 2.1 відображений алгоритм генерації інтер’єру. Алгоритм складаєтсья з 2 фаз: розміщення обов’язкових об’єктів та спроба розміщення всіх інших об’єктів. Але перед запуском алгоритму користувач має вибрати конфігурацію з інформацією, на базі якої буде згенеровано інтер’єр.

Загальний процес генерації інтер’єру можна поділити на 3 фази: конфігурування інформації про генерацію, розміщення обов’язкових об’єктів та спроби розміщення всіх інших об’єктів.

Рис. 2.1 Алгоритм генерації інтер’єру



2.5.1 Конфігурування інформації про генерацію інтер’єру

Перед запуском алгоритму генерації інтер’єру обов’язковим етапом є попередня конфігурація вхідних параметрів системи. На цьому етапі користувач здійснює вибір відповідного набору конфігураційних даних із заздалегідь визначеного переліку.

Обраний набір визначає загальні умови та обмеження, у межах яких відбуватиметься подальший процес генерації, і таким чином безпосередньо впливає на кінцевий результат формування інтер’єру.

У поточній реалізації програмного модуля конфігураційна інформація щодо генерації інтер’єру задається безпосередньо на рівні програмного коду.
Такий підхід спрощує задавання конфігураційних даних для тестування спектру вхідних даних дослідження і сфокусований на отриманні конкретних результатів тестування програмного модуля.

Водночас архітектура системи спроєктована з урахуванням можливості подальшого розширення користувацького інтерфейсу для надання прямого задання інформації про генерацію. Зокрема підтримки конфігурування параметрів безпосередньо користувачем або завантаження відповідних даних із зовнішніх файлів, наприклад у форматах JSON або XML.

Коректність правильного конфігурування інформації про генерацію напряму впливає на результати створення інтер’єру. Важливо враховувати, що всі параметри компонентів пов’язані між собою. Результуючий кінцевий інтер’єр може кардинально змінюватися від найменших змін параметрів конфігурації інформації про генерацію.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ГЕНЕРАЦІЇ ІНТЕР'ЄРУ

3.1 Програмні засоби реалізації

У процесі розробки системи для оптимізації розміщення спрощених моделей об'єктів інтер'єру було використано сучасний стек інструментів, орієнтований на високу продуктивність, контроль над ресурсами та спрощення розробки інтерактивних графічних застосунків.

Зокрема, основною мовою програмування обрано Odin, для реалізації логіки компонування елементів користувацького інтерфейсу використано бібліотеку C Layout (Clay), а для візуалізації та рендерінгу графічних елементів застосовано бібліотеку Raylib, яка доступна для використання разом з компайлером мови програмування.

Мова програмування Odin є компільованою, процедурно-орієнтованою мовою загального призначення, розробленою з урахуванням потреб системного програмування та створення високопродуктивних застосунків. Основними цілями Odin є простота синтаксису, передбачувана поведінка програм та зручність супроводу коду.

Основною філософією цієї мови є пряма робота з структурами даних та управлінням пам'яттю, що робить її близькою за філософією до мови C, але водночас більш безпечною та виразною. Мова має вбудовані засоби для роботи з динамічними масивами, структурами, переліками та модулями, що сприяє створенню чіткої та масштабованої архітектури програмного забезпечення.

Для організації розміщення елементів користувацького інтерфейсу в роботі було використано бібліотеку C Layout (Clay). Дана бібліотека реалізує підхід декларативного опису структури інтерфейсу та автоматичного обчислення позицій і розмірів елементів на основі заданих обмежень. Clay є реалізацією концепцій, подібних до flexbox або box layout, але адаптованою для використання в середовищі мови C та сумісних із нею мов програмування, зокрема Odin.

Основною перевагою Clay є її незалежність від конкретної системи рендерінгу. Бібліотека не займається відображенням елементів на екрані, а лише обчислює геометричні параметри інтерфейсу. Такий підхід дозволяє чітко розділити логіку компонування та візуалізації, що відповідає принципам модульності та слабкої зв’язаності програмних компонентів.

Безпосередній рендерінг графічних елементів та взаємодія з графічним обладнанням здійснювалися за допомогою бібліотеки Raylib. Raylib є кросплатформною бібліотекою з відкритим вихідним кодом, призначеною для спрощення розробки графічних застосунків, ігор та інтерактивних візуалізацій. Вона надає високорівневий інтерфейс для роботи з вікнами, 2D- та 3D-графікою, текстурами, шрифтами, введенням з клавіатури та миші, а також аудіо.

Суттєвою перевагою Raylib є її мінімалістичний дизайн та низький поріг входження, що дозволяє зосередитися на логіці застосунку, не заглиблюючись у складні деталі графічних API, таких як OpenGL або Vulkan. Водночас бібліотека забезпечує достатній рівень продуктивності для використання в застосунках реального часу. Завдяки тому, що Raylib написана мовою C, вона легко інтегрується з Odin і добре поєднується з бібліотекою Clay, яка постачає готові дані для відображення інтерфейсних елементів.

3.2 Архітектура системи

Рис. 3.1 Переходи між станами системи, які представлені сторінками програми

Система поділяється на 3 стани, кожен з яких представлений сторінкою візуального інтерфейсу програми. На рисунку 3.1 зображено ці три сторінки: головна сторінка, сторінка вибору шаблону конфігурації генерації та сторінка генерації та відображення інтер'єру.

Головна сторінка виступає початковим станом системи, з якої користувач починає свою взаємодію з програмою. На рисунку 3.2 зображено, що головна сторінка складається з назви програми «Furnugen», кнопки «Change Gen. Rules», що є скороченою версією «Change Generation Rules», тексту з назвою поточного обраного шаблону конфігурації і кнопки «Generate Interior».

Рис. 3.2 Головна сторінка програми

Натискання кнопки «Change Gen. Rules» змінить стан програму і користувачу буде відображення сторінка вибору шаблону конфігурації генерації. Після того як користувач успішно обрав шаблон конфігурації та повернувся на головну сторінку — текст біля цієї кнопки буде оновлено для відображення поточного обраного набору.

При натисканні кнопки «Generate Interior» буде проведена перевірка на те, чи був вибраний шаблон конфігурації. У випадку, якщо набір конфігурації не був вибраний перед кліком на кнопку, користувачу буде відображено помилку, а кнопку буде деактивовано для взаємодії, тобто вона стане сірою і не реагуватиме на кліки, як показано на рис. 3.3. Після того як користувачем буде обраний коректний набір конфігурації генерації інтер'єру, кнопка «Generate Interior» знову стане активна. Якщо ж при натискані кнопки вибраний набір є прильним,

програма змінить стан і користувачу буде відображена сторінка генерації та відображення інтер’єру.

При переході між станами системою зберігається останній вибраний набір конфігурації генерації допоки користувачем не буде вибраний новий набір конфігурації на сторінці «Change Gen. Rules».

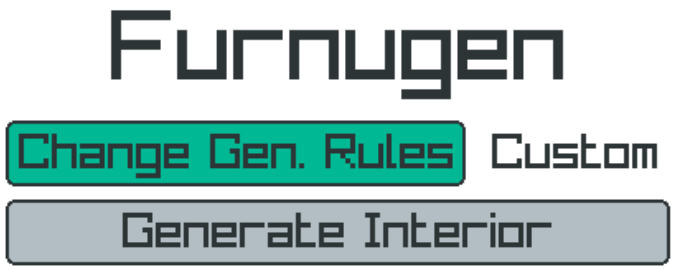


Рис. 3.3 Кнопка «Generate Interior» деактивована,
поки не буде обрано правильний шаблон конфігурації

Сторінка вибору шаблону конфігурації генерації поділяється на меню в лівій частині сторінки і відображенні сконфігурованої інформації для генерації відносно вибраного шаблону в правій частині сторінки, як зображено на рисунку 3.4.

Ліва частина виконує роль меню для управління сторінкою, надаючи користувачу контрол для вибору шаблону конфігурації та кнопку для підтвердження вибору і переходу в головне меню - «Confirm Rules».

При зміні вибраного шаблону одразу ж змінюється і блок відображення інформації про конфігурацію генерації.

Права частина виконує інформаційную функцію, відображаючи користувачу інформацію про конфігурацію, а саме: інформацію про описані об’єкти інтер’єру, які використовуватимуться при генерації інтер’єру — блок «Specified interior objects», які саме об’єкти інтер’єру вказані як обов’язкові — блок «Interior objects set as mandatory» та блок «Rules», в якому описані вказані звичайні правила обмеження розміщення об’єктів.

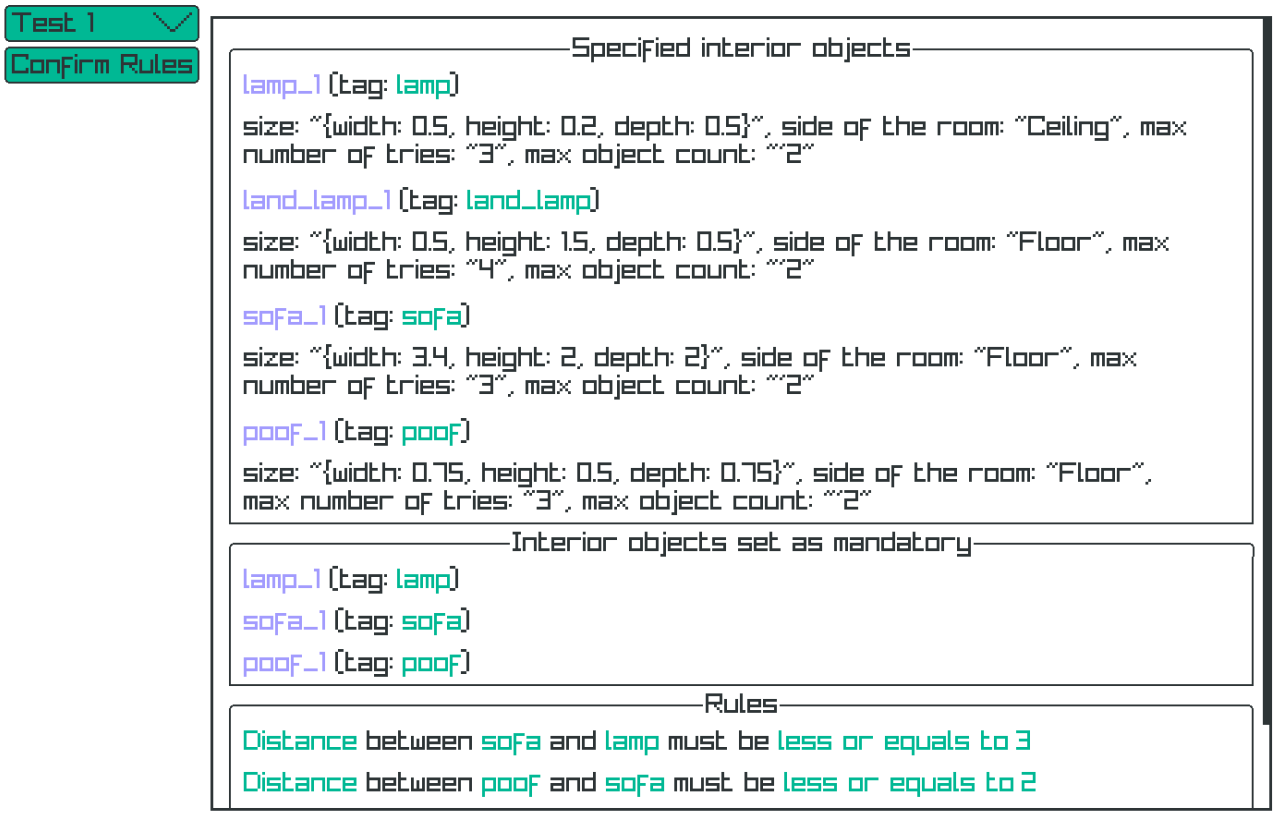


Рис. 3.4 Сторінка вибору шаблону конфігурації генерації з вибраним шаблоном «Test 1»

В інформаційному блоці про конфігурацію генерації важливі параметри були виділені кольором, для легшої візуальної навігації при перегляді чи порівнянні інформацій про конфігурації.

Сторінка генерації та відображення інтер'єру може відобразити один з двох своїх станів при відкритті: інтер'єр було успішно згенеровано і він відображається в центрі сторінки, як показано на рисунку 3.5, або інтер'єр не вийшло згенерувати через неуспішні спроби розміщення обов'язкового об'єкта, що буде вказано при відображенні помилки, як показано на рисунку 3.6.

Якщо нова спроба генерації інтер'єру не була вдалою, то на сторінці відображається попередній успішно згенерований інтер'єр.

Сторінка генерації та відображення інтер'єру розділена на ліву і праву частину. В лівій частині представлені кнопки «Generate new interior», яка запускає процес повторної генерації інтер'єру, та кнопки «Back», яка повертає користувача на головну сторінку. В правій частині відображається спрощена модель кімнати,

в якій буде згенеровано інтер’єр і спрощені моделі об’єктів інтер’єру, які розміщені всередині цих граней кімнати відповідно до вказаних параметрів про об’єкт, таких як прилягаюча частина кімнати, розмір, максимальна можлива кількість об’єктів в кімнаті, тощо.

Інтер’єр може бути успішно згенерований тільки в тому випадку, якщо розміщення об’єктів в інтер’єрі задовольняє вказані правила з конфігурації генерації.

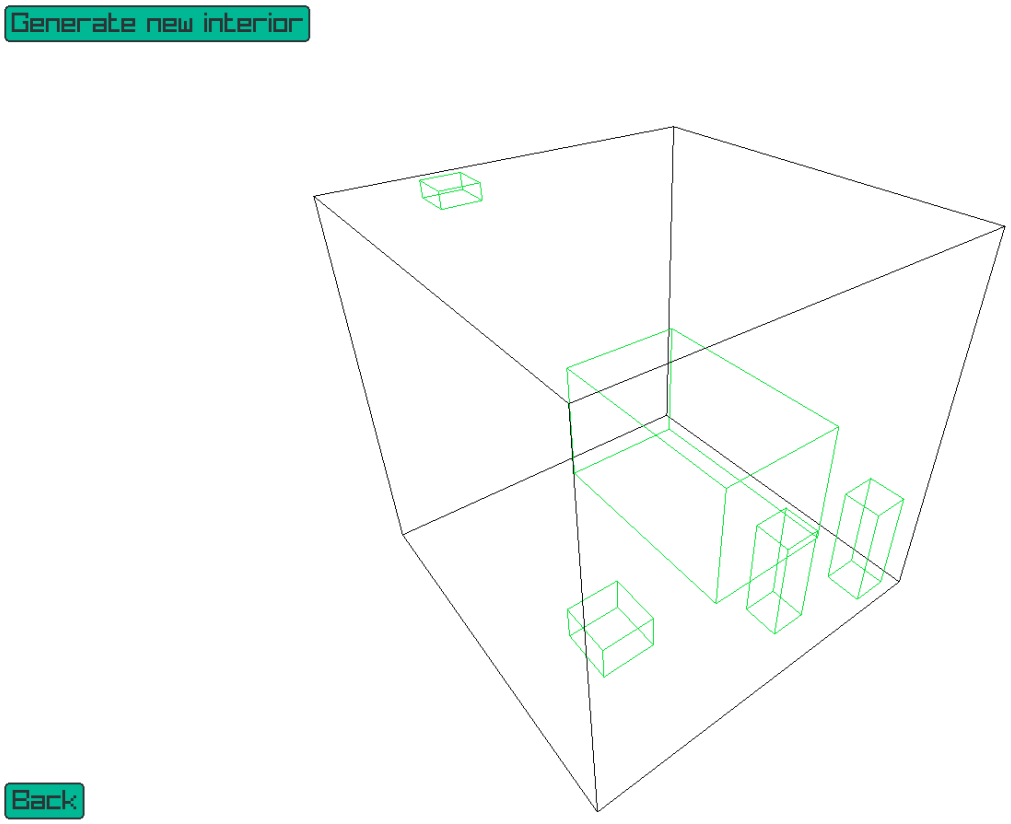


Рис. 3.5 Сторінка генерації та відображення інтер’єру
з успішно згенерованим інтер’єром

