ДВНЗ «ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»

Факультет комп'ютерно-інформаційних технологій та автоматизації

Кафедра прикладної математики і інформатики

«До захисту допущено»
В. о. завідувача кафедри
Наталія МАСЛОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)
«_____»_____2023 р.

Випускна кваліфікаційна робота

бакалавра
(освітній ступінь)

на тему «Розробка додатку для створення ігрових світів для Minecraft з відкритих карт»

зі спецчастиною: «Розробка та програмна реалізація додатку засобами Java»

Виконав: студент 4 курсу, групи ІПЗ-19
(цифри групи)

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

освітньої програми 121 Інженерія програмного забезпечення

Матюхін І. Д.
(прізвище та ініціали) (підпис)

Керівник доц. кафедри ПМІ, к.т.н. Тетяна АЛТУХОВА
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Рецензент доц. кафедри ЕТ, к.т.н., доц. Олександр ШТЕПА
(посада, науковий ступінь, вчене звання, Ім'я та ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Луцьк – 2023р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра прикладної математики та інформатики
Освітній ступінь бакалавр
Спеціальність 121 Інженерія програмного забезпечення

Освітня програма 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
В.о. завідувача кафедри
_____Наталія МАСЛОВА
"____" _____20___року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Матюхіну Івану Денисовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка додатку для створення ігрових світів для Minecraft з відкритих карт» зі спецчастиною: «Розробка та програмна реалізація додатку засобами Java»
Керівник роботи Алтухова Тетяна Володимирівна, к.т.н., доц. кафедри ПМІ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від "05" травня 2023 року № 175
2. Строк подання студентом роботи ___09.06.2023____
3. Вихідні дані до роботи результати переддипломної практики
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд сучасних аналогів додатків ранжирування музики;
- огляд сучасних інструментів проєктування та методів дослідження Web-додатків;
- проєктування Web-додатку;
- реалізація Web-додатку;
- тестування Web-додатку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
робота містить 14 рисунків, 4 таблиці, 1 формулу та інший матеріал, у кількості, достатній для демонстрації результатів кваліфікаційної роботи бакалавра.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Назарова І.А., доцент каф. ПМІ		9.06.2023

7. Дата видачі завдання 05.05.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	05.05.2023-12.05.2023	
2	Визначення основних завдань і вимог до застосунку	13.05.2023-19.05.2023	
3	Вивчення можливостей додаткових бібліотек	20.05.2023-22.05.2023	
4	Проєктування додатку для генерації світів Minecraft, створення ескізу інтерфейсу	23.05.2023-25.06.2023	
5	Написання коду для додатку, створення інтерфейсу	26.05.2023-04.06.2023	
6	Підготовка пояснювальної записки	05.06.2023-07.06.2023	
7	Оформлення супутніх матеріалів (розробка графічного матеріалу, тощо) та рецензування роботи	07.06.2023-09.06.2023	
8	Захист роботи	22.06.23	

Студент _____ Іван МАТЮХІН
(підпис) (Ім'я та ПРІЗВИЩЕ)

Керівник роботи _____ Тетяна АЛТУХОВА
(підпис) (Ім'я та ПРІЗВИЩЕ)

АНОТАЦІЯ

Матюхін Іван Денисович. Розробка додатку для створення ігрових світів для Minecraft з відкритих карт» зі спецчастиною: «Розробка та програмна реалізація додатку засобами Java» / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення». ДВНЗ ДонНТУ, Покровськ, Луцьк, 2023.

Дана дипломна робота присвячена розробці додатку, який спрямований на полегшення процесу створення ігрових світів для популярної гри Minecraft з використанням відкритих карт. Гра Minecraft, яка є однією з найуспішніших ігор в світі, надає гравцям великі можливості для творчості, але процес створення власних світів може бути складним і завданням яке займає багато часу.

Метою цієї роботи є розробка зручного та інтуїтивно зрозумілого інструменту, який допоможе користувачам створювати свої власні ігрові світи для Minecraft на основі відкритих карт. Додаток буде забезпечувати можливість імпортувати готові картографічні дані з джерел Openstreetmap та даних рельєфу отриманих під час місії SRTM, і перетворювати їх у формат, сумісний з грою Minecraft. Крім того, додаток буде містити систему гнучких налаштувань для створення світу.

Ключові слова: MINECRAFT, OPENSTREETMAP, ГЕНЕРУВАННЯ ІГРОВИХ СВІТІВ MINECRAFT, SRTM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Minecraft – це відеогра, що заснована на відкритому світі та будівництві. Гравці мають можливість розміщувати та руйнувати блоки, будувати структури, досліджувати та взаємодіяти з різноманітними об'єктами та персонажами.

Openstreetmap – це проєкт зі створення вільної та відкритої географічної бази даних. Він дозволяє користувачам з усього світу збирати, редагувати та використовувати географічну інформацію, таку як мапи, дороги, будівлі та інші об'єкти.

Місія SRTM (Shuttle Radar Topography Mission) була спільним проектом NASA та Національного географічного агентства США, який використовував радарну технологію для створення деталізованої цифрової моделі рельєфу Землі. Головною метою місії було отримання високоякісних географічних даних, зокрема цифрових моделей висот, які використовуються в геодезії, картографії та наукових дослідженнях.

Файл osm – це файл який можна завантажити з офіційного сайту openstreetmap, файл містить географічну інформацію і використовується для зберігання карт. Назва імені файлу що вказує на формат закінчується на «.osm».

Файл GeoTIFF — це файл який містить інформацію про висоти рельєфу на певній території. Назва імені файлу що вказує на формат закінчуєтьс на «.tif».

ВСТУП

У сучасному світі комп’ютерні ігри сильно увійшли в життя багатьох людей. Одною з найпопулярніших ігор на даний момент є Minecraft, - гра у жанрі пісочниця, яка дозволяє користувачам створювати ігрові світи. У даній роботі буде розглянуто розробку спеціального додатку для автоматичного генерування світів для гри Minecraft з відкритих карт.

Метою роботи є розробка додатку, який надасть можливість користувачам створювати ігрові світи для гри Minecraft використовуючи дані з відкритих карт. Основною метою додатку буде надання інструменту користувачам для створення ігрових світів у Minecraft, дані для яких застосунок буде отримувати з відкритих карт, а саме з Openstreetmap та GeoTIFF. Таким чином, користувачі зможуть відтворювати різні міста або їх частини, населені пункти або ландшафти, отримані з відкритих карт у ігровому середовищі Minecraft.

Перший розділ роботи буде описувати аналіз сучасного стану програм для створення ігрових світів. У даному розділі буде розглянуто потенційних користувачів подібних додатків і чому вони можуть бути корисними, як використовуються дані додатки. Буде розглянуто які існують додатки та/або програмні модулі (плагіни) для створення ігрових світів загалом, їх переваги та недоліки. Буде описано у чому саме користь від додатку для створення ігрових світів у Minecraft і хто може бути його потенційним користувачем.

Другий розділ описує формування вимог для додатку для створення ігрових світів, які саме повинні бути функції у цього додатку, які алгоритми він використовує, які є обмеження у додатку та на яких операційних системах додаток повинен працювати. Звідки і з яких карт для додатку буде використано інформацію і яку інформацію містять які карти.