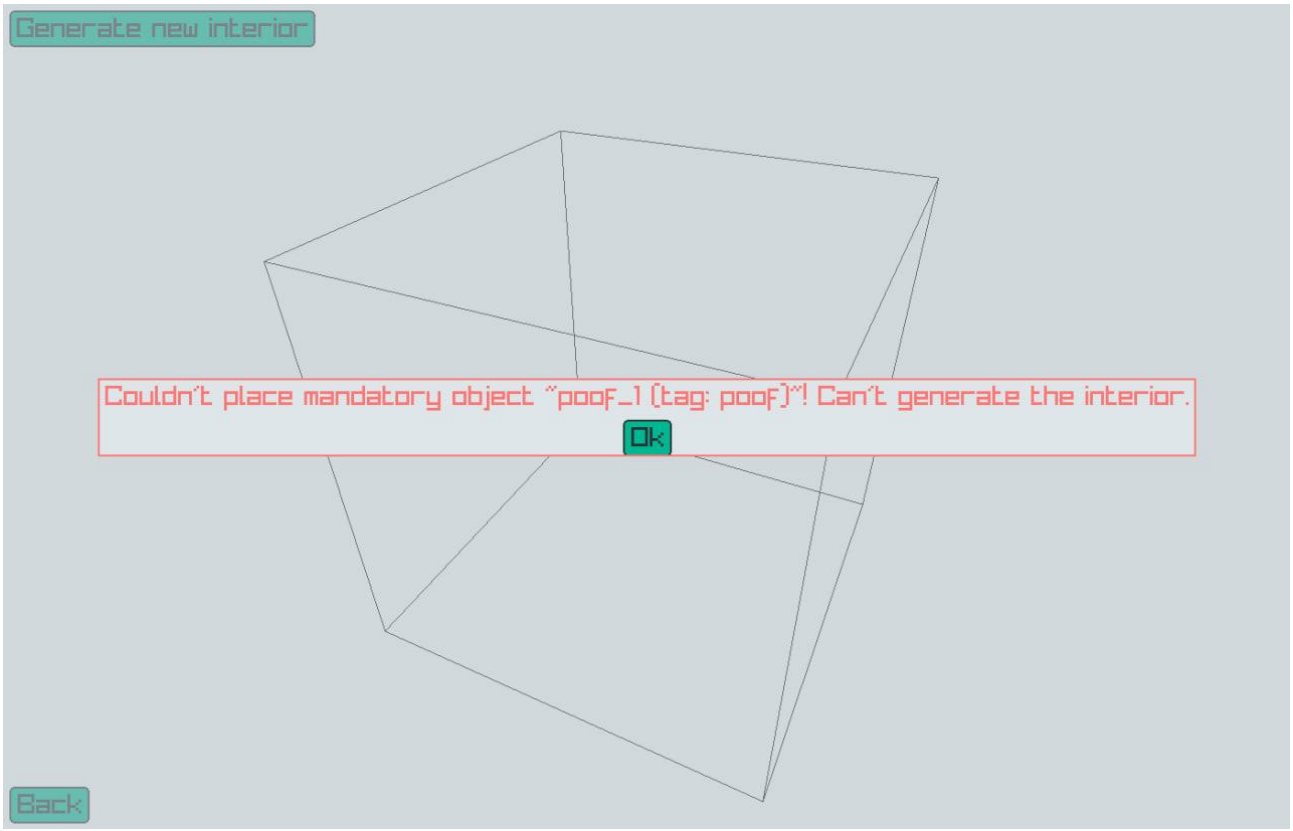


Рис. 3.6 Сторінка генерації та відображення інтер'єру з відображенням помилки генерації інтер'єру через неможливість розміщення обов'язкового об'єкта

3.3 Методика генерації

При генерації інтер'єру алгоритм ітеративно проходить по всім правилам обов'язкових об'єктів, знаходячи відповідну інформацію про об'єкт. Далі для кожного з них виконується процедура «try_add_object», з передаванням параметрів для спроби додавання об'єкта до інтер'єру. Цими параметрами є: масив вже доданих об'єктів інтер'єру, розмір кімнати, інформація про об'єкт, масив звичайних правил розміщення об'єктів інтер'єру, максимальна кількість спроб для розміщення.

Якщо обов'язковий об'єкт не вийшло додати до інтер'єру після сконфігурованої кількості спроб — генерація зупиняєтсья і вважається неуспішною. Після цього користувачу відображається повідомлення про помилку генерації з специфікацією обов'язкового об'єкта, який не вдалося додати до інтер'єру, як це показано на рисунку 3.6.

Далі алгоритм формує «список можливих об’єктів інтер’єру». Алгоритм ітеративно проходить по масиву інформацій про об’єкти, ігноруючи об’єкти, які вказані як обов’язкові. По кожному об’єкта генерується випадкове числове значення, в діапазоні від нуля і до значення параметра «максимальна кількість екземплярів» з інформації про об’єкт. В розмірі цього числа алгоритм додає кількість екземплярів цього об’єкта до списку.

Далі з списку можливих об’єктів інтер’єру випадковим чином обирається можливий об’єкт і виконується спроба додати цей об’єкт до інтер’єру за допомогою процедури «try_add_object». На відміну від обов’язкового об’єкта, якщо додати об’єкт не вдалося, алгоритм зменшує значення кількісті проведених спроб додати об’єкт до інтер’єру. Коли кількість спроб сягне нулю — екземпляр об’єкта прибирається з списку можливих об’єктів інтер’єру. У випадку успішного додавання об’єкта до інтер’єру екземпляр також прибирається з списку.

Алгоритм обирає наступний випадковий екземпляр, допоки в списку не закінчаться екземпляри і він не буде пустий.

Після цього генерація інтер’єру вважається успішно завершеною і відображається на сторінці генерації та відображення інтер’єру.

Процедура «try_add_object» має просту структуру. Ітеруючи до вказаної кількості можливих спроб розміщення проводиться ряд перевірок.

Спочатку випадковим чином обирається позиція в рамках кімнати для розміщення цього об’єкта. Для вибору позиції враховується розмір об’єкта і сторона дотичності до кімнати, щоб об’єкт не виходив за рамки вказаних розмірів кімнати.

Далі ітеруються всіх правила розміщення, в яких цей об’єкт вказаний суб’єктом обмеження. Якщо правило вказує об’єкт обмеження, тоді з списку вже доданих об’єктів інтер’єру відфільтровуються всі об’єкти з відповідним тегом. Вважається, що суб’єкт не порушує правило тільки якщо воно виконується для всіх об’єктів вказаних в цьому правилі, які вже були додані до інтер’єру.

Останнім етапом є перевірка на відсутність перетину спрощеної моделі об’єкта з усіма вже доданими до інтер’єру об’єктами.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Тестування системи

Для визначення висновків дослідження аспектів генерації та впливу відповідних параметрів конфігурації було проведено тестові генерації інтер'єрів. Всі інтер'єри генерувались в рамках кімнати з довжиною 5 метрів, шириною 5 метрів та висотою 5 метрів. Ця висота кімнати була вибрана для кращої візуальної чіткості результуючих інтер'єрів.

Для кожного окремого тесту було проведено 20 генерацій, що зменшує ступінь впливу випадкових чинників на аналіз результатів. Окрім цього кожен тест буде поділений на етапи, в яких загальна складність конфігурації буде збільшуватися для більш чіткої перевірки конкретної тези тесту.

Для проведення тестів використовувались наступні об'єкти інтер'єру:

- Диван - з назвою та тегом «sofa». Є найбільшим об'єктом в представлених кофнігураціях інтер'єрів;
- Торшер — з назвою та тегом «land_lamp»;
- Придиванний стіл — з назвою та тегом «table»;
- Лампа на стелі — з назвою та тегом «lamp»;

Метою першого тесту є отримання даних про вплив вказаної кількості обов'язкових об'єктів на результуючий інтер'єр. Основна гіпотеза полягає в тому, що зі збільшенням обов'язкових об'єктів зменшується загальний відсоток успішних генерацій.

На рисунку 4.1 представлено початкову конфігурацію генерації інтер'єру на першому етапі першого тесту.

На першому етапі маємо 4 сконфігуровані об'єкти інтер'єру, тільки один з яких вказаний обов'язковим - «sofa».

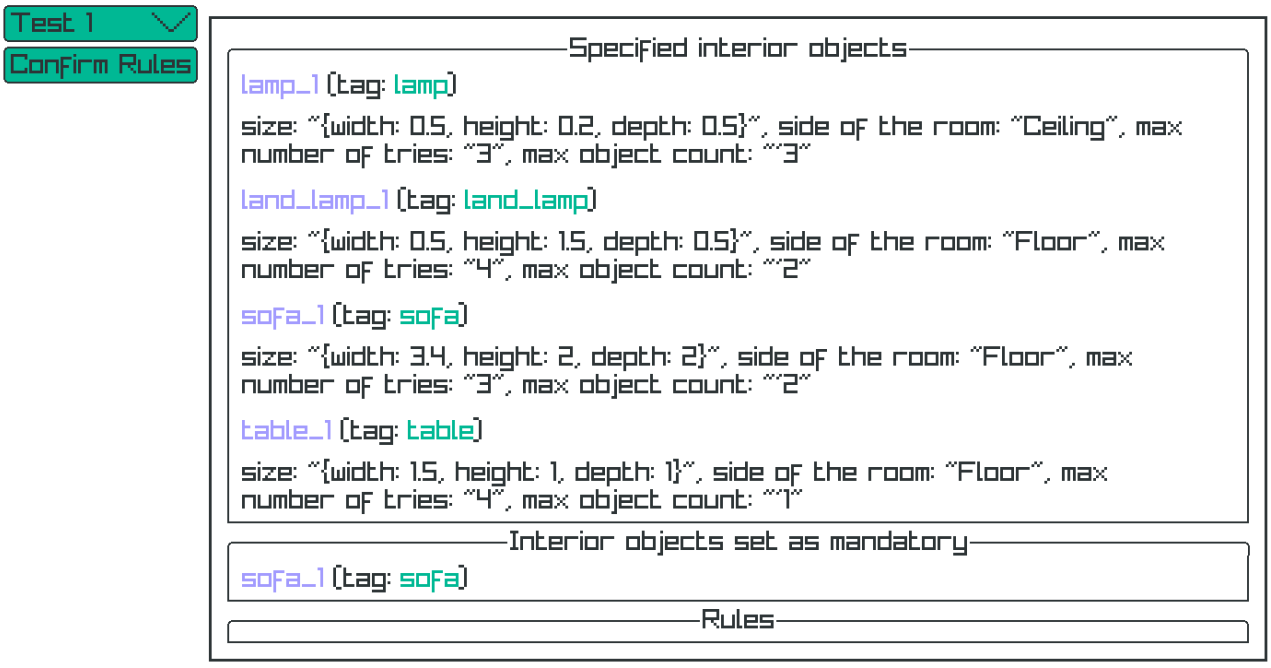


Рис. 4.1 Початкова конфігурація першого етапу першого тесту

В таблиці 4.1 наведено результати 20 генерацій інтер'єрів використовуючи конфігурацію для першого етапу.

Таблиця 4.1

Результати генерації 20 інтер'єрів для першого етапу першого тесту

№ Генерації	Кількість об'єктів «sofa»	Кількість об'єктів «land lamp»	Кількість об'єктів «table»	Кількість об'єктів «lamp»
1	1	1	1	2
2	1	2	1	2
3	1	1	1	3
4	1	2	1	3
5	1	2	1	3
6	1	2	1	1
7	1	1	1	2

Продовження таблиці 4.1

Результати генерації 20 інтер’єрів для першого етапу першого тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
8	1	1	1	1
9	1	1	1	2
10	1	2	1	3
11	1	2	1	1
12	1	1	1	1
13	1	1	1	2
14	1	2	1	1
15	1	1	1	2
16	1	2	1	2
17	1	2	1	2
18	1	1	1	2
19	1	1	1	1
20	1	2	1	2

Так як обов’язковим об’єктом в генерації виступа тільки один «диван», то і кінцевий інтер’єр був наповнений всіми вказаними об’єктами, в тому числі і доволі габаритним «придиванним столиком».

Приклади генерацій інтер’єрів з номерами 10 та 20 відображені на рисунках 4.2 та 4.3 відповідно.

Результати першого етапу дають нам можливість відштовхуватися від цих значень для порівняння з результатами наступного етапу першого тесту.

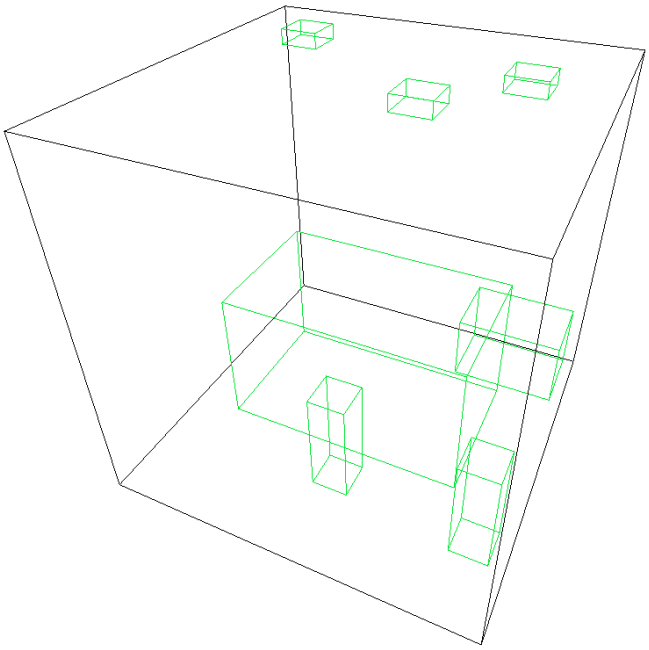


Рис. 4.2 Згенерований інтер’єр номер 10 першого етапу першого тесту

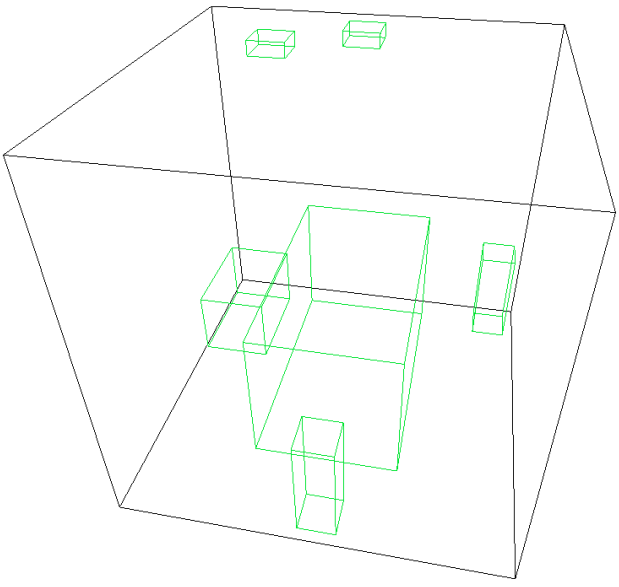


Рис. 4.3 Згенерований інтер’єр номер 20 першого етапу першого тесту

Для другого етапу тестування в масив правил обов’язкових об’єктів інтер’єру було додано також правило для об’єкта «торшер». Тобто, на другому етапі першого тесту масив правил обов’язкових об’єктів складається з «sofa» та «land_lamp».

Таблиця 4.2

Результати генерації 20 інтер’єрів для другого етапу першого тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	1	1	1	2
2	1	1	1	1
3	1	1	1	1
4	1	1	1	2
5	1	1	1	2
6	1	1	1	2
7	1	1	1	2
8	1	1	1	3
9	1	1	1	2
10	1	1	1	2
11	1	1	1	3
12	1	1	1	1
13	1	1	1	1
14	1	1	1	1
15	1	1	1	2
16	1	1	1	2
17	1	1	1	2
18	1	1	1	1
19	1	1	1	2
20	1	1	1	3

Результати генерацій другого етапу першого тесту є дуже схожими на результати генерацій першого етапу першого тесту. Це обумовлено тим, що різницею двох конфігурацій виступає вказування об’єкта «торшер» як обов’язкового. Так як алгоритм завжди генерує тільки один об’єкт на кожне правило обов’язкового об’єкта, кількість об’єктів «торшер» зменшилася до одного у всіх випадків.

Візуальне відображення інтер’єрів з генерацій номер 10 та 20 можна побачити на рисунках 4.4 та 4.5 відповідно.

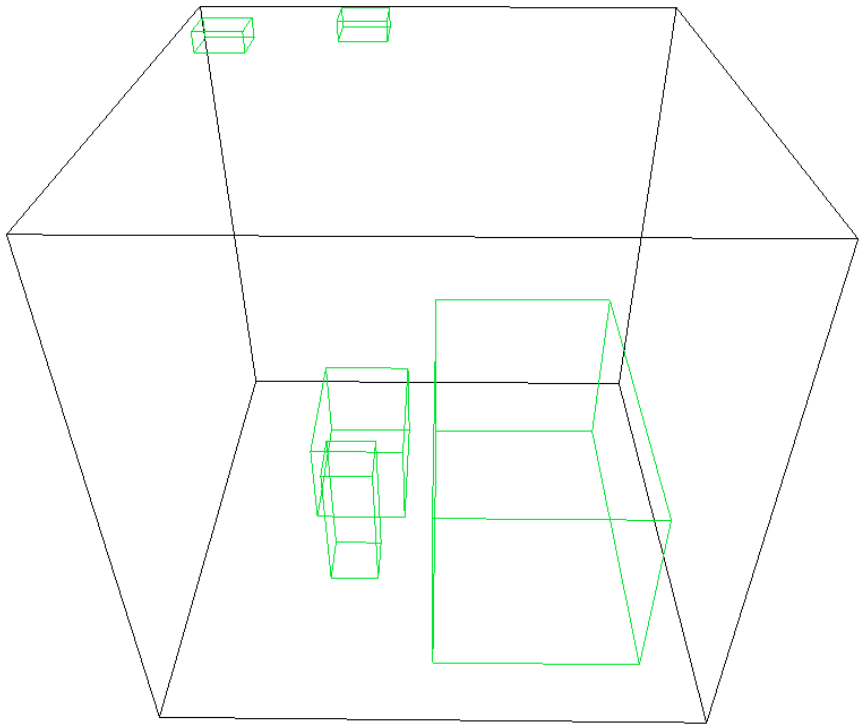


Рис. 4.4 Згенерований інтер’єр номер 10 другого етапу першого тесту

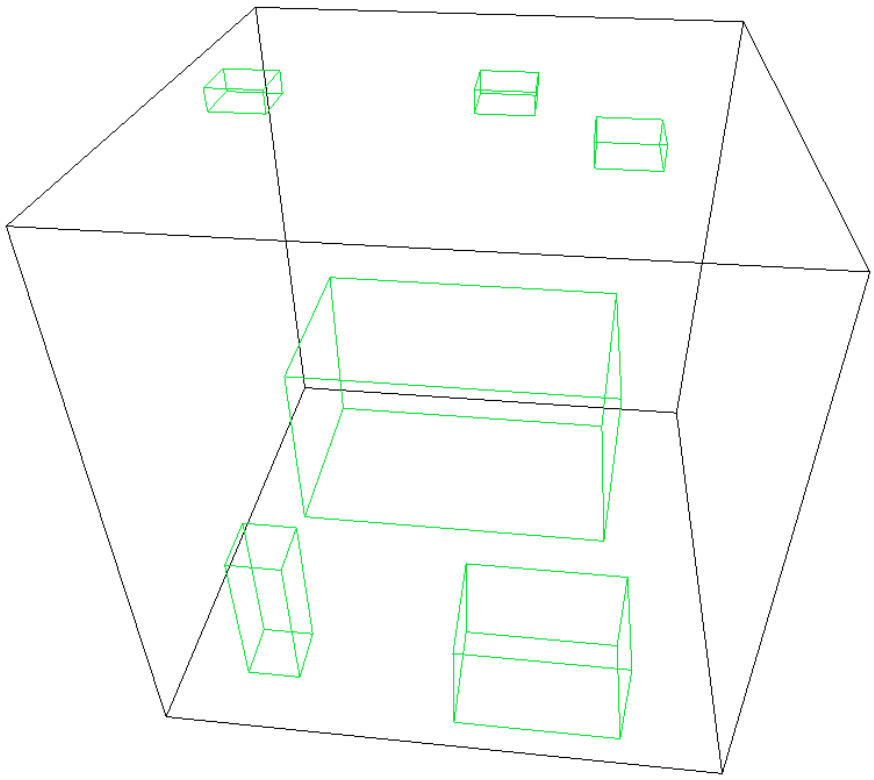


Рис. 4.5 Згенерований інтер’єр номер 20 другого етапу першого тесту

Для третього етапу першого тесту до масиву правил було додано правило третього обов’язкового об’єкта - «table».

Таблиця 4.3

Результати генерації 20 інтер’єрів для третього етапу першого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	«table»				
2	«table»				
3		1	1	1	2
4		1	1	1	1
5		1	1	1	3

Продовження таблиці 4.3

Результати генерації 20 інтер’єрів для третього етапу першого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
6	«table»				
7		1	1	1	3
8		1	1	1	1
9		1	1	1	3
10		1	1	1	1
11		1	1	1	2
12		1	1	1	1
13	«table»				
14	«land lamp»				
15		1	1	1	1
16	«table»				
17	«table»				
18		1	1	1	1
19	«table»				
20		1	1	1	3

За результатами, представленими в таблиці 4.3, бачимо, що наявність 3 правил обов’язкових об’єктів вже призводить до того, що деякі з генерацій є неуспішними, так як не алгоритм не зміг розмістити обов’язковий об’єкт в кількість спроб, дозволену для об’єкта.

Кількість неуспішних генерацій можна зменшити більшою кількістю спроб розміщення для об’єктів.

Важливо враховувати, що відношення розмірів об’єкта до розміру кімнати та кількості правил обов’язкових об’єктів напряму впливають на успішність розміщення, тому це треба враховувати вказуючи кількість спроб для розміщення

об’єкта інтер’єру. Інтер’єри з генерацій номер 10 та 20 зображені на рисунках 4.6 та 4.7 відповідно.

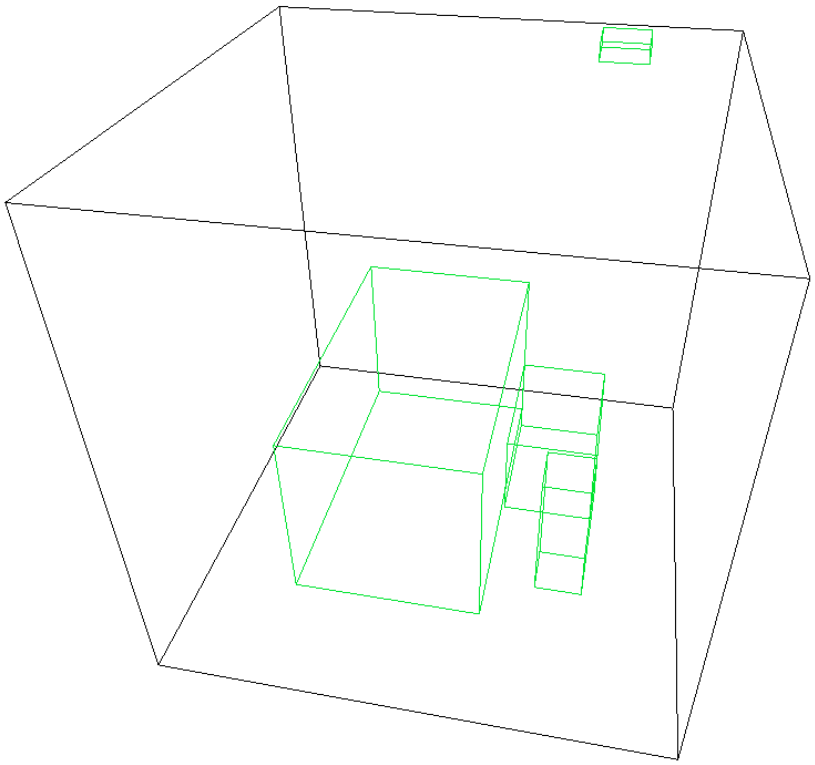


Рис. 4.6 Згенерований інтер’єр номер 10 третього етапу першого тесту

В четвертому етапі першого тесту до списку правил обов’язкових об’єктів було додано ще один об’єкт «land_lamp». Таким чином список правил обов’язкових об’єктів складається з «sofa», двох «land_lamp» та «table». Результати 20 генерацій інтер’єрів з конфігурацією для четвертого етапу першого тесту представлені в таблиці 4.4.

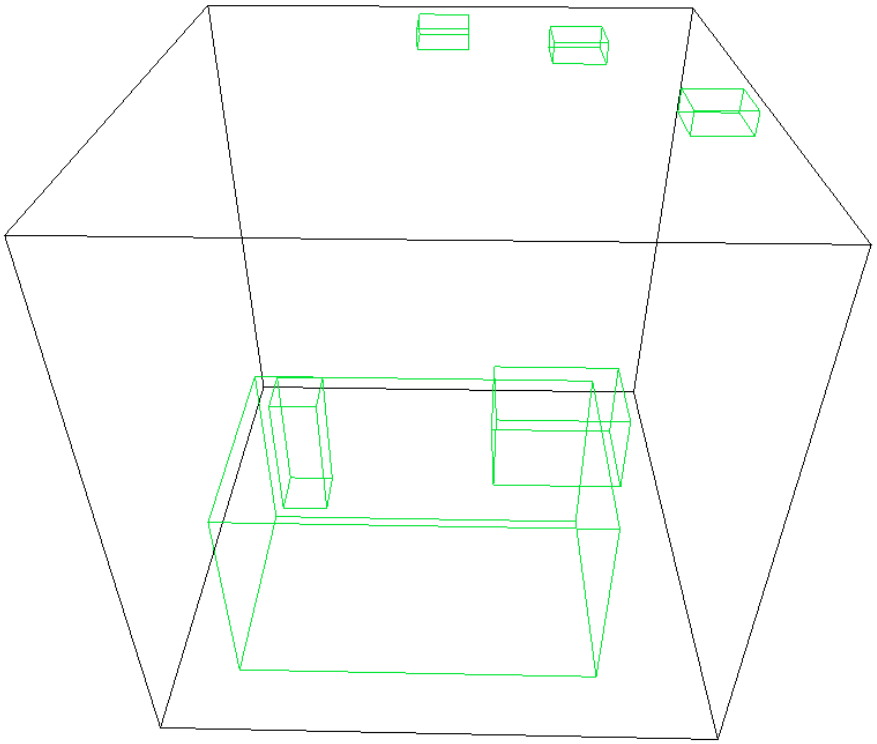


Рис. 4.7 Згенерований інтер’єр номер 20 третього етапу першого тесту

Таблиця 4.4

Результати генерації 20 інтер’єрів для четвертого етапу першого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	«table»				
2	«table»				
3		1	2	1	3
4	«table»				
5		1	1	1	3
6		1	2	1	2
7	«table»				
8		1	2	1	2
9	«table»				

Продовження таблиці 4.4

Результати генерації 20 інтер’єрів для четвертого етапу першого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
10	«table»				
11		1	2	1	2
12	«table»				
13		1	2	1	3
14		1	1	1	1
15		1	1	1	3
16		1	2	1	2
17	«table»				
18		1	1	1	2
19		1	2		1
20	«table»				

З результатів створених інтер’єрів бачимо, що додавання четвертого правила про обов’язковий об’єкт збільшило кількість неуспішних генерацій інтер’єру.

На третьому етапі кількість неуспішних генерацій сягала приблизно 30% від всіх генерацій. З додаванням четвертого правила обов’язкового об’єкта, ця кількість збільшилась до 45%.

Інтер’єри під номерами 8 та 19 тестування четвертого етапу першого тесту зображені на рисунках 4.8 та 4.9 відповідно.

По результатам проведення першого тесту, після 4 етапів тестування з збільшенням кількості правил обов’язкових об’єктів на кожному наступному етапі, можна визначити, що кількість обов’язкових об’єктів в генерації має великий вплив на вірогідність успішної генерації інтер’єру.

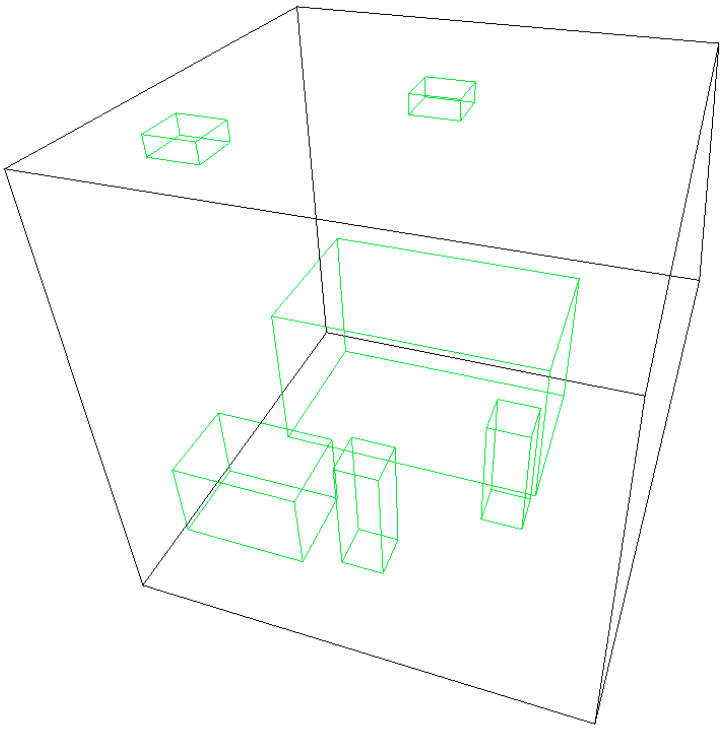


Рис. 4.8 Згенерований інтер’єр номер 8 четвертого етапу першого тесту

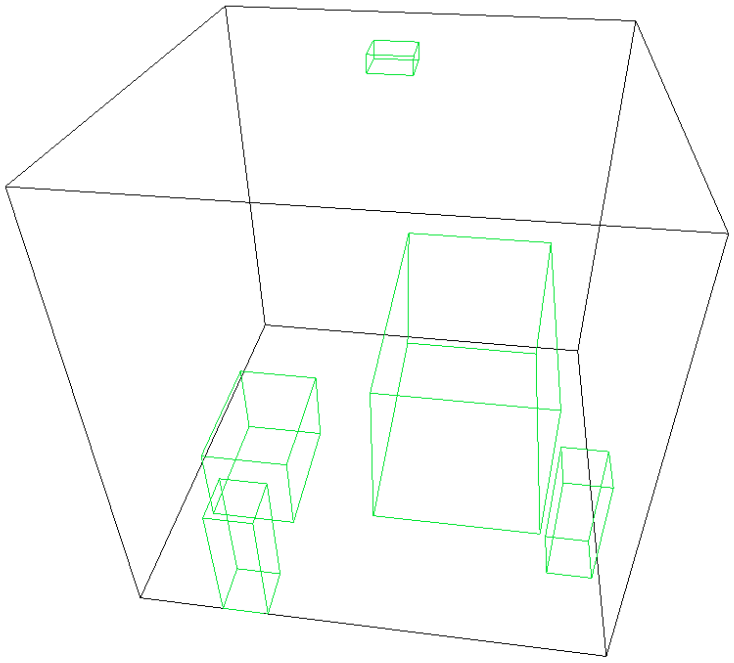


Рис. 4.9 Згенерований інтер’єр номер 19 четвертого етапу першого тесту

Можна зауважити, що для об’єктів генерацій, кількість правил обов’язкових об’єктів яких сягає більше двох, мають бути збільшені кількості спроб розміщення.

Формулу для визначення цієї кількості визначити майже неможливо, так як факторами впливу є відношення розміру об’єкта до розміру кімнати інтер’єру, розмір об’єкта до розмірів інших обов’язкових об’єктів, а також загальна кількість цих обов’язкових об’єктів, і ці значення мають знаходитися екпериментально для кожної окремої конфігурації генерації інтер’єру.

На рисунку 4.10 представлено початкову конфігурацію першого етапу другого тесту. Другий тест сфокусований на досліджені впливу наявності обов’язкових об’єктів на розміщення всіх інших об’єктів інтер’єру.

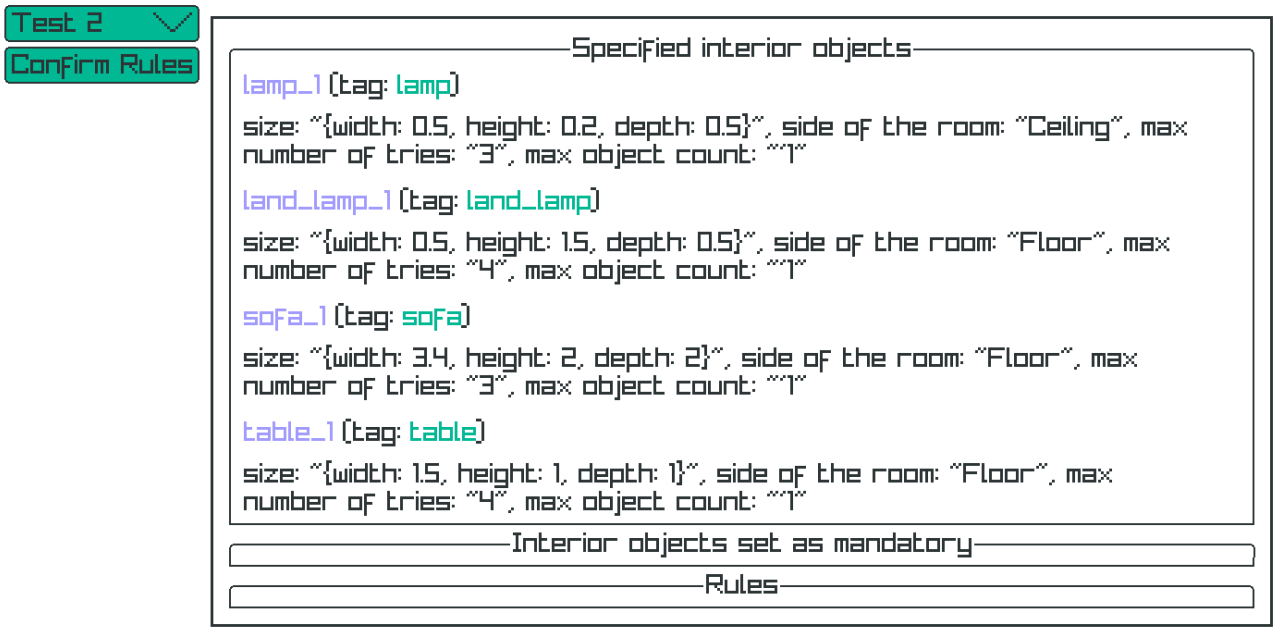


Рис. 4.10 Початкова конфігурація першого етапу другого тесту

На першому етапі в конфігурації не вказані обов’язкові об’єкти інтер’єру, для всіх об’єктів параметр максимальної кількості екземплярів вказаний зі значенням один, для створення базових результатів генерацій з якими можна буде порівнювати результати наступних етапів другого тесту.

Таблиця 4.5

Результати генерації 20 інтер’єрів для першого етапу другого тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1
8	1	1	1	1
9	1	1	1	1
10	1	1	1	1
11	1	1	1	1
12	1	1	1	1
13	1	1	1	1
14	1	1	1	1
15	1	1	1	1
16	1	1	1	1
17	1	1	1	1
18	1	1	1	1
19	1	1	1	1
20	1	1	1	1

Результати відтворюють початкове очікування від конфігурації для першого етапу — всі об’єкти інтер’єру мають достатню кількість вказаних спроб для розміщення, щоб кожен з вказаних об’єктів був присутній на кожній з двадцяти генерацій інтер’єрів. Інтер’єри з номерами генерацій 10 та 20 зображені на рисунках 4.11 та 4.12 відповідно.