Третій розділ описує архітектуру додатку, його класи і важливі методи і поля у цих класах. Також у розділі наведено UML діаграму основних класів і

ескіз інтерфейсу для взаємодії з користувачем. Крім цього розділ має описи бібліотек які буде використано під час розробки.

Четвертий розділ описує процес використання додатку, показує його тестування і результати його застосування. Також показує як додаток обробляє помилки, які можуть виникнути під час використання програми.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану створення ігрових світів

Наразі існують різні інструменти які дозволяють генерувати моделі певних місцевостей з даних отриманих з відкритих карт, вони дозволяють розробникам ігор автоматично створювати моделі різних місцевостей, що може значно пришвидшити процес розробки певної гри. Проте для певного сегменту користувачів існує потреба у можливості створення ігрових світів з відкритих карт у іграх які вже існують.

На даний момент існують ігри у жанрі пісочниця які дозволяють створювати ігрові світи, однією з найпопулярніших ігор такого типу є Minceraft. У цій грі можна створювати світи і займатися творчістю у цих світах, а саме — будувати різні об’єкти. Іноді є потреба створити світ схожий на певну місцевість, що існує у реальному житті. При цьому створювати дану місцевість власноруч у грі Minecraft може бути тривалим процесом, який потребує багато праці. Вирішити або поліпшити дану проблему може спеціальний додаток, що отримуватиме файли з даними про певну місцевість і генеруватиме каталог зі створеним світом для гри Minecraft.

Даний застосунок може бути корисним наступним сегментам користувачів:

1) Люди і організації, що застосовують технології гейміфікації для взаємодії з дітьми.

2) Головні адміністратори серверів Mintcraft.

3) Гравці Minecraft, що мають намір швидко згенерувати світ у грі схожий на певну місцевість у реальному світі.

1.2 Огляд аналогів додатків створення ігрових світів

На даний момент існують різні доповнення для різних програмних систем що дозволяють генерувати світи або 3Д-моделі світів з карт Openstreetmap [1], серед них можна виділити доповнення для програми Blender і доповнення для серверів гри Minecraft.

Розглядаючи доповнення для програми Blender можна виділити переваги і недоліки. Перевагами цього доповнення є гнучкість налаштувань і вільне редагування світу як 3Д-моделі, далі згенеровані моделі можуть бути застосовані розробниками ігор на різних ігрових рушіях для створення ігрового світу певної місцевості. Недоліком є складне застосування і те, що цей плагін підійде для користувачів що створюють власні ігри, що не вирішує проблему користувачів які хочуть генерувати світи для ігор які вже існують.

Також можна розглянути доповнення для сервера Minecraft, яке дозволяє створити ігровий світ з відкритих карт. Перевагою даного плагіну є наявність багатьох налаштувань. Недоліком даного плагіну є те, що його використання потребує створення власного серверу, так як він створений для серверів, також недоліком є те, що його застосування потребує досить сильного досвіду користувача і вміння налаштовувати сервер і працювати з командним рядком у Minecraft.

1.3 Формування мети та основних завдань

Метою і основним завданням є створення додатку призначеного для використання на персональних комп’ютерах, який буде приймати дані з файлу відкритих карт Openstreetmap у форматі osm і дані для генерування рельєфу з файлів формату GeoTIFF, після чого додаток буде надавати користувачу

можливість генерації світу для гри Minecraft і збереження згенерованого світу у вигляді каталогу зі спеціальними файлами за вказаною директорією.

Потенційними користувачами даного додатку є:

1) Люди і організації що працюють з дітьми застосовуючи технології гейміфікації.
2) Головні адміністратори серверів Minecraft.
3) Гравці.

Основні завдання для розробки додатку включають:

- Розробка архітектури додатку.
- Створення алгоритмів для імпорту та аналізу картографічних даних з файлів формату osm.
- Створення алгоритмів для імпорту та біквадратичної інтерполяції даних ландшафту з файлів формату GeoTIFF.
- Створення алгоритмів для побудови рельєфу і об’єктів у світі Minecraft з отриманих даних.
- Впровадження інтерфейсу користувача, який дозволяє зручно взаємодіяти з програмою та налаштовувати параметри для створення світу.
- Тестування роботи додатку.

РОЗДІЛ 2
ФОРМУВАННЯ ВИМОГ ТА ОГЛЯД АЛГОРИТМІВ ДЛЯ СТВОРЕННЯ ДОДАТКУ

2.1 Визначення та опис основних вимог додатку що розробляється

Опис додатку: додаток є програмою яку можна використовувати на персональних комп’ютерах на базі операційних систем Windows і Linux і який дозволяє користувачам генерувати ігрові світи у грі Minecraft використовуючи інформацію про будівлі, дороги, водойми та типи місцевостей з відкритих карт Openstreetmap і інформацію про рельєф з карт NASA отриманих під час місії SRTM. Результатом генерації є файл світу Minecraft, який можна відкрити у грі Minecraft Java Edition версії 1.8 та вище. Згенерований світ для гри повторюватиме дороги, водойми, будівлі та типи місцевостей що були позначені на карті Openstreetmap і рельєф, отриманий з карт отриманих під час місії SRTM. Згенерований світ буде формату «порожній» і міститиме дані тільки з ділянки території з карт згідно налаштувань.

Призначення: забезпечити можливість використовувати дані з відкритих карт Openstreetmap та карт отриманих у місії NASA SRTM для генерування світів у грі Майнкрафт за даними з цих карт.

Як користувачі повинні використовувати додаток: для генерації світу для Minecraft користувачу потрібно зайти на офіційний сайт Openstreetmap та знайти територію яку він хоче згенерувати, після чого він повинен за допомогою функції «Експорт» завантажити територію з сайту у вигляді файлу формату osm. Далі користувач повинен відкрити додаток і обрати у ньому шлях до файлу osm і зачекати доки програма проаналізує файл. Після аналізу файлу, користувачу буде показано панель налаштувань для генерації світу, використовуючи дану панель користувач повинен встановити усі параметри за власним бажанням і натиснути

кнопку «Згенерувати» і зачекати, поки світ буде згенеровано і збережено у вказану директорію. Треба зазначити, що вказання шляху до файлів рельєфу GeoTIFF відбувається через панель налаштувань.

Головні функції програмного продукту для користувача:

1)Завантаження файлу карти osm у додаток.

2)Перегляд проаналізованої інформації з завантаженого файлу карти Openstreetmap (розширення osm).

3)Налаштування параметрів для генерації світу.

4)Генерація світу для гри Minecraft.

Перелік інформації яку виводить додаток користувачу після аналізу файлу формату osm:

1)Список типів доріг які було знайдено на карті (тип доріг вказано у файлах osm).

2)Список типів місцевостей які було знайдено на карті (тип місцевостей вказано у файлах osm).

3)Список типів водойм знайдених на карті (тип водойми вказано у файлах osm).

4)Список типів будівель, які було знайдено на карті (тип будівлі визначається за кількістю поверхів).

5)Список доступних налаштувань для генерації світу.

Примітка: для типів доріг, типів місцевостей і типів водойм додаток повинен провести вказання назв даних місцевостей українською мовою. Переклад для назв наведено у таблиці 2.2.1. Усі назви об’єктів які не буде знайдено у таблиці перекладу при аналізі файлу osm треба вказати без перекладу використовуючи позначення, яке було застосоване у файлі osm.

Таблиця 2.2.1 — Переклад назв об’єктів

Об’єкт	Назва у файлі osm	Назва українською
Дорога	motorway	Автостради
Дорога	trunk	Магістралі
Дорога	primary	Головні дороги
Дорога	secondary	Міжміські дороги
Дорога	tertiary	Місцеві дороги
Дорога	unclassified	Некласифіковані дороги
Дорога	residential	Житлові дороги
Дорога	motorway_link	З'єднання з автострадами
Дорога	trunk_link	З'єднання з магістралями
Дорога	primary_link	З'єднання з головними дорогами
Дорога	secondary_link	З'єднання з міжміськими дорогами
Дорога	tertiary_link	З'єднання з місцевими дорогами
Дорога	living_street	Дороги житлових місцевостей
Дорога	service	Дороги службового призначення
Дорога	pedestrian	Пішохідні дороги
Дорога	bus_guideway	Автобусні колії
Дорога	escape	Дороги для аварійних зупинок
Дорога	raceway	Дороги для перегонів
Дорога	road	Невизначені дороги
Дорога	busway	Автобусні дороги
Дорога	footway	Тротуари
Дорога	bridleway	Дороги для конів
Дорога	steps	Сходи
Дорога	corridor	Дороги через будівлі
Дорога	path	Неспецифічні шляхи
Дорога	cycleway	Велосипедні доріжки
Дорога	proposed	Заплановані дороги
Дорога	construction	Дороги що будуються
Місцевість	commercial	Комерційні місцевості
Місцевість	construction	Місцевості будівництва
Місцевість	education	Місцевості освіти

Продовження таблиці 2.2.1 -

Об’єкт	Назва у файлі osm	Назва українською
Місцевість	Fairground	Місцевості ярмарку
Місцевість	industrial	Індустріальні місцевості
Місцевість	residential	Житлові масиви
Місцевість	retail	Роздрібні підприємства
Місцевість	institutional	Інституційні місцевості
Місцевість	aquaculture	Аква-ферми
Місцевість	allotments	Ділянки для місцевих
Місцевість	farmland	Сільськогосподарські угіддя
Місцевість	farmyard	Ділянки з господарськими будівлями
Місцевість	flowerbed	Території для квітів
Місцевість	forest	Ліси та посадки
Місцевість	greenhouse_horticulture	Території теплиць
Місцевість	meadow	Луги та пасовища
Місцевість	orchard	Плодові дерева та кущі
Місцевість	plant_nursery	Виробництво рослин
Місцевість	vineyard	Виноградники
Місцевість	brownfield	Території для забудов
Місцевість	cemetery	Цвинтарі
Місцевість	depot	Депо
Місцевість	garages	Гаражі
Місцевість	grass	Трава
Місцевість	greenfield	Території запланованих забудов
Місцевість	landfill	Звалища
Місцевість	religious	Релігійні території
Місцевість	village_green	Громадські земля у селі
Місцевість	winter_sports	Зони зимового спорту
Місцевість	recreation_ground	Зони загального відпочинку
Водойма	basin	Басейни
Водойма	reservoir	Водосховища
Водойма	salt_pond	Соляні стави

Продовження таблиці 2.2.1 -

Об’єкт	Назва у файлі osm	Назва українською
Водойма	River	Річки
Водойма	oxbow	Заводі
Водойма	canal	Канали
Водойма	ditch	Канави
Водойма	lock	Шлюзові камери
Водойма	fish_pass	Зони приходу риби
Водойма	lake	Озера
Водойма	pond	Стави
Водойма	basin	Басейни
Водойма	lagoon	Лагуни
Водойма	stream_pool	Басейни струмків
Водойма	reflecting_pool	Невеликі басейни
Водойма	moat	Рови
Водойма	wastewater	Стічні води
Водойма	riverbank	Береги річок
Водойма	stream	Потоки
Водойма	tidal_channel	Припливні канали
Водойма	drain	Дренажі
Водойма	fairway	Судноплавні маршрути
Водойма	dock	Зони для кораблів
Водойма	boatyard	Місця будівництва суден
Водойма	dam	Дамби
Водойма	weir	Водозливи
Водойма	waterfall	Водоспади
Водойма	lock_gate	Шлюзи

Перелік і опис налаштувань, які доступні користувачу після аналізу файлу карти Openstreetmap і перед генеруванням світу для гри Minecraft:

1)Для генерації світу є налаштування, що вказує шлях до директорії для збереження світу для гри Minecraft. За замовчуванням не вказано.

2)Для генерації світу є налаштування що вказує на назву Світу. За замовчуванням назва світу «NewWorld1».

3)Для генерації світу є налаштування що визначає масштаб. Масштаб вказує на кількість блоків у грі Minecraft для позначення одного метра у реальному світі, повинна бути можливість обрати зі списку 1 метр це 1 блок, 1 метр до 2 блоки або ввести масштаб власноруч. За замовчуванням 1 метр це 1 блок.

4)Для генерації світу є налаштування того, у яких одиницях виміру буде застосовано значення розміру території яку буде згенеровано, доступні одиниці виміру: метри, блоки. За замовчуванням метри.

5)Для генерації світу є налаштування значення розміру для територій, яку буде згенеровано, доступні розміри для обрання 250 на 250, 1000 на 1000, 2500 на 2500, 5000 на 5000, 10000 на 10000. За замовчуванням 250 на 250.

6)Для генерації світу є налаштування що вказує на точку на карті Openstreetmap у широті і довготі, яку буде позначено як центр для обирання даних для генерації території. За замовчуванням це центр області, яка вказана у завантаженому файлі osm.

7)Для генерації світу є налаштування, що вказує на центр згенерованої території у світі Minecraft. За замовчуванням точка (0, 0).

8)Для генерації світу є налаштування того, чи треба генерувати рельєф. Якщо треба генерувати рельєф, то стають доступними налаштування для генерації рельєфу. За замовчуванням вказано що рельєф генерувати не треба.

9)Для генерації рельєфу є налаштування того, який тип рельєфу треба обрати, доступні варіанти: згенерувати плоский рельєф або згенерувати рельєф з файлів.

10) Для генерування плоского рельєфу доступне налаштування що вказує на висоту плоского рельєфу.

11) Для генерації рельєфу з файлів є вказання шляху до директорії з файлами рельєфу, які будуть використані для отримання даних рельєфу. Формат файлів GeoTIFF. За замовчуванням шлях не вказано.

12) Для генерації рельєфу з файлів є налаштування того, чи треба вказувати власне значення для масштабу висоти. За замовчуванням не треба.

13) Для генерації рельєфу з файлів є налаштування для масштабу висоти. За замовчуванням вказано 1 метр висоти у реальному світі до 1 метра висоти у світі Minecraft.