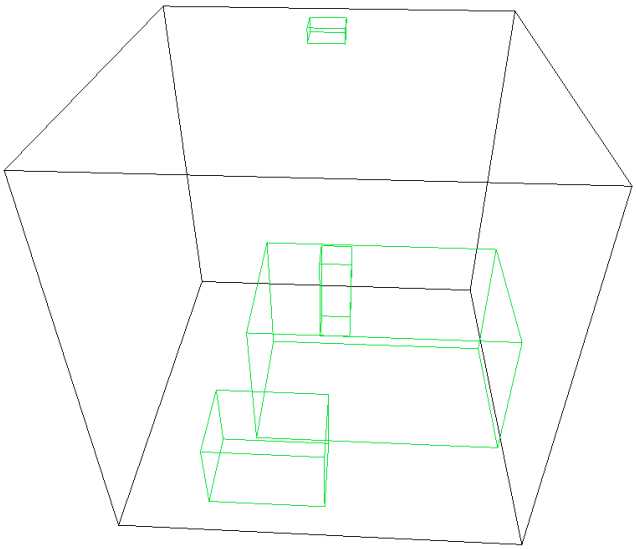


Рис. 4.11 Згенерований інтер’єр номер 10 першого етапу другого тесту

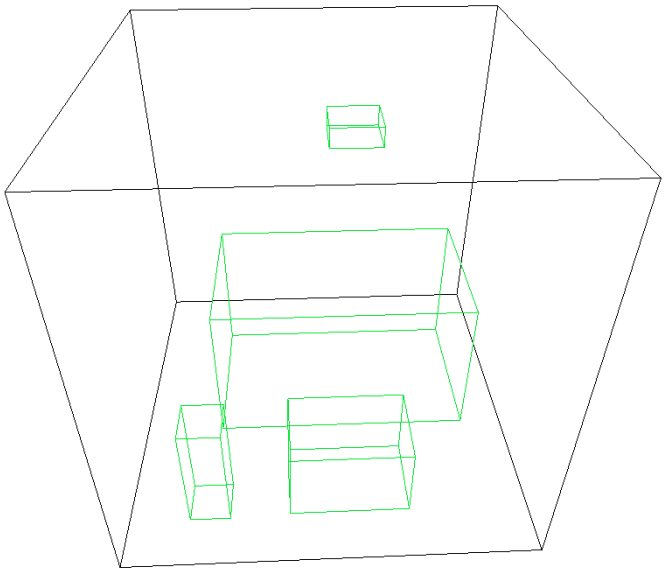


Рис. 4.12 Згенерований інтер’єр номер 20 першого етапу другого тесту

Для тестування другого етапу другого тесту в конфігурації генерації інтер’єру було вказане правило обов’язкового об’єкта, а саме «sofa».

Таблиця 4.6

Результати генерації 20 інтер’єрів для другого етапу другого тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	1	1	1	1
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1
5	1	1	1	1
6	1	1	1	1
7	1	1	1	1
8	1	1	1	1
9	1	1	1	1
10	1	1	1	1
11	1	1	1	1
12	1	1	1	1
13	1	1	1	1
14	1	1	1	1
15	1	1	1	1
16	1	1	1	1
17	1	1	1	1
18	1	1	1	1
19	1	1	1	1
19	1	1	1	1
20	1	1	1	1

Як можна зрозуміти з результатів генерацій другого етапу другого тесту з таблиці 4.6, наявність одного правила обов’язкового об’єкта не змінює стабільність розташування всіх вказаних об’єктів в згенерованому інтер’єрі.

Виходячи з цього, а також результатів першого та другого етапів першого тесту, можна визначити, що за відсутності додаткових правил розміщення і невеликого набору об’єктів інтер’єру — генерація залишатиметься стабільною навіть за наявності одного правила обов’язкового об’єкта.

На рисунках 4.13 та 4.14 можна побачити результати генерації з номерами 10 та 20 другого етапу другого тесту відповідно.

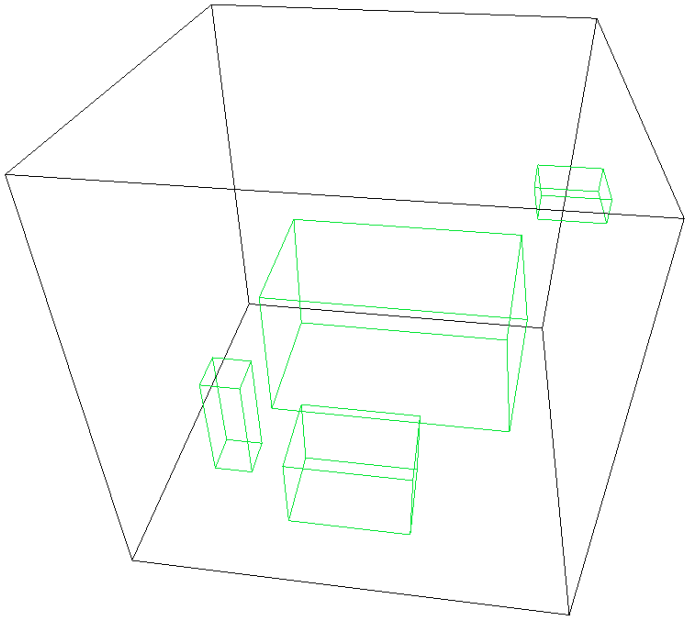


Рис. 4.14 Згенерований інтер’єр номер 10 другого етапу другого тесту

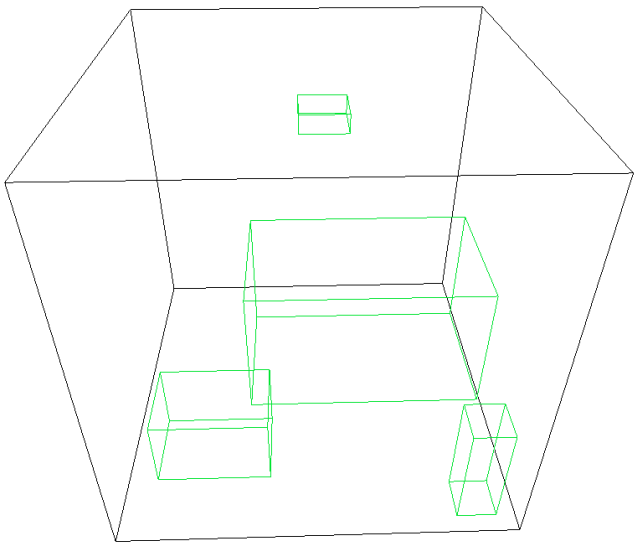


Рис. 4.15 Згенерований інтер’єр номер 20 другого етапу другого тесту

Для третього етапу тестування в два рази було збільшено максимальну кількість екземплярів для об’єктів інтер’єру. Це дозволить перевірити, наскільки генерація інтер’єру залишатиметься стабільною з більшою кількістю об’єктів інтер’єру і тільки одним правилом обов’язкого об’єкта.

Таблиця 4.7

Результати генерації 20 інтер’єрів для третього етапу другого тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	1	1	2	1
2	1	1	1	2
3	1	1	2	1
4	1	1	2	2
5	1	1	1	2
6	1	2	1	1

Продовження таблиці 4.7

Результати генерації 20 інтер’єрів для третього етапу другого тесту

7	1	2	1	1
8	1	1	2	2
9	1	2	2	1
10	1	2	2	1
11	1	2	1	1
12	1	1	1	1
13	1	2	2	1
14	1	1	2	1
15	1	2	2	2
16	1	1	1	2
17	1	2	1	2
18	1	1	1	2
19	1	2	2	1
20	1	2	2	1

По результатам 20 генерацій третього етапу другого тесту, що представлені в таблиці 4.7, можемо бачити, що кількість об’єктів в середньому коливається від одного до двох, що є очікуваним результатом для об’єктів з максимальною кількістю екземплярів, що дорівнює двум.

Це вказує на те, що навіть при збільшенні кількості можливих екземплярів об’єктів, наявність тільки одного обов’язкового об’єкта не змінює стабільності генерації інтер’єру.

Відображення генерацій 10 та 20 третього етапу другого тесту можна зображені на рисунках 4.16 та 4.17 відповідно.

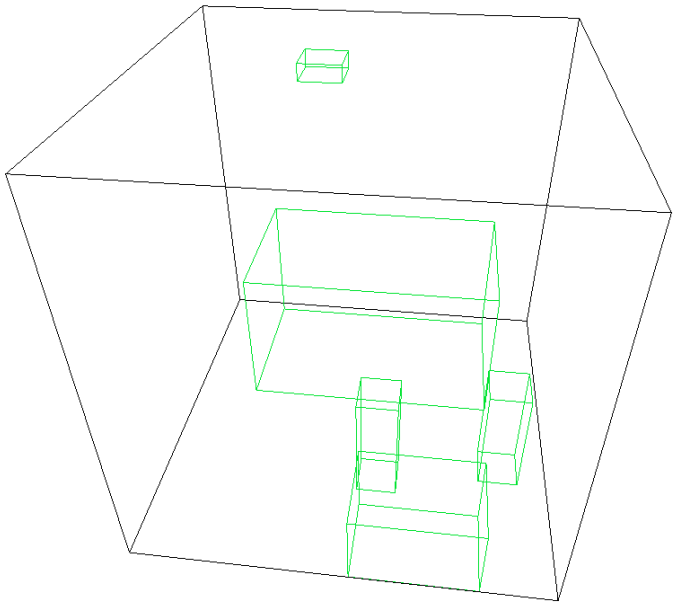


Рис. 4.16 Згенерований інтер’єр номер 10 третього етапу другого тесту

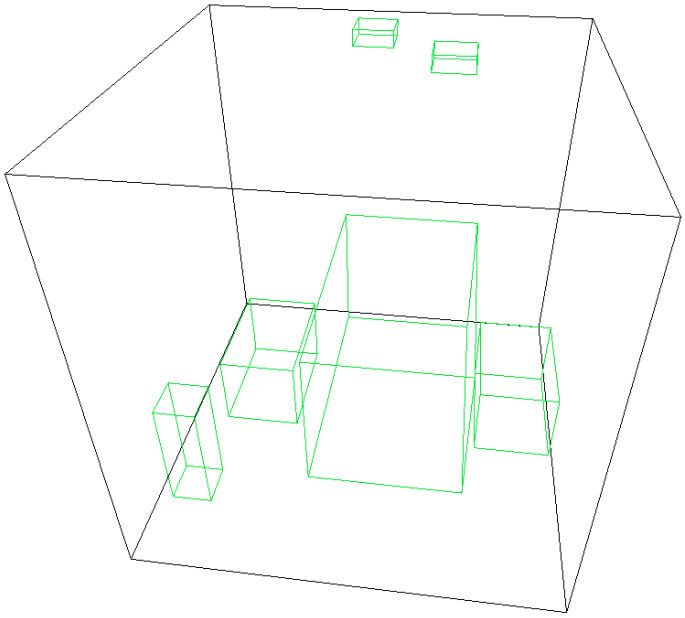


Рис. 4.17 Згенерований інтер’єр номер 20 третього етапу другого тесту

В конфігурацію четвертого етапу другого тесту було додано ще одне правило обов’язкого об’єкта, повторно для об’єкта «sofa».

На базі результатів цього етапу стане зрозуміло, наскільки перехід від одного правила обов’язкового об’єкта до двох правил обов’язкових об’єктів може змінити успішність генерацій інтер’єрів.

Таблиця 4.8

Результати генерації 20 інтер’єрів для четвертого етапу другого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	«sofa»				
2	«sofa»				
3	«sofa»				
4	«sofa»				
5	«sofa»				
6	«sofa»				
7	«sofa»				
8	«sofa»				
9		2	1	0	1
10	«sofa»				
11		2	1	0	2
12	«sofa»				
13	«sofa»				
14	«sofa»				
15		2	2	0	1
16		2	1	1	2
17		2	1	1	2
18		2	1	1	2
19		2	2	0	1
20		2	2	0	1

Як видно з даних про генерації для четвертого етапу другого тексту, наявність другого обов’язкового об’єкта суттєво погіршує шанси на успішну генерацію інтер’єру. З двадцяти проведених генерацій інтер’єрів тільки вісім були успішними.

Це пов’язано з тим, що об’єкт «sofa» є найбільш габаритним з усіх представлених і з урахуванням співвідношення розміру кімнати до кількості сконфігурованих спроб для цього об’єкта, другому обов’язковому об’єкту «sofa» не завжди залишається достатньо місця для успішного розміщення.

Також треба звернути увагу на те, що через велику габаритність обох обов’язкових об’єктів, місця для об’єкту «table» майже ніколи не залишалось і об’єкт не додавався до інтер’єру.

На рисунках 4.18 та 4.19 зображені два успішно згенерованих інтер’єра під номерами 9 та 20 відповідно.

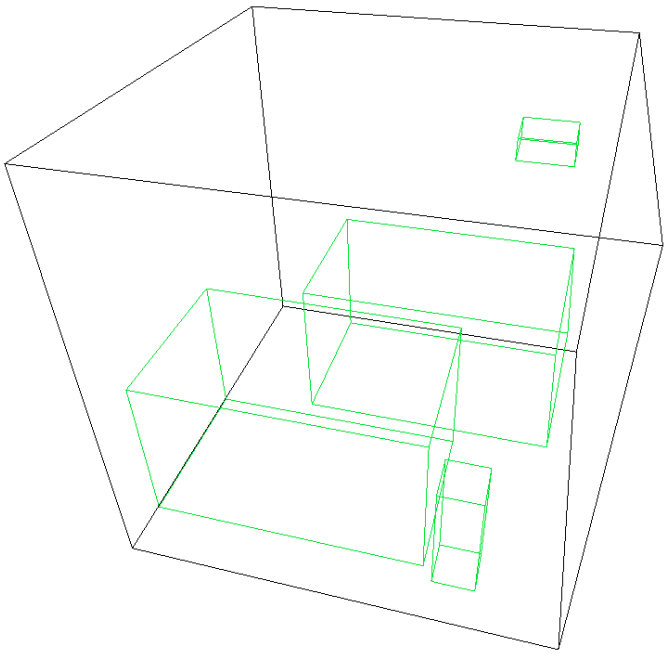


Рис. 4.18 Згенерований інтер’єр номер 9 четвертого етапу другого тесту

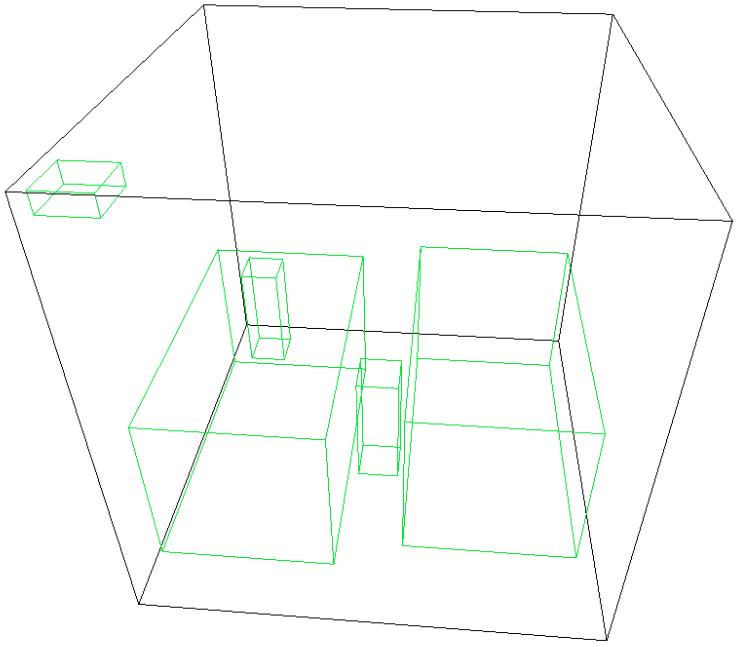


Рис. 4.19 Згенерований інтер’єр номер 20 четвертого етапу другого тесту

Для останнього, п’ятого етапу другого тесту кількість правил обов’язкових об’єктів була збільшена до трьох: 2 правила про обов’язковий об’єкт «sofa» і одне правило про обов’язковий об’єкт «land lamp».

Таблиця 4.9

Результати генерації 20 інтер’єрів для п’ятого етапу другого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
1	«sofa»				
2	«sofa»				
3		2	1	0	2
4	«sofa»				
5	«sofa»				
6	«sofa»				
7		2	1	0	2

Продовження таблиці 4.9

Результати генерації 20 інтер’єрів для п’ятого етапу другого тесту

№ Генерації	Не вийшло розмістити	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «table»	Кількість об’єктів «lamp»
8		2	1	0	1
9	«sofa»				
10		2	1	0	2
11	«sofa»				
12	«sofa»				
13	«sofa»				
14	«sofa»				
15	«land lamp»				
16	«sofa»				
17		2	1	0	2
18	«sofa»				
19	«sofa»				
20	«land lamp»				

Результати генерації інтер’єрів п’ятого етапу другого тесту не є несподіваними.

Наявність трьох обов’язкових об’єктів з доволі невеликим значенням кількості спроб та великою площою заповнення відносно кімнати має зовсім невеликі шанси на успішну генерацію інтер’єру.

Дві успішні генерації з номерами 10 та 17 відображені на рисунках 4.20 та 4.21 відповідно.