14) Для кожного типу дороги є налаштування того, чи треба її генерувати. За замовчуванням кожен тип дороги треба генерувати.

15) Для кожного типу дороги є налаштування що визначає ширину дороги у метрах. За замовчуванням для кожної дороги вказано 3 метри.

16) Для кожного типу дороги є налаштування що визначає вид блоків, з яких буде згенеровано дорогу. За замовчуванням для кожної дороги вказано вид блоків «Каміння»

17) Для кожного типу місцевості є налаштування того, чи треба покривати вказаними блоками місцевості з даним типом. За замовчуванням усі місцевості треба покрити блоками вказаного типу.

18) Для кожного типу місцевості є налаштування що визначає вид блоків, якими буде покрито дану місцевість. За замовчуванням для покриття кожної місцевості вказано вид блоків «Земля з травою».

19) Для кожного типу водойми є налаштування того, чи треба заповнювати водою водойм з даним типом. За замовчуванням для кожної водойми вказано що треба заповнити його водою з вказаним типом.

20) Для кожного типу будівлі є налаштування того, чи треба генерувати даний тип будівлі. За замовчуванням вказано що треба генерувати кожен тип будівлі.

21) Для кожного типу будівлі є налаштування що визначає вид блоку, з якого буде згенеровано будівля даного типу. За замовчуванням вказано вид блоків «Жовта шерсть»

22) Для усіх будівель одразу є налаштування того, яка повинна бути висота поверху у метрах. За замовчуванням 4 метри.

Під час користування програмою можуть виникнути помилки, при виникненні помилок користувачу будуть виведені певні повідомлення:

1)Помилка «Помилка аналізу або читання файлу карти osm» може виникнути у разі відсутності файлу, відсутності права на читання файлу або пошкодження файлу osm.

2)Помилка «Помилка читання файлів GeoTIFF» може виникнути у разі відсутності права на читання одного з потрібних файлів або пошкодження одного з потрібних файлів GeoTIFF, або якщо файли не відповідають структурі формату.

3)У разі введення у налаштування неправильних даних, таких як букви у поле для введення числа або занадто великого числа усі поля з некоректними даними повинні бути зафарбовані червоним кольором. При виникненні цієї помилки буде виведено повідомлення, що вказуватимуть на те, у яких саме полях значення повинно бути змінено.

4)Помилка «Виникла невідома помилка при збереженні світу» може виникнути у разі відсутності прав доступу на створення каталогів або файлів у директорії для збереження створеного світу.

2.2 Визначення алгоритму реалізації процесу створення ігрових світів

Для генерування світу користувачу спочатку потрібно завантажити у програму файл території на мапі у форматі osm. Коли користувач вказує шлях до файлу, програма зчитує його вміст у оперативну пам’ять, після чого починається аналіз отриманої інформації, який складається з двох етапів.

Файли формату osm мають структуру xml, тому програма після читання файлу osm починає перший етап аналізу файлу, який полягає у тому, щоб розібрати структуру xml усередині файлу. Результатом першого етапу аналізу файлу буде екземпляр класу, який надає список методів для отримання інформації з будь-якого вузла (node) зі структури xml, включаючи: назву вузла, його вміст у вигляді тексту, список параметрів вузла (назви параметрів і значення параметрів), список дочірніх вузлів.

Далі програма переходить до другого етапу, — отримання потрібної інформації. Використовуючи екземпляр класу, сформований під час першого етапу, програма отримує значення для двох крайніх точок, що вказані у широті і довготі і які вказують область на поверхні Землі, географічні дані для якої вказано у цьому файлі. Також програма отримує значення у широті і довготі для усіх точок усіх полігонів і ліній, що позначені на мапі і на що вказує даний полігон або лінія. Полігони що вказують на будівлі, певні території і водойми та лінії що вказують на дороги і водойми (річки) зберігаються для подальшої обробки; додатково до цього, програма отримує іншу інформацію, наприклад, кількість поверхів для кожної будівлі. На даному етапі програма формує списки екземплярів класів для будівель, доріг, водойм і місцевостей та зберігає ці списки у спеціальному екземплярі класу, що описує дані отримані з мапи.

Після аналізу обраного файлу, програма надає доступ користувачу до налаштувань. У налаштуваннях є списки отриманих з файлу типів доріг, типів будівель, типів місцевостей і типів водойм. Для кожного з цих типів користувач може зробити власні налаштування зазначених параметрів. Додатково користувач може вказати власні значення загальних (для усього світу) налаштувань.

Після вказання налаштувань користувач може перейти до генерації світу, для цього йому треба натиснути відповідну кнопку. При переході до генерації світу програма перевіряє усі вказані налаштування на наявність помилок, якщо помилок нема, програма переходить до генерування і збереженні світу.

Якщо у налаштуваннях користувачем було вказано на необхідність генерації рельєфу і також було вказано шлях до файлів GeoTIFF, то програма переходить до аналізу файлів GeoTIFF. Для аналізу кожного файлу GeoTIFF програма отримує значення у широті і довготі для точок, що описують область, яку покриває кожен з файлів, також для кожного файлу визначається показник роздільної здатності висот рельєфу.

Генерування світу проходить у декілька етапів, спочатку програма створює п’ять двовимірних масивів. Перший масив містить висоти рельєфу для генерації;

другий масив містить дані про блоки для генерування будівель; третій масив містить дані про блоки для генерування доріг, четвертий масив містить дані про блоки для генерування територій; четвертий масив містить дані про блоки для генерування водойм.

Для генерації світу використовуються дані з файлу osm і файлів GeoTIFF. При наявності вказаних файлів GeoTIFF для генерації рельєфу програма спочатку буде генерувати рельєф.

Для генерування рельєфу програма використовує дані з файлів GeoTIFF які вказують на висоти рельєфу у різних точках поверхні Землі над рівнем моря. Спочатку програма виділяє дані для рельєфу для певної території з файлів GeoTIFF, потім вона знаходить різницю між найнижчою і найвищою точками рельєфу для даної території, після чого знайдену різницю помножує на масштаб вказаний для налаштувань для рельєфу (якщо явно не вказано масштаб, то масштаб дорівнює 1) і перевіряє на те, що ця різниця менша за 255. Якщо різниця більша за 255 то висоти рельєфу буде змінено пропорційно таким чином, щоб різниця між найвищою і найнижчою точкою була 255. Для генерування рельєфу у світі Minecraft для кожної точки буде застосовано біквадратичну інтерполяцію.

Для генерування будівель, доріг, місцевостей і водойм програма використовує полігони і ламані лінії у вигляді списків точок, отриманих з файлу osm. Далі програма виконує математичні перетворення щоб перевести усі точки, координати для яких вказано у широті і довготі, у координати двовимірної площини.

Для генерування доріг і водойм які позначені незамкнутим полігоном програма малює отримані лінії у відповідні масиви. Для доріг малювання виконується з певним радіусом для пера, яке малює лінію, у двовимірний масив доріг. Для водойм, які позначені лінією, а не замкнутим полігоном, радіус для малювання дорівнює одиниці.

Для генерування типів місцевостей програма малює полігон у двовимірний масив, що покриває дану місцевість.

Для генерування будівель і водойм які позначені замкнутим полігоном програма використовує дані рельєфу для визначення оптимальних висот водойм та будівель, після чого програма малює полігони у відповідні двовимірні масиви у вигляді вертикальних ліній з блоків певного типу. Для будівель тип блоків налаштовує користувач, а для водойм тип блоків завжди дорівнює блоку «Вода».

Для визначення відстані у кілометрах між двома точками дані для яких вказано у широті і довготі буде застосовано формулу гаверсинуса [2] (2.1).

$$d = 2 \cdot r \cdot arcsin\left(\sqrt{sin^2\left(\frac{lat_2 - lat_1}{2}\right) + cos(lat_1) \cdot sin^2\left(\frac{lon_2 - lon_1}{2}\right)}\right) \quad (2.1)$$

d – відстань між двома точками у кілометрах, lat_1 - широта першої точки, lat_2 - широта другої точки, lon_1 - довгота першої точки, lon_2 - довгота другої точки, r - приблизний радіус Землі.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ СТВОРЕННЯ ІГРОВИХ СВІТІВ З КАРТ OPENSTREETMAP

Для створення додатку було вирішено використати наступні бібліотеки що доступні для використання мовою Java:

- J2Blocks;
- Geotools.

Бібліотека J2Blocks надає можливості для створення ігрових світів для гри Minecraft і встановлення довільних блоків у довільні точки у світі гри Minecraft,

Geotools - це відкрита бібліотека для геопросторового аналізу та обробки геоданих у програмному середовищі Java. Вона надає розширені функції для роботи з географічними об'єктами, картографією, просторовими операціями та взаємодією з різноманітними форматами геоданих. Geotools підтримує багато різних форматів геоданих, включаючи растрові та векторні дані. З його допомогою можна зчитувати та записувати файли, такі як Shapefile, GeoTIFF, GML, KML, PostGIS та багато інших. При створенні даного застосунку бібліотеку було використано для зчитування файлів рельєфу формату GeoTIFF

Для зчитування даних з файлів osm було вирішено створити власний аналізатор файлів osm без застосування сторонніх бібліотек.

Для застосунку було розроблено внутрішню архітектуру у вигляді UML діаграми класів (Рис 3.1) та ескіз інтерфейсу програми (рис. 3.2).