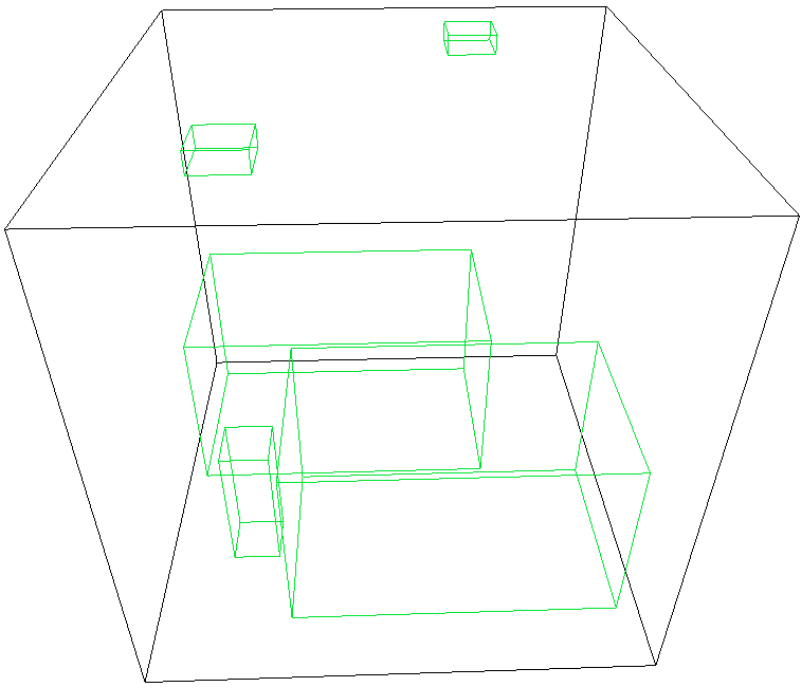


Рис. 4.20 Згенерований інтер’єр номер 10 п’ятого етапу другого тесту

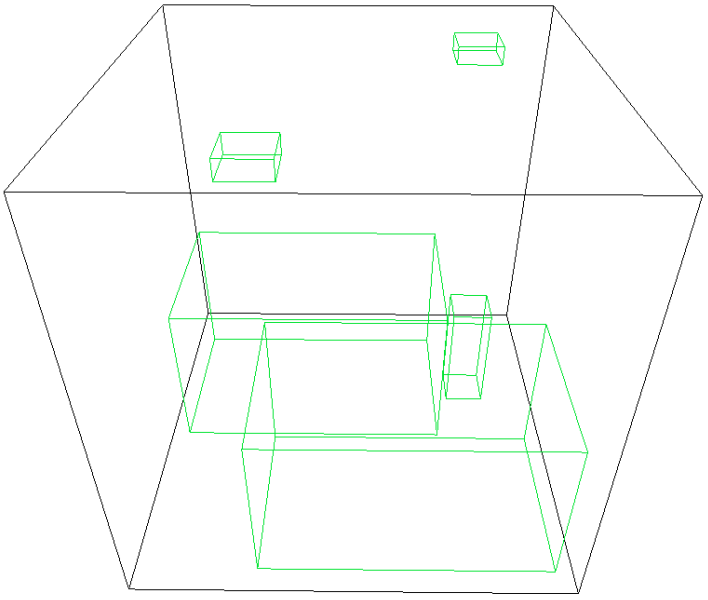


Рис. 4.21 Згенерований інтер’єр номер 17 п’ятого етапу другого тесту

По результатам всіх етапів другого тесту стає зрозуміло, що навіть при збільшенні можливих екземплярів об’єктів, основним рушієм успішності генерації інтер’єру все ще залишається конфігурація обов’язкових об’єктів, підтверджуючи твердження аналізу результатів першого тесту.

Третій тест сфокусований на проведення дослідження різних конфігурацій правил розміщення. Початкову конфігурацію третього тесту зображено на рисунку 4.22.

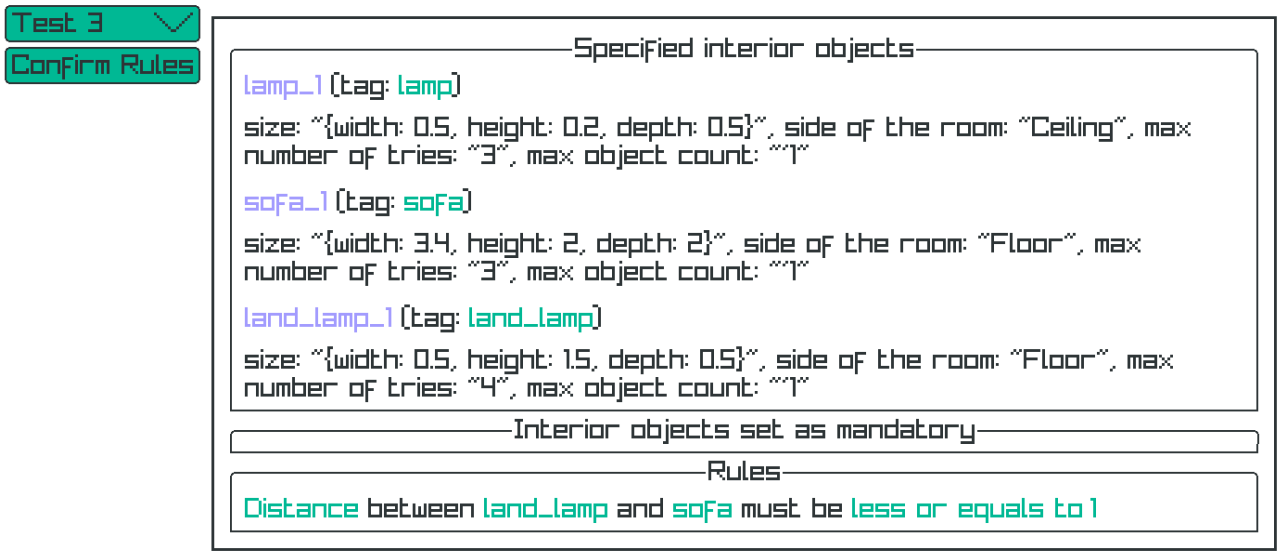


Рис. 4.22 Початкова конфігурація першого етапу третього тесту

На першому етапі третього тесту генерація буде сконфігурована з використанням трьох об’єктів інтер’єру та одного правила, яке обмежує один з цих об’єктів.

Таблиця 4.10

Результати генерації 20 інтер’єрів для першого етапу третього тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «lamp»
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1

Продовження таблиці 4.10

Результати генерації 20 інтер’єрів для першого етапу третього тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «lamp»
5	1	1	1
6	1	1	1
7	1	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
18	1	1	1
19	1	1	1
20	1	1	1

Через те, що об’єкти інтер’єру додаються в випадковому порядку, можуть виникати ситуації, за яких при додаванні суб’єкта обмеженого правилом, відповідний об’єкт за яким вказано обмеження — ще не був доданий до інтер’єру на цей момент, тому правило не перевіряється.

Окрім цього, невелика кількість об’єктів дає змогу генерації залишатися стабільною, як і продемонстровано в таблиці 4.10.

Відображення генерацій з номерами 10 та 20 зображено на рисунках 4.23 та 4.24 відповідно.

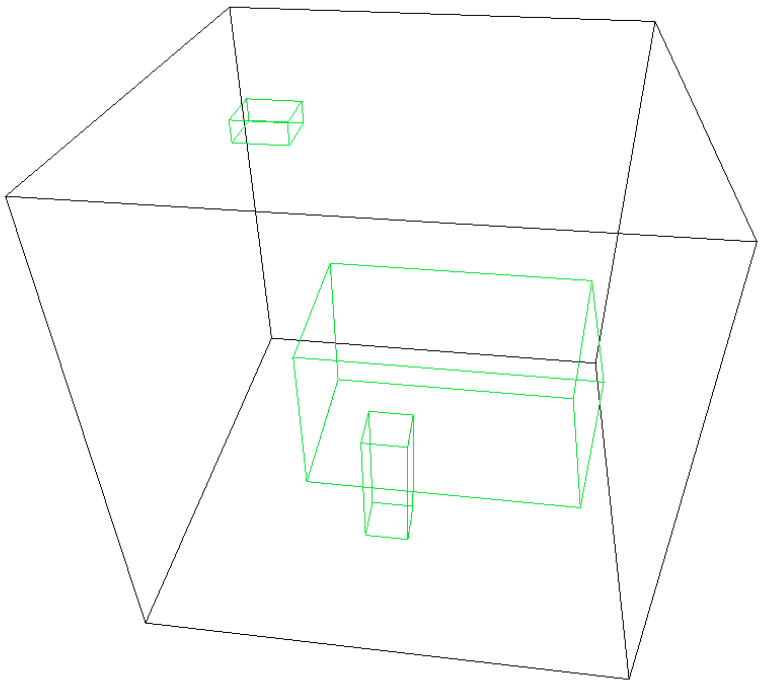


Рис. 4.23 Згенерований інтер’єр номер 10 першого етапу третього тесту

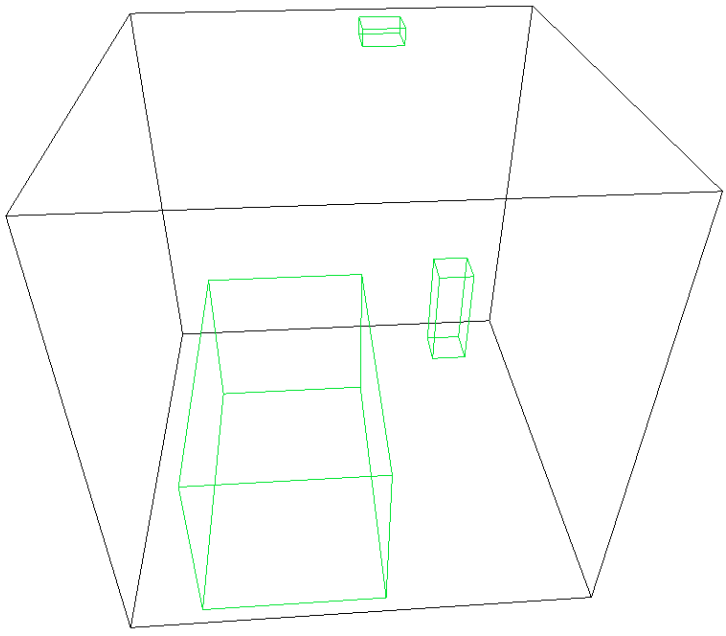


Рис. 4.24 Згенерований інтер’єр номер 20 першого етапу третього тесту

Для другого етапу третього тесту до конфігурації було додано ще одне правило обмеження відстані, між об’єктами «lamp» та «land_lamp». Таким чином список конфігурації другого етапу складається з правила обмеження відстані між «land_lamp» та «sofa» і правила обмеження відстані між «lamp» та «land_lamp».

Таблиця 4.11

Результати генерації 20 інтер’єрів для другого етапу третього тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «lamp»
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	0	1	1
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	1	1
16	1	1	1
17	1	1	1
18	1	1	1
19	0	1	1
20	1	1	1

З наявних даних в таблиці 4.11 бачимо, що хоча в середньому генерація інтер’єрів залишалась стабільною, в генераціях з номерами 7 та 19 був відсутній об’єкт «sofa». Це прямий наслідок обмеження правилом, а також недостатньо велика кількість спроб, яка сконфігурована для цього об’єкта, враховуючи відношення розміру об’єкта до розміру кімнати і обмеження правилом.

Успішні генерації з номерами 10 та 20 відображені на рисунках 4.25 та 4.26.

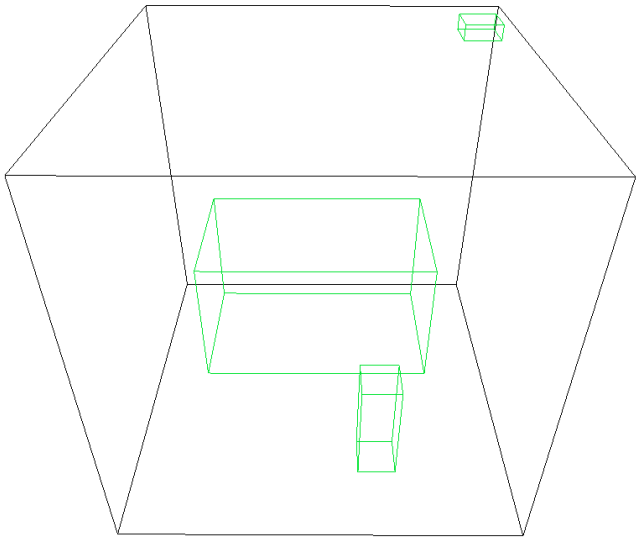


Рис. 4.25 Згенерований інтер’єр номер 10 другого етапу третього тесту

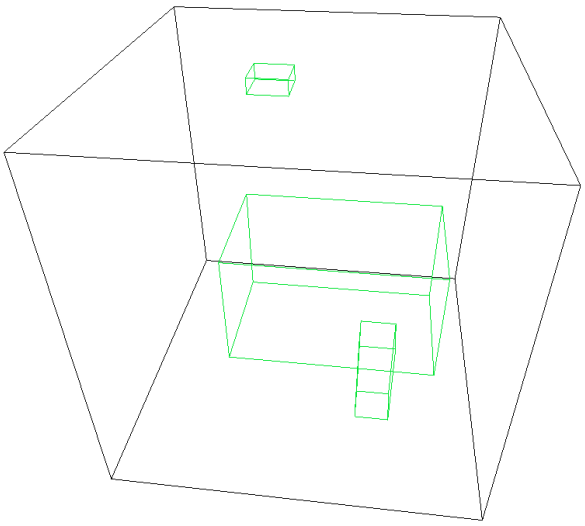


Рис. 4.26 Згенерований інтер’єр номер 20 другого етапу третього тесту

Для третього етапу третього тесту максимальна кількість екземплярів об’єктів була збільшена до двох. За подібної конфігурації буде можливість краще проаналізувати вплив двох правил розміщення на більш натуральну генерацію інтер’єру.

Таблиця 4.12

Результати генерації 20 інтер’єрів для третього етапу третього тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «lamp»
1	1	2	1
2	1	2	1
3	1	1	1
4	1	2	1
5	1	2	2
6	1	2	2
7	1	2	2
8	1	1	1
9	1	1	2
10	1	2	1
11	1	2	1
12	2	1	1
13	1	1	2
14	2	1	2
15	1	2	1
16	1	2	2
17	1	1	2
18	1	1	1
19	1	1	2
20	1	2	2

З результатів генерації 20 інтер’єрів для третього етапу третього тесту, дані про які відображені в таблиці 4.12, бачимо стабільну генерацію інтер’єру в рамках очікуваних кількостей об’єктів.

Успішні генерації 10 та 20 відображені на рисунках 4.27 та 4.28 відповідно.

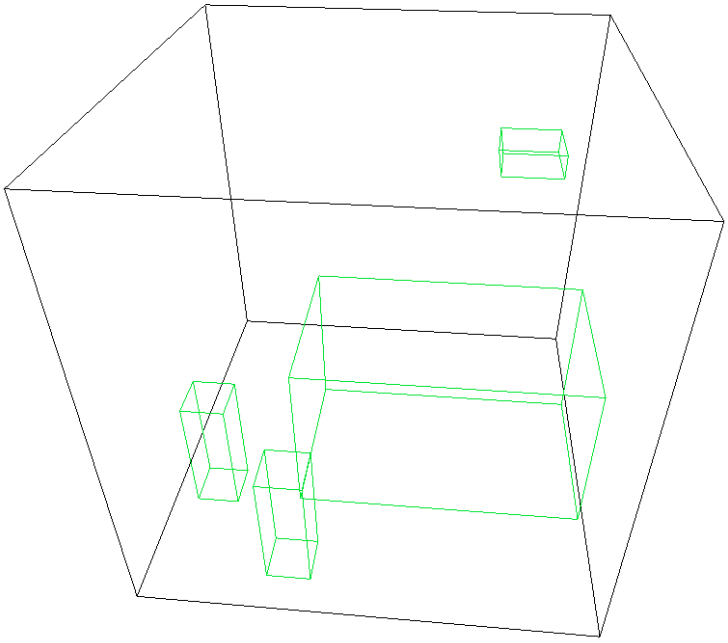


Рис. 4.27 Згенерований інтер’єр номер 10 третього етапу третього тесту

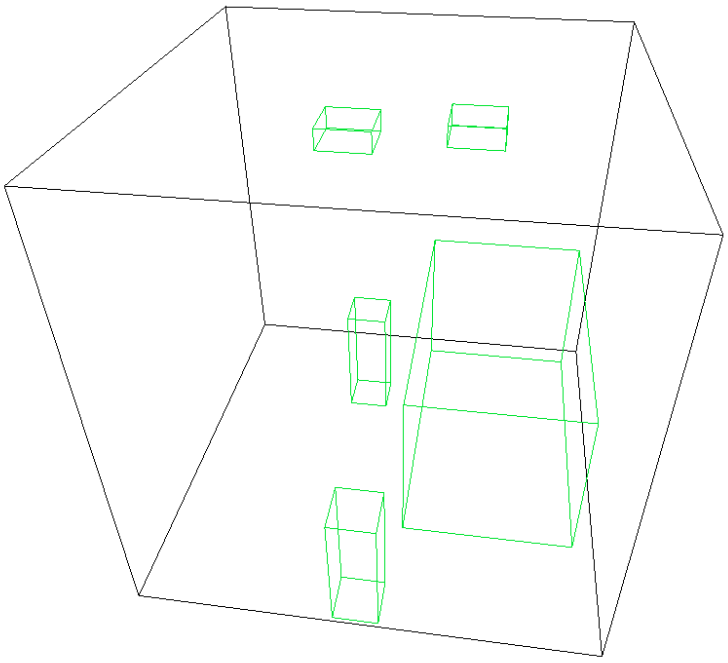


Рис. 4.28 Згенерований інтер’єр номер 20 третього етапу третього тесту

Для четвертого етапу третього тесту до конфігурації було додано третє правило обмеження відстані — між об’єктами «sofa» та «lamp».

Таблиця 4.13

Результати генерації 20 інтер’єрів для четвертого етапу третього тесту

№ Генерації	Кількість об’єктів «sofa»	Кількість об’єктів «land lamp»	Кількість об’єктів «lamp»
1	1	1	1
2	2	1	2
3	1	2	1
4	1	1	2
5	1	2	1
6	1	1	1
7	1	1	1
8	2	1	2
9	1	1	2
10	0	2	1
11	1	1	2
12	1	2	2
13	1	1	1
14	1	1	1
15	1	1	1
16	1	2	1
17	1	1	2
18	2	2	2
19	1	2	2
20	1	2	1

Виходячи з даних в таблиці 4.13, можна визнати, що поточна конфігурація зберігає стабільність генерації навіть за наявності трьох правил обмеження відстані між об’єктами. З двадцяти проведених тестових генерацій тільки в одній генерації був відсутній сконфігурований об’єкт - «sofa».

Це вказує на те, що вказана кількість екземплярів є достатньою з врахуванням кількості об’єктів для генерації інтер’єру, відношення розмірів об’єктів до розміру кімнати та кількості правил є збалансованою. Генерації 10 та 20 зображені на рисунках 4.29 та 4.30 відповідно.

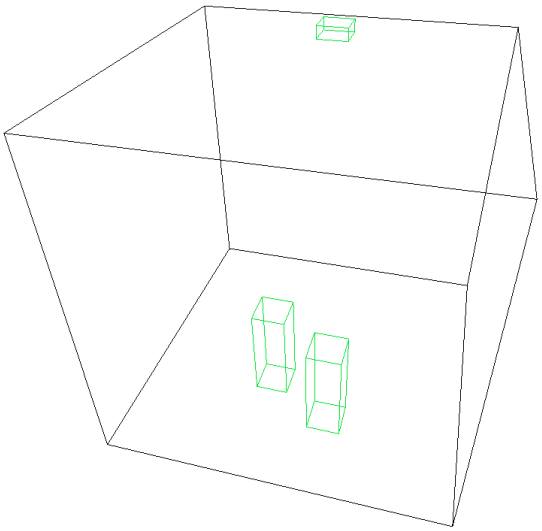


Рис. 4.29 Згенерований інтер’єр номер 10 четвертого етапу третього тесту

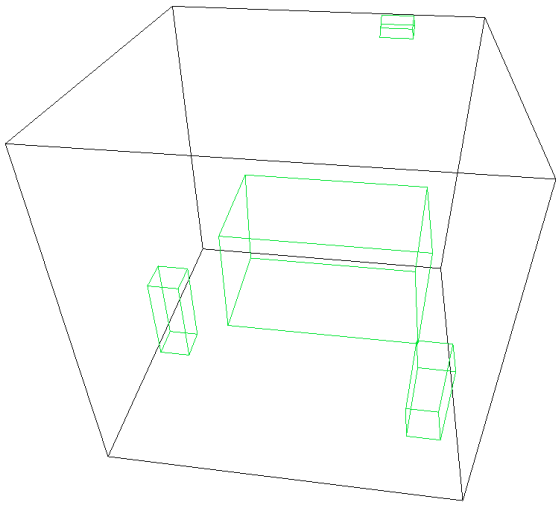


Рис. 4.30 Згенерований інтер’єр номер 20 четвертого етапу третього тесту

Підсумовуючи аналіз етапів третього тесту можна прийти до висновку, що без вказання правил, конфігурації для генерацій інтер’єрів навіть з об’єктами з вказаними кількостями екземплярів як одиниця можуть бути стабільними. Тим не менш, збільшення кількості екземплярів особливо важливе у випадку конфігурування з використанням правил.

4.2 Оцінка ефективності та зручності використання

Розроблена система для генерації інтер’єрів з використанням спрощених моделей об’єктів інтер’єру під назвою «Furnugen» має базис для можливості оптимального конфігурування генерацій інтер’єрів.

Оптимізований підхід до створення генерації дає змогу використати потужності програмних засобів для перевірок логічного співвідношення можливих варіантів розміщення об’єктів за допомогою описаних правил, які формують важливі логічні зв’язки елементів інтер’єру для задоволення потреб користувача.

Тим не менш, для можливості широкого використання мають бути доповнені можливості для можливості створення наборів конфігурацій без зміни програмного коду, можливість завантажувати чи підвантажувати файли з вже сконфігурованими даними.

Також незамінним функціоналом для користувачів може стати можливість експортування готового макету інтер’єру в поширених форматах трьохвимірних сцен.

ВИСНОВКИ

У межах даної магістерської роботи було розглянуто проблему оптимізації початкових етапів проєктування інтер'єрів, а саме процесу створення попередніх скетчів і планувальних заготовок, які традиційно виконуються дизайнерами вручну або з використанням загальних інструментів комп'ютерної графіки. Проведений аналіз робочих підходів фахівців у сфері дизайну інтер'єру показав, що на етапі формування первинної концепції значна частина часу витрачається не на творчу складову, а на рутинні операції з розміщення типових об'єктів, перевірку їх сумісності, габаритів та взаємних відстаней. Це створює передумови для автоматизації зазначеного етапу з використанням алгоритмічних методів генерації.

На основі результатів дослідження було обґрунтовано доцільність застосування спрощених моделей об'єктів для автоматичної генерації інтер'єрів. В рамках кваліфікаційної роботи розроблено програмний модуль, що використовує паралелепіпеди як абстракції реальних об'єктів інтер'єру, що дозволяє суттєво зменшити обчислювальну складність перевірок перетину та просторових обмежень. Такий підхід є виправданим для стадії попереднього проєктування, де важливішими є загальні пропорції та функціональне зонування, ніж точна геометрична деталізація.

Важливою особливістю розробленого модуля є використання системи спеціалізованих правил, які задаються користувачем. Ці правила дозволяють обмежувати кількість об'єктів у сцені, визначати допустимі відстані між ними, а також накладати обмеження на їхні розміри. Такий механізм забезпечує гнучкість алгоритму генерації та адаптацію до різних типів приміщень і дизайнерських вимог. Користувач має змогу змінювати правила розміщення об'єктів інтер'єру, що дозволяє формалізувати власний досвід і професійні навичка у вигляді набору параметрів та обмежень.

У процесі виконання роботи було проведено серію тестових досліджень з різними конфігураціями вхідних даних. Зокрема, було виконано перевірки генерації інтер'єрів з різною кількістю обов'язкових об'єктів. Результати

показали, що зі збільшенням кількості обов’язкових об’єктів складність задачі зростає нелінійно, що підтверджує важливість коректного підбору правил та параметрів генерації.

Окрему групу експериментів було присвячено аналізу впливу загальної кількості об’єктів інтер’єру, включаючи як обов’язкові, так і необов’язкові об’єкти. У ході цих тестів було досліджено, яка частина об’єктів не додається до фінальної сцени через просторові або правилові обмеження. Отримані результати дозволили зробити висновок, що за при правильному підборі параметрів під конкретну задачу генерації інтер’єру дає можливість працювати з стабільною генерацією інтер’єру, яка на максимум реалізує задані обмеження, залишаючи найбільшу кількість об’єктів для наповнення.

Третій набір перевірок був спрямовані на оцінку впливу різних наборів правил на ефективність генерації для різної кількості об’єктів. Порівняльний аналіз показав, що оптимально сконфігуровані параметри інформації про об’єкти дозволяють значно зменшити кількість невдалих спроб генерації та скоротити час пошуку допустимого розміщення. Це підтверджує гіпотезу про те, що якість результату генеративного алгоритму значною мірою залежить не лише від самої реалізації, а й від коректності заданої кофнігурації генерації інтер’єру.

Розроблений програмний продукт, який зумовлений зростаючим попитом на інструменти, що поєднують автоматизацію та інтерактивність у сфері проєктування інтер’єрів. Запропонований підхід може бути використаний як допоміжний засіб для дизайнерів на ранніх етапах роботи, а також як основа для подальшого розвитку систем процедурної генерації інтер’єрів у комп’ютерних іграх, віртуальній та доповненій реальності, а також у системах швидкого прототипування житлових і комерційних приміщень.

Підсумовуючи, можна стверджувати, що поставлені завдання до оптимізації генерації спрощених моделей об’єктів інтер’єру було виконано, а мету дослідження — досягнуто. Розроблений програмний модуль демонструє потенціал для можливості широкого використання для гнучкої та швидкої генерації інтер’єрів.

Результати дослідження апробовано та опубліковано у наступних тезах доповіді на конференціях:

1. Сергієнко К.В., Золотухіна О.А. Процес створення плану інтер’єру та можливі вирішення загальних проблем програмним шляхом. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.528-530.

2. Сергієнко К.В., Золотухіна О.А. Огляд програмних засобів, які використовують для створення дизайнів інтер’єрів. Всеукраїнська науково-технічна конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.293-294.

ПЕРЕЛІК ПОСИЛАНЬ

1. Швець, Олена & Коломієць, Дмитро & Бабчук, Юрій. (2025). Синтез мистецтва і дизайну інтер'єрів. Synthesis of art and interior design. Мистецтво в культурі сучасності: теорія та практика навчання. 8-16. 10.31652/3041-1017-2024(4)-01.

2. Andrews G. Cellular automata and applications. *Whitman College*. URL: https://www.whitman.edu/documents/Academics/Mathematics/andrewgw.pdf (дата звернення: 13.10.2025).

3. Antoniuk, Izabella. (2023). Generating layout for complex cave-like levels with schematic maps and Cellular Automata. Machine Graphics and Vision. 32. 45-65. 10.22630/MGV.2023.32.2.3.

4. Bellotti, Francesco & Berta, Riccardo & Cardona, Rosario & De Gloria, Alessandro. (2011). An architectural approach to efficient 3D urban modeling. Computers & Graphics. 35. 1001-1012. 10.1016/j.cag.2011.07.004.

5. Bradecki, Tomasz & Uherek-Bradecka, Barbara. (2021). Work Models in the Design Process for House Interior and Exterior: Physical or Virtual? Related content Models for Experimental High Density Housing Work Models in the Design Process for House Interior and Exterior: Physical or Virtual?. IOP Conference Series Materials Science and Engineering.

6. Chu, Xiaoyu & Xu, Rui & Wang, Guangjun. (2024). Design and Application of Modern Interior Design Style System Based on Unity3D. Journal of Electronic Research and Application. 8. 46-51. 10.26689/jera.v8i6.8569.

7. Fahmy, Hoda & Blostein, D.. (1992). A survey of graph grammars: theory and applications. 294 - 298. 10.1109/ICPR.1992.201776.

8. Laitaruk I. F., Hryshanovych T. O. Overview of modern algorithms for world procedural generation in computer games. *CEUR workshop proceedings*. 2024. С. 152–163. URL: https://cssesw.easyscience.education/cssesw2024/CSSESW2024/paper21.pdf (дата звернення: 13.10.2025).

9. Mahendarto, Trias. (2025). History of Artificial Intelligence and Its Potential to Revolutionising the Interior Design Industry. Journal of Artificial Intelligence in Architecture. 4. 1-13. 10.24002/jarina.v4i1.9111.

10. Merrell, Paul & Manocha, Dinesh. (2009). Constraint-based model synthesis. Proceedings - SPM 2009: SIAM/ACM Joint Conference on Geometric and Physical Modeling. 101-111. 10.1145/1629255.1629269.

11. Metropolis, N. et al. (1953). Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics. Vol. 21(6). P. 1087–1092. DOI: 10.1063/1.1699114.

12. Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains. Biometrika. Vol. 57(1). P. 97–109. DOI: 10.1093/biomet/57.1.97.

13. Ross, S. M. (2014). Introduction to Probability Models. 11th ed. Academic Press. ISBN: 978-0124079489.

14. Dechter, R. (2003). Constraint Processing. Morgan Kaufmann. ISBN: 978-1558608900.

15. Barták, R. (1999). Constraint programming: In pursuit of the holy grail. Proceedings of WDS. P. 555–564.

16. Shen, Zhenyuan. (2022). Procedural Generation in Games: Focusing on Dungeons. SHS Web of Conferences. 144. 02005. 10.1051/shsconf/202214402005.

17. Song, Peihua & Zheng, Youyi & Jia, Jinyuan & Gao, Yan. (2019). Web3D-based automatic furniture layout system using recursive case-based reasoning and floor field. Multimedia Tools and Applications. 78. 10.1007/s11042-018-6334-5.

18. Yu, Lap-Fai & Yeung, Sai & Tang, Chi-Keung & Terzopoulos, Demetri & Chan, Tony & Osher, Stanley. (2011). Make it Home: Automatic Optimization of Furniture Arrangement. ACM Trans. Graph.. 30. 86. 10.1145/2010324.1964981.

19. Smelik, R. M. et al. (2014). A survey on procedural modeling for virtual worlds. IEEE Computer Graphics and Applications. Vol. 34(6). P. 31–50. DOI: 10.1109/MCG.2014.42.

20. Van der Linden, R. et al. (2013). Procedural generation of dungeons. IEEE TCIAIG. Vol. 6(1). P. 78–89. DOI: 10.1109/TCIAIG.2013.2290372.

21. Wonka, P. et al. (2003). Instant Architecture. ACM Transactions on Graphics. Vol. 22(3). P. 669–677. DOI: 10.1145/882262.882324.

22. Parish, Y. & Müller, P. (2001). Procedural Modeling of Cities. SIGGRAPH. P. 301–308. DOI: 10.1145/383259.383292.

23. Müller, P. et al. (2006). Procedural Modeling of Buildings. ACM Transactions on Graphics. Vol. 25(3). P. 614–623. DOI: 10.1145/1141911.1141931.

24. Eastman, C. et al. (2011). BIM Handbook. 2nd ed. Wiley. ISBN: 978-0470541371.

25. Oxman, R. (2017). Thinking difference: Theories and models of parametric design thinking. Design Studies. Vol. 52. P. 4–39. DOI: 10.1016/j.destud.2017.06.001.

26. Kolarevic, B. (2003). Architecture in the Digital Age. Taylor & Francis. ISBN: 978-0415278092.

27. Salvatori. The 4 key phases of an interior design project. URL: https://www.salvatoriofficial.com/en/gb/stories/the-4-key-phases-of-an-interior-design-project/ (дата звернення: 13.10.2025).

28. Occa Design Studio. Work stages in interior design. URL: https://occa-design.com/service/interior-design/work-stages/ (дата звернення: 13.10.2025).

29. Cormen, T. et al. (2022). Introduction to Algorithms. 4th ed. MIT Press. ISBN: 978-0262046305.

30. Sedgewick, R. & Wayne, K. (2011). Algorithms. 4th ed. Addison-Wesley. ISBN: 978-0321573513.

31. Knuth, D. E. (1998). The Art of Computer Programming, Vol. 1. 3rd ed. Addison-Wesley. ISBN: 978-0201896831.

32. Bentley, J. (1986). Programming Pearls. Communications of the ACM. Vol. 29(9). P. 858–866.

33. Tarjan, R. (1983). Data Structures and Network Algorithms. SIAM. ISBN: 978-0898711875.

34. Gamma, E. et al. (1994). Design Patterns. Addison-Wesley. ISBN: 978-0201633610.

35. Nystrom, R. (2014). Game Programming Patterns. Genever Benning. ISBN: 978-

0990582908.

36. Brooks, F. P. (1995). The Mythical Man-Month. Addison-Wesley. ISBN: 978-0201835953.

37. Eberly, David H. (2001). 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics. Morgan Kaufmann. ISBN: 978-1558605930.

38. Ericson, Christer. (2005). Real-Time Collision Detection. Morgan Kaufmann. ISBN: 978-1558607323.

39. Lengyel, Eric. (2012). Mathematics for 3D Game Programming and Computer Graphics. 3rd ed. Cengage Learning. ISBN: 978-1435458864.

40. de Berg, Mark & Cheong, Otfried & van Kreveld, Marc & Overmars, Mark. (2008). Computational Geometry: Algorithms and Applications. 3rd ed. Springer. ISBN: 978-3540779735.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«Оптимізація розміщення спрощених моделей об’єктів інтер’єру»

Виконав: студент групи ПДМ-61 Кирило СЕРГІЄНКО

Керівник: канд. техн. наук, доцент кафедри ІПЗ Оксана ЗОЛОТУХІНА

Київ - 2025

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: оптимізація розміщення моделей об’єктів інтер’єру за рахунок використання спрощених геометричних фігур

Об’єкт дослідження: розміщення спрощених моделей об’єктів інтер’єру.

Предмет дослідження: метод оптимізації розміщення спрощених моделей об’єктів інтер’єру.