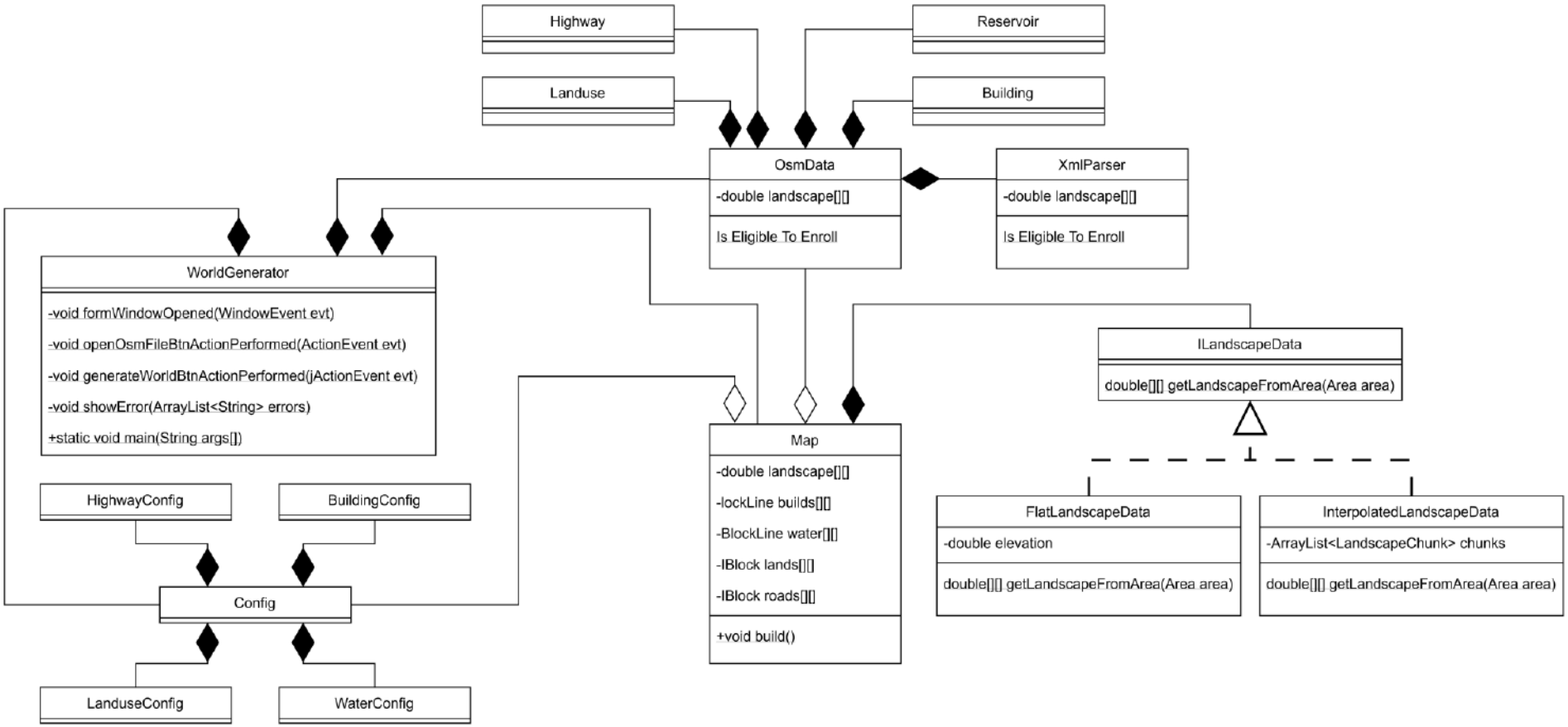

Рисунок 3.1 – UML діаграма базової архітектури додатку

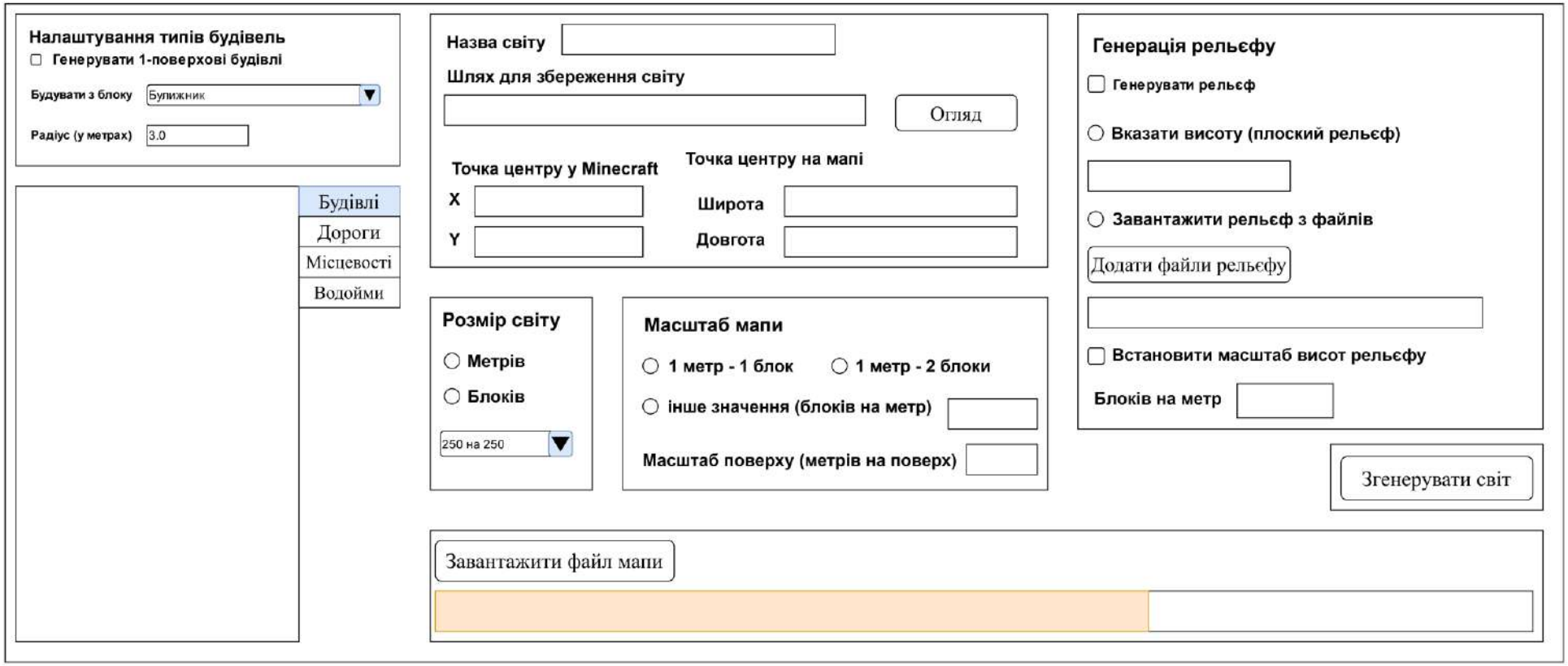

Рисунок 3.2 – Ескіз графічного інтерфейсу для додатку

Треба зазначити, що на діаграмі зображено тільки основні класи і основні властивості класів, більш детальний опис класів, їх властивостей і методів наведено далі.

Клас MapMath має статичні функції для здійснення певних математичних операцій які буде застосовано у розрахунках.

Функція lonRLatRToXYZ(Vector2D point) перетворює координати точки вказаної у широті і довготі у радіанах до точки тривимірного простору декартової системи координат покладаючи дану точку на сферу з радіусом який дорівнює одиниці.

Функція XYZToLonRLatR(Vector3D point) перетворює точку, яка лежить на сфері з одиничним радіусом у декартовій системі координат до системи координат широти і довготи, при цьому значення точки будуть вказані у радіанах.

Функція getCenter(Vector3D point1, Vector3D point2) повертає центральну точку між двома точками.

Функція pointToRadians(Vector2D point) перетворює значення у точці з градусів до радианів, не змінює початкову точку, результат повертає у іншому об’єкті.

Функція pointToDegrees(Vector2D point) перетворює значення у точці з радианів до градусів, не змінює початкову точку, результат повертає у іншому об’єкті.

Функція getAnotherPoint(Vector3D center, Vector3D point, double z) використовується для знаходження точки, яка лежить на колі яке має точку point і лежить у площині паралельній до площини XY, і відстань від якої до точки center така сама як відстані від точки point до точки center.

Функція solveSquareEq(double a, double b , double c) використовується для знаходження коренів квадратного рівняння, повертає масив з двох значень, функція є приватною і використовується іншими публічними функціями класу.

Функція getDistance(Vector2D pointRadians1, Vector2D pointRadians2) повертає відстань у метрах між двома точками на поверхні Землі.

Функція getXYFromLonLat(Vector2D point, Vector2D pointAR, Vector2D

pointBR, Vector2D pointCR, Vector2D pointDR) перетворює координати вказані у широті і довготі в градусах до координат на площині у метрах. Функція приймає у параметрах точку point яку потрібно перевести і точки які є чотирма кутами трапеції території, на якій знаходиться точка point. Для чотирьох кутів широту і довготу вказано в радіанах і точка pointAR означає точку з координатами (0, 0).

Функція getXYFromLonLatRad(Vector2D point, Vector2D pointAR, Vector2D pointBR, Vector2D pointCR, Vector2D pointDR) перетворює координати вказані у широті і довготі в радіанах до координат на площині у метрах. Функція приймає у параметрах точку point яку потрібно перевести і точки які є чотирма кутами трапеції території, на якій знаходиться точка point. Для чотирьох кутів широту і довготу вказано в радіанах і точка pointAR означає точку з координатами (0, 0).

Функція isInside(Vector2D pointRadians, Vector2D pointRadiansA, Vector2D pointRadiansB, Vector2D pointRadiansC, Vector2D pointRadiansD) перевіряє, чи знаходиться точка pointRadians у трапеції, яку визначають інші чотири точки передані у параметрах. Значення для усіх точок вказано у широті і довготі в радіанах.

Функція getHighOfTriangeB(double a, double b, double c), приймає довжини трьох сторін трикутника, повертає значення висоти трикутника проведеної до сторони b, функція є приватною і використовується іншими публічними функціями класу.

Функція getHypotenuse(double a, double b) приймає довжини катетів прямокутного трикутника і повертає довжину гіпотенузи.

Функція getInterpolatedValue(double[] points, double x) повертає значення біквадратичної інтерполяції для точки x між чотирма точками з масиву points.

Функціональний інтерфейс PointSetter має опис для методу setPoint(int x, int y), реалізація якого передбачає встановлення точки за координатами. Куди саме повинно бути встановлено точку — передбачає реалізація методу.

Клас MapUtill зберігає загальні статичні функції, які реалізують стандартні алгоритми для генерування полігонів і ліній.

Функція drawPolygon(ArrayList<Vector2D> polygon, PointSetter function)

генерує полігон на площині точки для якого задано у списку polygon. Для встановлення точки у певній області (наприклад, масиві) застосовує функціональний інтерфейс, який передано у параметрах з ім’ям function.

Функція drawLine(int x1, int y1, int x2, int y2, PointSetter function) генерує відрізок на площині, точки початку і закінчення відрізку приймає у параметрах. Для встановлення точки у певній області (наприклад, масиві) застосовує функціональний інтерфейс, який передано у параметрах з ім’ям function.

Функція drawLineToArray(int x1, int y1, int x2, int y2, int array[][]) генерує відрізок у масиві, точки початку і закінчення відрізку приймає у параметрах. Значення для точок у масиві які належать відрізку прирівнює до одиниці.

Клас Vector2D призначений для збереження даних для двовимірного вектору, який описаний двома дробовими числами. Клас має методи для встановлення і отримання значень вектору.