АКТУАЛЬНІСТЬ РОБОТИ

Назва методу	Переваги	Недоліки
Метод рекурсивних міркувань на основі прецедентів та поля підлоги	1. Метод підтримує плани приміщень нестандартної форми 2. Можуть враховуватися взаємопов'язаності об'єктів інтер'єру 3. Враховується ергономіка розміщення	1. Фіксовані правила взаємопов'язаності об'єктів інтер'єру 2. Неможливість редагування/зміни правила розміщення об'єктів
Ітеративне створення інтер'єру на основі бібліотеки прикладів та додаткових обмежень	1. Метод підтримує складні взаємозв'язки об'єктів інтер'єру 2. Враховується ергономіка розміщення 3. Велика кількість ітеративних спроб розміщення для максимально можливого врахування всіх обмежень	1. Неможливість редагування/зміни правила розміщення об'єктів 2. Фіксовані правила взаємопов'язаності об'єктів інтер'єру 3. Повільна генерація через складність правил розміщення

3

ОПИС ПРАВИЛ РОЗМІЩЕННЯ ОБ'ЄКТІВ ІНТЕР'ЄРУ

Звичайне правило: <Суб'єкт> <Відношення> {<Відстань> <Об'єкт>|<Розмір> <Вісь> <Об'єкт>|<Розмір> <Вісь> <Скаляр>}

Правило обов'язкового об'єкта: <Суб'єкт>

Суб'єкт: елемент масиву сконфігурованих об'єктів інтер'єру, розміщення якого обмежуються правилом

Об'єкт: вже доданий до інтер'єру об'єкт

Відношення: вказівник відношення наступного специфікатора відносно відповідного значення суб'єкта {==, <, >, >=, <=}

Відстань: відстань можливої позиції суб'єкта до позиції об'єкта в метрах

Розмір: розмір суб'єкта відносно наступного специфікатора

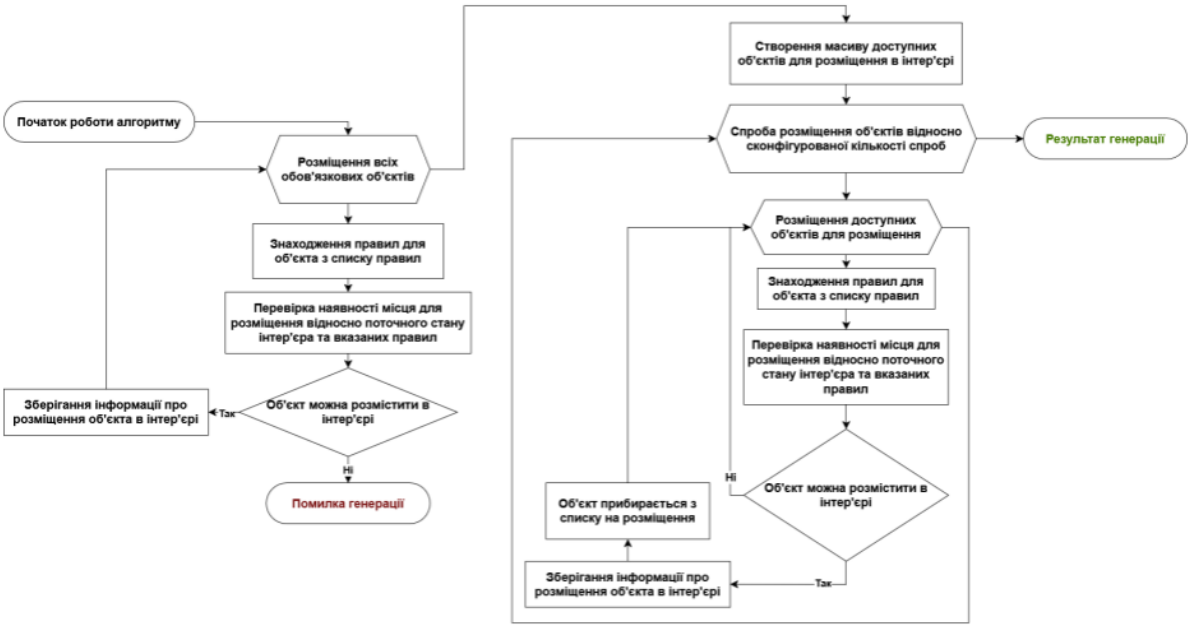
Вісь: задає напрям вимірювання розміру

Кількість: вказана кількість суб'єктів до розміщення

Скаляр: числова константа

4

АЛГОРИТМ ГЕНЕРАЦІЇ



5

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

6

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Правило	Мінімальна складність	Максимальна складність
Обов'язкового об'єкта	O(N)	O(tn * N)
Звичайне відносно іншого об'єкта	O(oN * N)	O(oN * tn * N * gn)
Звичайне відносно константи	O(N)	O(tn * N * gn)

Де:
N - кількість вже доданих до інтер'єру об'єктів
aoN - кількість вже доданих до інтер'єру обов'язкових об'єктів
oN - кількість вже доданих до інтер'єру об'єктів, відносно яких вказане обмеження

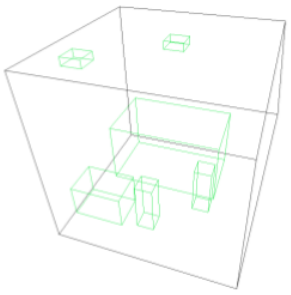
tn - сконфігурована кількість спроб для розміщення об'єкту
gn - сконфігурована кількість поколінь спроб розміщення об'єктів

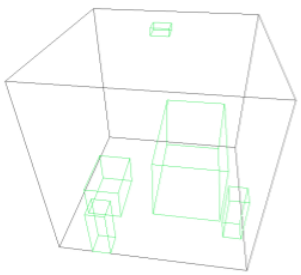
7

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Тестування впливу кількості обов'язкових об'єктів на результати генерації інтер'єру

Етап тестування	Кількість обов'язкових об'єктів	Відсоток відношення максимальної кількості об'єктів до фактичної	Відсоток успішних генерацій
Перший	1	66%	100%
Другий	2	62%	100%
Третій	3	36%	60%
Четвертий	4	40%	55%



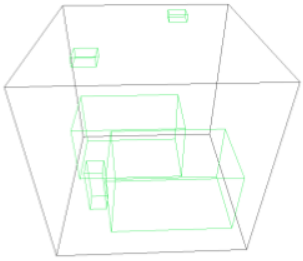


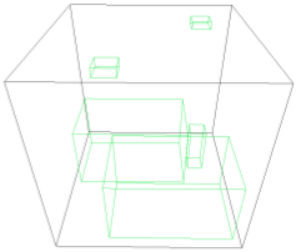
8

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Тестування різних комбінацій наборів обов'язкових та звичайних об'єктів інтер'єру

Етап тестування	Кількість обов'язкових об'єктів	Відсоток відношення максимальної кількості об'єктів до фактичної	Відсоток успішних генерацій
Перший	0	100%	100%
Другий	1	100%	100%
Третій	1	66%	100%
Четвертий	2	21%	40%
П'ятий	3	12%	25%



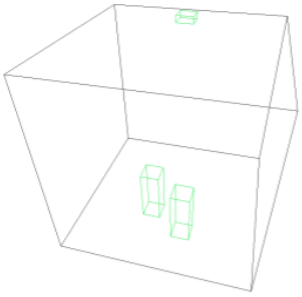


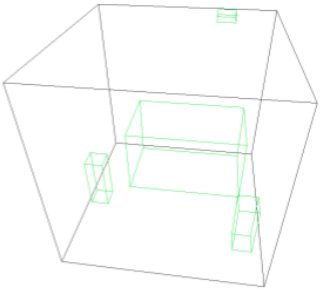
9

РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Тестування впливу кількості правил обмеження розміщення на результати генерації інтер'єрів

Етап тестування	Кількість правил обмеження відстані	Відсоток відношення максимальної кількості об'єктів до фактичної	Відсоток успішних генерацій
Перший	1	100%	100%
Другий	2	97%	100%
Третій	2	69%	100%
Четвертий	3	70%	100%





10

ВИСНОВКИ

Дослідивши підхід дизайнерів до створення інтер'єрів, було виокремлено етап, який можна вдосконалити та оптимізувати - створення схеми інтер'єру перед додаванням конкретних деталей.

Існуючі методи дають можливість генерувати комплексні інтер' єри, але обмежені в швидкості генерації та можливостях для впливу на результат.

Представлений метод оптимізації розміщення спрощених об'єктів інтер'єру базується на оптимізації генерації інтер'єру за рахунок використання паралелепіпедів замість складних форм кінцевих об' єктів. Такий підхід дозволяє використовувати прості методи перевірки доступних місць для розміщення об'єктів інтер'єру, значно пришвидшуючи час знаходження вільного місця.

Описавши можливі правила обмежень розміщення об'єктів та алгоритм роботи методу вдалося визначити складності операцій розміщення відносно виду правила та кількості вже розміщених об' єктів в інтер'єрі. Такий аналіз дозволяє зрозуміти, в рамках яких налаштувань генерації інтер'єрів метод є найбільш оптимальним та ефективним для використання.

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Сергієнко К.В., Золотухіна О.А. Процес створення плану інтер'єру та можливі вирішення загальних проблем програмним шляхом. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.528-530.
2. Сергієнко К.В., Золотухіна О.А. Огляд програмних засобів, які використовують для створення дизайнів інтер'єрів. Всеукраїнська науково-технічна конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.293-294.

ДОДАТОК Б. ЛІСТИНГ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

```
package furnugen

import rand "core:math/rand"
import "core:strings"
import rl "vendor:raylib"

BBox :: rl.BoundingBox

Room_Bounding_Side :: enum {
    None = 0,
    Ceiling,
    Floor,
    Wall,
    Corner,
    // Ceiling_And_Wall,
    // Floor_And_Wall,
    // Corner_And_Ceiling,
    // Corner_And_Floor,
}

Room_Wall :: enum {
    None = 0,
    Front,
    Right,
    Back,
    Left,
}

Size :: struct {
    width, height, depth: f32,
}

to_vec3 :: proc(size: Size) -> rl.Vector3 {
    return {size.width, size.height, size.depth}
}

Interior_Object_Info :: struct #all_or_none {
    name:           string,
    tag:            Object_Tag,
    size:           Size,
    room_side:      Room_Bounding_Side,
    number_of_tries: u8,
    max_population:  u8,
}

Object_Tag :: distinct string

Possible_Interior_Object :: struct {
    interior_object_info: Interior_Object_Info,
    already_tried_count:  u8,
}

Generator_Info :: struct #all_or_none {
    room_size:           Size,
    object_infos:        []Interior_Object_Info,
    object_tags:         []Object_Tag,
    mandatory_object_rules: []Mandatory_Object_Rule,
    optional_object_rules:  []Optional_Object_Rule,
}

create_generator_info :: proc(
    generation_info_kind: Program_Generation_Info_Kind,
    allocator := context.allocator,
) -> Generator_Info {
    switch generation_info_kind {
    case .Test_1:
        return create_test_1_generator_info(allocator)
    case .Test_2:
        return create_test_2_generator_info(allocator)
    case .Test_3:
        return create_test_3_generator_info(allocator)
    case .Test_4:
        return create_test_4_generator_info(allocator)
    case .Custom, .None:
        return {}
    }

    unreachable()
}
        create_test_1_generator_info :: proc(allocator :=
            context.allocator) -> Generator_Info {
    object_tags := make([dynamic]Object_Tag, allocator =
allocator)

    // object infos
    object_infos := make([dynamic]Interior_Object_Info, 0,
2, allocator = allocator)

    append(
        &object_infos,
        Interior_Object_Info {
            name = "lamp_1",
            tag = get_or_add_object_tag(&object_tags, "lamp",
allocator = allocator),
            size = {width = 0.5, height = 0.2, depth = 0.5},
            room_side = .Ceiling,
            number_of_tries = 3,
            max_population = 3,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "land_lamp_1",
            tag = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
            size = {width = 0.5, height = 1.5, depth = 0.5},
            room_side = .Floor,
```

```
            number_of_tries = 4,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "sofa_1",
            tag = get_or_add_object_tag(&object_tags, "sofa",
allocator = allocator),
            size = {width = 3.4, height = 2, depth = 2},
            room_side = .Floor,
            number_of_tries = 3,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "table_1",
            tag = get_or_add_object_tag(&object_tags, "table",
allocator = allocator),
            size = {width = 1.5, height = 1, depth = 1},
            room_side = .Floor,
            number_of_tries = 4,
            max_population = 1,
        },
    )

    // mandatory object rules
    mandatory_object_rules :=
make([dynamic]Mandatory_Object_Rule, 0, 2, allocator =
allocator)
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "sofa_1"})
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "land_lamp_1"})
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "land_lamp_1"})
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "table_1"})

    // optional rules
    optional_rules :=
make([dynamic]Optional_Object_Rule, 0, 1, allocator =
allocator)

    generator_info := Generator_Info {
        room_size            = {5, 5, 5},
        mandatory_object_rules = mandatory_object_rules[:],
        optional_object_rules  = optional_rules[:],
        object_infos         = object_infos[:],
        object_tags          = object_tags[:],
    }
    return generator_info
}

create_test_2_generator_info :: proc(allocator :=
context.allocator) -> Generator_Info {
    object_tags := make([dynamic]Object_Tag, allocator =
allocator)

    // object infos
    object_infos := make([dynamic]Interior_Object_Info, 0,
2, allocator = allocator)

    append(
        &object_infos,
        Interior_Object_Info {
            name = "lamp_1",
            tag = get_or_add_object_tag(&object_tags, "lamp",
allocator = allocator),
            size = {width = 0.5, height = 0.2, depth = 0.5},
            room_side = .Ceiling,
            number_of_tries = 3,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "land_lamp_1",
            tag = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
            size = {width = 0.5, height = 1.5, depth = 0.5},
            room_side = .Floor,
            number_of_tries = 4,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "sofa_1",
            tag = get_or_add_object_tag(&object_tags, "sofa",
allocator = allocator),
            size = {width = 3.4, height = 2, depth = 2},
            room_side = .Floor,
            number_of_tries = 3,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "table_1",
            tag = get_or_add_object_tag(&object_tags, "table",
allocator = allocator),
            size = {width = 1.5, height = 1, depth = 1},
            room_side = .Floor,
            number_of_tries = 4,
            max_population = 2,
        },
    )
```

```
    mandatory_object_rules :=
make([dynamic]Mandatory_Object_Rule, 0, 2, allocator =
allocator)
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "sofa_1"})
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "sofa_1"})
    append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "land_lamp_1"})

    optional_rules :=
make([dynamic]Optional_Object_Rule, 0, 1, allocator =
allocator)

    generator_info := Generator_Info {
        room_size             = {5, 5, 5},
        mandatory_object_rules = mandatory_object_rules[:],
        optional_object_rules  = optional_rules[:],
        object_infos          = object_infos[:],
        object_tags           = object_tags[:],
    }
    return generator_info
}

create_test_3_generator_info :: proc(allocator :=
context.allocator) -> Generator_Info {
    object_tags := make([dynamic]Object_Tag, allocator =
allocator)

    object_infos := make([dynamic]Interior_Object_Info, 0,
2, allocator = allocator)

    append(
        &object_infos,
        Interior_Object_Info {
            name = "lamp_1",
            tag = get_or_add_object_tag(&object_tags, "lamp",
allocator = allocator),
            size = {width = 0.5, height = 0.2, depth = 0.5},
            room_side = .Ceiling,
            number_of_tries = 3,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "sofa_1",
            tag = get_or_add_object_tag(&object_tags, "sofa",
allocator = allocator),
            size = {width = 3.4, height = 2, depth = 2},
            room_side = .Floor,
            number_of_tries = 3,
            max_population = 2,
        },
    )

    append(
        &object_infos,
        Interior_Object_Info {
            name = "land_lamp_1",
            tag = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
            size = {width = 0.5, height = 1.5, depth = 0.5},
            room_side = .Floor,
            number_of_tries = 4,
            max_population = 2,
        },
    )

    mandatory_object_rules :=
make([dynamic]Mandatory_Object_Rule, 0, 2, allocator =
allocator)

    optional_rules :=
make([dynamic]Optional_Object_Rule, 0, 1, allocator =
allocator)

    append(
        &optional_rules,
        Optional_Object_Rule {
            subject = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
            relation_spec = Rule_Relation_Specification {
                kind = .Less_Or_Equals_Than,
                relation = Rule_Distance_Object_Relation {
                    object = get_or_add_object_tag(&object_tags,
"sofa", allocator = allocator),
                    distance = 1,
                },
            },
        },
    )

    append(
        &optional_rules,
        Optional_Object_Rule {
            subject = get_or_add_object_tag(&object_tags,
"lamp", allocator = allocator),
            relation_spec = Rule_Relation_Specification {
                kind = .Less_Or_Equals_Than,
                relation = Rule_Distance_Object_Relation {
                    object = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
                    distance = 5,
                },
            },
        },
    )

    append(
        &optional_rules,
        Optional_Object_Rule {
            subject = get_or_add_object_tag(&object_tags,
"sofa", allocator = allocator),
            relation_spec = Rule_Relation_Specification {
                kind = .Less_Or_Equals_Than,
                relation = Rule_Distance_Object_Relation {
```

```
                object = get_or_add_object_tag(&object_tags,
"lamp", allocator = allocator),
                distance = 3,
            },
        },
    },
)

generator_info := Generator_Info {
    room_size             = {5, 5, 5},
    mandatory_object_rules = mandatory_object_rules[:],
    optional_object_rules  = optional_rules[:],
    object_infos          = object_infos[:],
    object_tags           = object_tags[:],
}
return generator_info
}

create_test_4_generator_info :: proc(allocator :=
context.allocator) -> Generator_Info {
object_tags := make([dynamic]Object_Tag, allocator =
allocator)

object_infos := make([dynamic]Interior_Object_Info, 0,
2, allocator = allocator)

append(
    &object_infos,
    Interior_Object_Info {
        name = "lamp_1",
        tag = get_or_add_object_tag(&object_tags, "lamp",
allocator = allocator),
        size = {width = 0.5, height = 0.2, depth = 0.5},
        room_side = .Ceiling,
        number_of_tries = 3,
        max_population = 2,
    },
)

append(
    &object_infos,
    Interior_Object_Info {
        name = "sofa_1",
        tag = get_or_add_object_tag(&object_tags, "sofa",
allocator = allocator),
        size = {width = 3.4, height = 2, depth = 2},
        room_side = .Floor,
        number_of_tries = 3,
        max_population = 2,
    },
)

append(
    &object_infos,
    Interior_Object_Info {
        name = "land_lamp_1",
        tag = get_or_add_object_tag(&object_tags,
"land_lamp", allocator = allocator),
        size = {width = 0.5, height = 1.5, depth = 0.5},
        room_side = .Floor,
        number_of_tries = 4,
        max_population = 2,
    },
)

append(
    &object_infos,
    Interior_Object_Info {
        name = "table_1",
        tag = get_or_add_object_tag(&object_tags, "table",
allocator = allocator),
        size = {width = 1.5, height = 1, depth = 1},
        room_side = .Floor,
        number_of_tries = 4,
        max_population = 2,
    },
)

mandatory_object_rules :=
make([dynamic]Mandatory_Object_Rule, 0, 3, allocator =
allocator)
append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "sofa_1"})
append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "lamp_1"})
append(&mandatory_object_rules,
Mandatory_Object_Rule{subject_name = "table_1"})

optional_rules :=
make([dynamic]Optional_Object_Rule, 0, 1, allocator =
allocator)

append(
    &optional_rules,
    Optional_Object_Rule {
        subject = get_or_add_object_tag(&object_tags,
"table", allocator = allocator),
        relation_spec = Rule_Relation_Specification {
            kind = .Less_Or_Equals_Than,
            relation = Rule_Distance_Object_Relation {
                object = get_or_add_object_tag(&object_tags,
"sofa", allocator = allocator),
                distance = 1.5,
            },
        },
    },
)

append(
    &optional_rules,
    Optional_Object_Rule {
        subject = get_or_add_object_tag(&object_tags,
"table", allocator = allocator),
        relation_spec = Rule_Relation_Specification {
            kind = .Less_Or_Equals_Than,
            relation = Rule_Distance_Object_Relation {
                object = get_or_add_object_tag(&object_tags,
"lamp", allocator = allocator),
                distance = 1,
            },
```

```
            },
        },
    )

    append(
        &optional_rules,
        Optional_Object_Rule {
            subject = get_or_add_object_tag(&object_tags,
"sofa", allocator = allocator),
            relation_spec = Rule_Relation_Specification {
                kind = .Less_Or_Equals_Than,
                relation = Rule_Distance_Object_Relation {
                    object = get_or_add_object_tag(&object_tags,
"lamp", allocator = allocator),
                    distance = 1,
                },
            },
        },
    )

    generator_info := Generator_Info {
        room_size             = {5, 5, 5},
        mandatory_object_rules = mandatory_object_rules[:],
        optional_object_rules  = optional_rules[:],
        object_infos          = object_infos[:],
        object_tags           = object_tags[:],
    }
    return generator_info
}

get_or_add_object_tag :: proc(
    object_tags: ^[dynamic]Object_Tag,
    tag_name: string,
    allocator := context.allocator,
) -> Object_Tag {
    new_tag := Object_Tag(tag_name)
    for tag in object_tags {
        if tag == new_tag do return tag
    }

    new_tag_str, error := strings.clone(tag_name, allocator)
    assert(error == nil, "Can't allocate new string for
Object_Tag")
    new_tag = Object_Tag(new_tag_str)
    append(object_tags, new_tag)

    return new_tag
}

delete_object_tag :: proc(tag: Object_Tag, allocator :=
context.allocator) {
    str := string(tag)
    delete(str, allocator)
}

delete_generator_info :: proc(generator_info:
Generator_Info, allocator := context.allocator) {
    using generator_info

    delete(mandatory_object_rules, allocator)
    delete(optional_object_rules, allocator)
    delete(object_infos, allocator)

    for tag in object_tags {
        delete_object_tag(tag)
    }
    delete(object_tags)
}

Interior_Object :: struct {
    bbox: BBox,
    tag:  Object_Tag,
    name: string,
    size: Size,
}

Generate_Interior_Error :: union {
    Generate_Interior_Cant_Place_Mandatory_Object,
}

Generate_Interior_Cant_Place_Mandatory_Object :: struct
#all_or_none {
    mandatory_object_name: string,
    mandatory_object_tag:  Object_Tag,
}

try_get_interior_object_info :: proc(
    interior_object_infos: []Interior_Object_Info,
    name: string,
) -> (
    info: Interior_Object_Info,
    ok: bool,
) {
    for info in interior_object_infos {
        if info.name == name do return info, true
    }

    return {}, false
}

generate_interior :: proc(
    generator_info: Generator_Info,
    allocator := context.allocator,
) -> (
    interior_objects: [dynamic]Interior_Object,
    error: Generate_Interior_Error,
) {
    interior_objects = make([dynamic]Interior_Object,
allocator = allocator)

    for mandatory_object_rule in
generator_info.mandatory_object_rules {
        // get info
        object_info, info_found :=
try_get_interior_object_info(
            generator_info.object_infos[:],
            mandatory_object_rule.subject_name,
        )
        assert(info_found, "Couldn't find object info")

        mandatory_object_bbox, ok := try_add_object(
```

```
            interior_objects[:],
            generator_info.room_size,
            object_info,
            generator_info.optional_object_rules,
            object_info.number_of_tries,
        )

        if !ok {
            delete(interior_objects)
            return nil,
Generate_Interior_Cant_Place_Mandatory_Object {
                mandatory_object_name = object_info.name,
                mandatory_object_tag = object_info.tag,
            }
        }

        append(
            &interior_objects,
            Interior_Object {
                bbox = mandatory_object_bbox,
                tag = object_info.tag,
                name = object_info.name,
                size = object_info.size,
            },
        )
    }

    possible_objects := make(
        [dynamic]Possible_Interior_Object,
        0,
        len(generator_info.object_infos),
        allocator = allocator,
    )

    oi: for object_info in generator_info.object_infos {
        for mandatory_object_rule in
generator_info.mandatory_object_rules {
            if object_info.name ==
mandatory_object_rule.subject_name do continue oi
        }

        possible_count :=
rand.int_max(int(object_info.max_population))
        for _ in 0 ..= possible_count {
            append(
                &possible_objects,
                Possible_Interior_Object {
                    interior_object_info = object_info,
                    already_tried_count =
object_info.number_of_tries,
                },
            )
        }
    }

    for len(possible_objects) > 0 {
        chosen_object, idx := choice_ptr(possible_objects[:])
        object_info, info_found :=
try_get_interior_object_info(
            generator_info.object_infos[:],
            chosen_object.interior_object_info.name,
        )
        assert(info_found, "Couldn't find object info")

        mandatory_object_bbox, succesfully_placed :=
try_add_object(
            interior_objects[:],
            generator_info.room_size,
            object_info,
            generator_info.optional_object_rules,
            object_info.number_of_tries,
        )

        if succesfully_placed {
            append(
                &interior_objects,
                Interior_Object {
                    bbox = mandatory_object_bbox,
                    tag = object_info.tag,
                    name = object_info.name,
                    size = object_info.size,
                },
            )

            unordered_remove(&possible_objects, idx)
            continue
        } else {
            chosen_object.already_tried_count -= 1
            if chosen_object.already_tried_count <= 0 {
                unordered_remove(&possible_objects, idx)
                continue
            }
        }
    }

    return interior_objects, {}
}

choice_ptr :: proc(array: $A/[]$E, gen :=
context.random_generator) -> (elem: ^E, idx: i64) {
    n := i64(len(array))
    if n < 1 {
        return nil, 0
    }

    idx = rand.int63_max(n, gen)
    return &array[idx], idx
}

Iterator_Tag_Filtering_Mode :: enum {
    Filter_Specified,
    Find_Specified,
}

Optional_Rules_Iterator :: struct {
    rules:          []Optional_Object_Rule,
    filter_tag:      Object_Tag,
```

```
    current_idx:        int,
    tag_filetring_mode: Iterator_Tag_Filtering_Mode,
}

iterate_optional_rules :: proc(iter:
^Optional_Rules_Iterator) -> (rule: Optional_Object_Rule,
ok: bool) {
    if iter.current_idx >= len(iter.rules) do return {}, false

    for rule, idx in iter.rules[iter.current_idx:] {
        iter.current_idx += 1
        switch iter.tag_filetring_mode {
        case .Filter_Specified:
            if rule.subject != iter.filter_tag {
                return rule, true
            }
        case .Find_Specified:
            if rule.subject == iter.filter_tag {
                return rule, true
            }
        }
    }

    return {}, false
}

Interior_Objects_Iterator :: struct {
    interior_objects:   []Interior_Object,
    filter_tag:         Object_Tag,
    current_idx:        int,
    tag_filetring_mode: Iterator_Tag_Filtering_Mode,
}

iterate_interior_objects :: proc(iter:
^Interior_Objects_Iterator) -> (rule: Interior_Object, ok:
bool) {
    if iter.current_idx >= len(iter.interior_objects) do return
{}, false

    for rule, idx in iter.interior_objects[iter.current_idx:] {
        iter.current_idx += 1
        switch iter.tag_filetring_mode {
        case .Filter_Specified:
            if rule.tag != iter.filter_tag {
                return rule, true
            }
        case .Find_Specified:
            if rule.tag == iter.filter_tag {
                return rule, true
            }
        }
    }

    return {}, false
}

get_distance :: get_distance_simple

get_distance_simple :: proc(box_a, box_b: BBox) -> f32 {
    if rl.CheckCollisionBoxes(box_a, box_b) do return 0

    box_a_center := box_a.max - box_a.min
    box_b_center := box_b.max - box_b.min

    ray_a := rl.Ray {
        position  = box_a_center,
        direction = rl.Vector3Normalize(box_b_center),
    }
    ray_a_coll := rl.GetRayCollisionBox(ray_a, box_b)

    ray_b := rl.Ray {
        position  = box_b_center,
        direction = rl.Vector3Normalize(box_a_center),
    }
    ray_b_coll := rl.GetRayCollisionBox(ray_b, box_a)

    distance := rl.Vector3Distance(ray_b_coll.point,
ray_a_coll.point)
    return distance
}

try_add_object :: proc(
    interior_objects: []Interior_Object,
    room_size: Size,
    subject_info: Interior_Object_Info,
    optional_rules: []Optional_Object_Rule,
    number_of_tries: u8,
) -> (
    bbox: BBox,
    ok: bool,
) {
    tries_loop: for i in 0 ..< number_of_tries {
        pos := pick_room_point(room_size, subject_info.size,
subject_info.room_side)
        new_box := BBox {
            min = pos,
            max = pos + to_vec3(subject_info.size),
        }

        opt_rules_iter := Optional_Rules_Iterator {
            rules            = optional_rules,
            filter_tag       = subject_info.tag,
            tag_filetring_mode = .Find_Specified,
        }
        for opt_rule in iterate_optional_rules(&opt_rules_iter)
{
            follows_rule :=
check_optional_rule(interior_objects, subject_info, pos,
new_box, opt_rule)
            if !follows_rule do continue tries_loop
        }

        if !check_collisions(interior_objects, new_box) do
return new_box, true
    }

    return {}, false
}

check_optional_rule :: proc(
```

```
    interior_objects: []Interior_Object,
    subject_info: Interior_Object_Info,
    subject_position: rl.Vector3,
    subject_bbox: BBox,
    opt_rule: Optional_Object_Rule,
) -> bool {
    switch rel in opt_rule.relation_spec.relation {
    case Rule_Size_Object_Relation:
        interior_objects_iter := Interior_Objects_Iterator {
            interior_objects   = interior_objects,
            filter_tag         = rel.object,
            tag_filetring_mode = .Find_Specified,
        }

        for interior_object in
iterate_interior_objects(&interior_objects_iter) {
            object_size_field_value :=
get_value_size_field(rel.kind, interior_object.size)
            switch opt_rule.relation_spec.kind {
            case .Equals:
                return get_value_size_field(rel.kind,
subject_info.size) == object_size_field_value
            case .Less_Or_Equals_Than:
                return get_value_size_field(rel.kind,
subject_info.size) <= object_size_field_value
            case .Bigger_Or_Equals_Than:
                return get_value_size_field(rel.kind,
subject_info.size) >= object_size_field_value
            case .None:
                unreachable()
            }
        }

        return true

    case Rule_Distance_Object_Relation:
        interior_objects_iter := Interior_Objects_Iterator {
            interior_objects   = interior_objects,
            filter_tag         = rel.object,
            tag_filetring_mode = .Find_Specified,
        }

        for interior_object in
iterate_interior_objects(&interior_objects_iter) {
            actual_distance :=
get_distance(interior_object.bbox, subject_bbox)
            distance_rule_applies := check_distance_relation(
                opt_rule.relation_spec.kind,
                rel.distance,
                actual_distance,
            )

            //* we can place new subject only if it's follows all
the rules
            if !distance_rule_applies do return false
        }

        return true

    case Rule_Size_Value_Relation:
        switch opt_rule.relation_spec.kind {
        case .Equals:
            return get_value_size_field(rel.kind,
subject_info.size) == rel.value
        case .Less_Or_Equals_Than:
            return get_value_size_field(rel.kind,
subject_info.size) <= rel.value
        case .Bigger_Or_Equals_Than:
            return get_value_size_field(rel.kind,
subject_info.size) >= rel.value
        case .None:
            unreachable()
        }
    }

    unreachable()
}

check_distance_relation :: proc(rel_kind:
Rule_Value_Relation_Kind, expected_dist, actual_dist:
f32) -> bool {
    switch rel_kind {
    case .Equals:
        return expected_dist == actual_dist
    case .Less_Or_Equals_Than:
        return expected_dist <= actual_dist
    case .Bigger_Or_Equals_Than:
        return expected_dist >= actual_dist
    case .None:
        unreachable()
    }

    unreachable()
}

get_value_size_field :: proc(kind: Rule_Size_Field_Kind,
size: Size) -> f32 {
    switch kind {
    case .Width:
        return size.width
    case .Height:
        return size.height
    case .Depth:
        return size.depth
    case .None:
        unreachable()
    }

    unreachable()
}

check_collisions :: proc {
    check_collisions_all,
    check_collisions_tag,
}

check_collisions_all :: proc(boxes: []Interior_Object, box:
BBox) -> (there_is_collision: bool) {
    for box_a in boxes {
```

```
        if rl.CheckCollisionBoxes(box_a.bbox, box) do return
true
    }

    return false
}

check_collisions_tag :: proc(
    boxes: []Interior_Object,
    object_tag: Object_Tag,
    box: BBox,
) -> (
    there_is_collision: bool,
) {
    for interior_object in boxes {
        if interior_object.tag != object_tag do continue
        if rl.CheckCollisionBoxes(interior_object.bbox, box)
do return true
    }

    return false
}

pick_room_point :: proc(room_size, object_size: Size,
room_side: Room_Bounding_Side) -> rl.Vector3 {
    assert(object_size.width < room_size.width)
    assert(object_size.height < room_size.height)
    assert(object_size.depth < room_size.depth)

    width, height, depth: f32
    switch room_side {
    case .Ceiling:
        height = room_size.height - object_size.height
        width = rand.float32_range(0, room_size.width -
object_size.width)
        depth = rand.float32_range(0, room_size.depth -
object_size.depth)

    case .Floor:
        height = 0
        width = rand.float32_range(0, room_size.width -
object_size.width)
        depth = rand.float32_range(0, room_size.depth -
object_size.depth)

    case .Wall:
        wall := rand.choice_enum(Room_Wall)
        if wall == .None do wall = .Front
        switch wall {
        case .Front:
            width = rand.float32_range(0, room_size.width -
object_size.width)
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = 0

        case .Right:
            width = room_size.width - object_size.width
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = rand.float32_range(0, room_size.depth -
object_size.depth)

        case .Back:
            width = rand.float32_range(0, room_size.width -
object_size.width)
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = room_size.depth - object_size.depth

        case .Left:
            width = 0
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = rand.float32_range(0, room_size.depth -
object_size.depth)

        case .None:
            unreachable()
        }

    case .Corner:
        wall := rand.choice_enum(Room_Wall)
        if wall == .None do wall = .Front
        //* corner from the wall and left wall of it
        switch wall {
        case .Front:
            //* front and left
            width = 0
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = 0

        case .Right:
            //* right and front
            width = room_size.width - object_size.width
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = 0

        case .Back:
            //* back and right
            width = room_size.width - object_size.width
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = room_size.depth - object_size.depth

        case .Left:
            //* left and back
            width = 0
            height = rand.float32_range(0, room_size.height -
object_size.height)
            depth = room_size.depth - object_size.depth

        case .None:
            unreachable()
        }

    // case .Ceiling_And_Wall, .Floor_And_Wall,
.Corner_And_Ceiling, .Corner_And_Floor:
```

```
    //  panic("Not implemented room side yet")
    case .None:
        panic("Room side wasn't set")
    }

    pos := rl.Vector3{width, height, depth}
    return pos
}

draw_interior :: proc(generator_info: Generator_Info,
interior_bounding_boxes: []Interior_Object, delta_time:
f32) {
    room_bbox := BBox {
        min = {},
        max = to_vec3(generator_info.room_size),
    }
    rl.DrawBoundingBox(room_bbox, rl.BLACK)

    for interior_object in interior_bounding_boxes {
        rl.DrawBoundingBox(interior_object.bbox,
rl.GREEN)
    }
}

check_bounding_box_collision :: proc(boxA, boxB: BBox)
-> bool {
    // Check if they are NOT overlapping on any axis
    if boxA.max.x <= boxB.min.x || boxA.min.x >=
boxB.max.x do return false
    if boxA.max.y <= boxB.min.y || boxA.min.y >=
boxB.max.y do return false
    if boxA.max.z <= boxB.min.z || boxA.min.z >=
boxB.max.z do return false

    // If none of the above are true, they are colliding
    return true
}
```