Клас Vector3D призначений для збереження даних для тривимірного вектору, який описаний трьома дробовими числами. Клас має методи для встановлення і отримання значень вектору.

Клас Area описує певну територію, її ширину і довжину. Клас приймає у конструкторі дві точки які є протилежними кутами географічної території на поверхні Землі, координати для яких вказано у широті і довготі в градусах і параметр, що вказує скільки повинно бути блоків у одному метрі. Клас має наступні методи.

Метод setWidthAndHeight() є приватним методом і використовується для встановлення значень ширини і довжини для території.

Метод setPoints() є приватним і використовується для розрахунку значень для усіх точок класу.

Метод getXYFromLonLatRad(Vector2D lonLatPoint) перетворює координати вказані у широті і довготі в градусах до координат на площині у метрах. Функція приймає у параметрах точку lonLatPoint яку потрібно перевести. Перша точка для території яку описує даний клас означає точку з координатами (0, 0) на площині.

Метод getPointA() повертає першу точку у широті і довготі в градусах для території яку описує даний клас.

Метод getPointB() повертає другу точку у широті і довготі в градусах для території яку описує даний клас.

Метод getPointC() повертає третю точку у широті і довготі в градусах для території яку описує даний клас.

Метод getPointD() повертає четверту точку у широті і довготі в градусах для території яку описує даний клас.

Метод getPointRadiansA() повертає першу точку у широті і довготі в радіанах для території яку описує даний клас.

Метод getPointRadiansB() повертає другу точку у широті і довготі в радіанах для території яку описує даний клас.

Метод getPointRadiansC() повертає третю точку у широті і довготі в радіанах для території яку описує даний клас.

Метод getPointRadiansD() повертає четверту точку у широті і довготі в радіанах для території яку описує даний клас.

Метод getPointXYZA() повертає першу точку яка лежить на одиничній сфері у тривимірному просторі декартової системи координат, для території яку описує даний клас.

Метод getPointXYZB() повертає другу точку яка лежить на одиничній сфері у тривимірному просторі декартової системи координат, для території яку описує даний клас.

Метод getPointXYZC() повертає третю точку яка лежить на одиничній сфері у тривимірному просторі декартової системи координат, для території яку описує даний клас.

Метод getPointXYZD() повертає четверту точку яка лежить на одиничній сфері у тривимірному просторі декартової системи координат, для території яку описує даний клас.

Метод getWidth() повертає ширину території округлену до цілого.

Метод getHeight() повертає довжину території округлену до цілого.

Метод getWidthD() повертає точне значення ширини території.

Метод getHeightD() повертає точне значення довжини території.

Метод getScale() повертає значення того, скільки повинно бути блоків на метр для території яку описує даний клас.

Клас BlockLine описує вертикальну лінію з блоків і має два конструктори, перший конструктор приймає інтерфейс який описує блок і значення рівня підйому, другий конструктор приймає ті самі значення і додатково значення висоти. Клас має методи що повертають інтерфейс який описує блок, значення рівня підйому і значення висоти.

Клас Blocks використовується для опису блоків і призначений для того щоб повертати інтерфейс який описує блок за назвою блоку. Клас має два статичні методи.

Метод getBlock(String name) призначений для отримання інтерфейсу який описує блок за іменем блоку у Minecraft.

Метод Set<String> getBlockNames() призначений для отримання множини імен доступних блоків.

Клас Building призначений для опису будівлі. Даний клас має полігон, який описує область яку покриває будівля на мапі, також у цьому класі вказано кількість поверхів для будівлі. Клас має методи для повернення його властивостей і конструктор для встановлення цих властивостей.

Клас Highway призначений для опису дороги. Даний клас має лінію з точок, яка описує дорогу на мапі. Клас має методи для повернення його властивостей і конструктор для встановлення цих властивостей.

Клас Landuse призначений для опису території. Даний клас має полігон, який описує область яку покриває територія на мапі. Клас має методи для повернення його властивостей і конструктор для встановлення цих властивостей.

Клас Reservoir призначений для опису водойм. Даний клас зберігає набір точок які можуть бути як полігоном, так і лінією. Даний набір точок описує водойму на мапі. Також в класі є властивість type яка вказує на те чи є ця водойма територією покритою полігоном або річкою чи джерелом води (струмком). Клас

має методи для повернення його властивостей і конструктор для встановлення цих властивостей.

Класи BuildingConfig, HighwayConfig, LanduseConfig і WaterConfig слугують для збереження властивостей для об’єктів певних типів, усі ці класи мають методи для отримання властивостей.

Клас BuildingConfig описує налаштування для будівель певного типу. Властивість класу isNeeded вказує на те чи треба будувати будівлі даного типу, властивість класу block вказує на те з якого блока треба будувати будівлі даного типу.

Клас HighwayConfig описує налаштування для доріг певного типу. Властивість класу isNeeded вказує на те чи треба будувати дороги даного типу, властивість класу radius вказує радіус у метрах для доріг даного типу, властивість класу block вказує на те з якого блока треба будувати дороги даного типу.

Клас LanduseConfig описує налаштування для територій певного типу. Властивість класу isNeeded вказує на те чи треба будувати територій даного типу, властивість класу block вказує на те з якого блока треба будувати територію даного типу.

Клас WaterConfig описує налаштування для водойми певного типу. Властивість класу isNeeded вказує на те чи треба будувати водойму даного типу, властивість класу block вказує на те з якого блока треба будувати водойми даного типу. Треба зазначити, що у поточній версії програми усі водойми за замовчуванням заповнюються водою, властивість block додано з припущення про можливість розширення функціоналу програми.

Клас Config містить загальні налаштування для генерування світу Minecraft, серед яких: назва світу, шлях для збереження світу, шлях до файлу osm, шлях до файлів GeoTIFF, масштаб (скільки блоків на метр требабудувати у світі), масштаб поверхів (скільки метрів займає один поверх), масштаб висот рельєфу, координати точки центу у світі Minecraft, координати точки центру на мапі реального світу (у широті і довготі в градусах), розмір території яку потрібно згенерувати, одиниця розміру території яку потрібно хгенерувати (блоки або

метри). Також клас містить множини з класами BuildingConfig, HighwayConfig, LanduseConfig та WaterConfig, і по одному екземпляру з цих класів як налаштування за замовчуванням для генерації об’єктів заданих типів. Для отримання усіх налаштувань для кожної змінної класу присутній метод для отримання її значення.

Інтерфейс ILandscapeData має один метод getLandscapeFromArea(Area area) який приймає екземпляр класу що описує територію і повертає масив значень висот для рельєфу.

Клас FlatLandscapeData реалізує інтерфейс ILandscapeData. Даний клас генерує плоский рельєф для світу Minecraft з певною висотою значення для якої приймає у конструкторі. Також у класі присутня реалізація методу getLandscapeFromArea(Area area) з інтерфейсу ILandscapeData.

Клас InterpolatedLandscapeData реалізує інтерфейс ILandscapeData. Даний клас має два конструктори, один з яких приймає у параметрах шлях до директорії з файлами GeoTIFF, інший приймає той самай параметр і додатково значення для масштабу рельєфую. Цей клас призначений для отримання значень рельєфу з файлів GeoTIFF для певної території і має наступні методи.

Метод setChunks(String path) є приватним методом і застосовується для завантаження інформації з усіх файлів формату GeoTIFF які знаходяться у директорії шлях до якої було передано у параметрі.

Метод getElevation(double lon, double lat) повертає значення висоти рельєфу для вказаної точки у довготі і широті.

Додатково у класі присутня реалізація методу getLandscapeFromArea(Area area) з інтерфейсу ILandscapeData.

Клас LandscapeChunk утримує інформацію щодо конкретного файлу формату GeoTIFF, даний клас має наступні методи.

Метод getElevation(double lon, double lat) повертаж значення висоти для певної точки, координати для якої вказано у довготі та широті, даний метод також може генрувати виключення, якщо точка виходить ща межу області отримання даних для рельєфу.

Метод getLonResolution() повертає відношення кількості градусів довготи до ширини зображення.

Метод getLatResolution() повертає відношення кількості градусів широти до висоти зображення.

Метод getLonResolutionRadians() повертає відношення кількості радіан довготи до ширини зображення.

Метод getLatResolutionRadians() повертає відношення кількості радіан широти до висоти зображення.

Метод getChunkWidth() повертає кількість точок для даних рельєфу по довготі.

Метод getChunkHeight() повертає кількість точок для даних рельєфу по широті.

Метод getDegWidth() повертає кількість градусів довготи для зони для якої вказано дані для рельєфу.

Метод getDegHeight() повертає кількість градусів широти для зони для якої вказано дані для рельєфу.

Метод getRadWidth() повертає кількість радіан довготи для зони для якої вказано дані для рельєфу.

Метод getRadHeight() повертає кількість радіан широти для зони для якої вказано дані для рельєфу.

Клас Map є дуже важливим класом, він містить методи призначені для побудови об’єктів у світі Minecraft і має у властивостях екземпляр класу Config для опису поточних налаштувань, екземпляр класу OsmData для отримання даних з карт, екземпляр класу який реалізує інтерфейс ILandscapeData для отримання даних рельєфу і екземпляр класу Area який описує масштаби території і призначений для перетворення координат точок з широти і довготи до координат поточної території. Також даний клас має у властивостях двовимірні масиви для висот рельєфу, для вертикальних ліній з блоків що описують будівлі та водойми, для блоків що описують дороги і місцевості. Клас має наступні методи.

Метод setBlock(int x, int y, int z, IBlock block, World world) призначений для встановлення блоку у світі Minecraft, головна мета методу це перевірити те, що блок знаходиться у межах території мапи і не перевищує певні діапазони у координатах, також метод забезпечує зміщення блоків у просторі відповідно до налаштувань центру мапи у екземплярі класу Config.

Метод build() призначений для заповнення світу блоками з масивів які описують висоти рельєфу, вертикальні ліній з блоків що описують будівлі та водойми, блоки що описують дороги і місцевості. Після заповнення світу блоками метод зберігає світ у директорію відповідно до налаштувань.

Метод fillReservoirs() призначений для почергового заповнення світу водоймами кожного типу, завантажує налаштування для кожного типу водойми, після чого викликає метод drawReservoir для водойми поточного типу.

Метод fillLanduses() призначений для почергового заповнення світу місцевостями кожного типу, завантажує налаштування для кожного типу місцевості, після чого викликає метод drawLanduse для місцевості поточного типу.

Метод fillRoads() призначений для почергового заповнення світу дорогами кожного типу, завантажує налаштування для кожного типу дороги, після чого викликає метод drawRoad для дороги поточного типу.

Метод fillBuildings() призначений для почергового заповнення світу будівлями кожного типу, завантажує налаштування для кожного типу будівлі, після чого викликає метод drawBuilding для будівлі поточного типу.

Метод setRoadPoint(int x, int y, IBlock block) призначений для встановлення блока у двовимірний масив призначений для відображення блоків доріг у світі.

Метод setPointOnArray(int x, int y, IBlock array[][], IBlock block) призначений для встановлення блока у будь-який масив блоків, метод додатково перевіряє те, щоб координати блока входили у межі масиву.

Метод drawCircle(float x, float y, float radius, IBlock block) призначений для встановлення кола з блоків у двовимірний масив призначений для відображення блоків доріг у світі.

Метод drawRoad(Highway road, float radius, IBlock block) призначений для заповнення масиву блоків для доріг яку передано у параметрах під назвою road, блоками що належать поточній дорозі, блоками з типом параметру block і радіусом для дороги з параметром radius.

Метод drawRoadLine(float x1, float y1, float x2, float y2, float radius, IBlock block) призначений для заповнення масиву блоків для доріг блоками що належать певному відрізку дороги, координати початку і кінця якого передано у параметрах.

Метод drawReservoir(Reservoir reservoir) призначений для заповнення масиву вертикальних ліній з блоків для водойм яку передано у параметрах під назвою reservoir, блоками з типом параметру block.

Метод getOptimalElevationForBuilding(int xBias, int yBias, int area[][]) приймає у параметрах масив, що визначає маску певної території, яку займає будівля, також приймає два числових значення, що визначають зміщення цієї маски у світових координатах. Метод повертає оптимальну висоту на рельєфі для нижньої точки будівлі.

Метод getOptimalElevationForReservoir(ArrayList<Vector2D> polygon) повертає оптимальну висоту для рівня водойми над рельєфом.

Метод drawLanduse(Landuse landuse, IBlock block) призначений для заповнення масиву блоків для територій яку передано у параметрах під назвою landuse, блоками з типом параметру block,

Метод drawBuilding(Building building, int levels, IBlock block) призначений для заповнення масиву вертикальних ліній з блоків для будівель яку передано у параметрах під назвою building, блоками з типом параметру block.

Метод drawReservoirLine(int x1, int y1, int x2, int y2, BlockLine blockLine) призначений для заповнення відрізку у масиві вертикальних ліній з блоків для водойм, блоками з типом параметру blockLine. Координати почату і закінчення відрізку передано у параметрах.

Метод setWater(int x, int y, BlockLine blockLine) призначений для встановлення вертикальної лінії з блоків у масив вертикальних ліній з блоків для

водойм. Перед встановленням метод перевіряє те, що координати лінії з блоків у масиві не виходять за межі масиву.

Метод setSaveProcess(ProgressWithComment saveProcess) встановлює екземпляр класу який реалізує інтерфейс ProgressWithComment для передавання даних щодо прогресу збереження світу Minecraft у директорію.

Метод setBuildProcess(ProgressWithComment buildProcess) встановлює екземпляр класу який реалізує інтерфейс ProgressWithComment для передавання даних щодо прогресу побудовни світу Minecraft.

Клас Node зберігає широту, довготу і ідентифікатор точки на мапі.

Клас Way зберігає список точок Node і список тегів для поточного екземпляру класу, також клас зберігає ідентифікатор даного списку точок на мапі.

Клас Relation зберігає список екземплярів класу Way (списків точок на мапі) і список дочірніх класів Relation, також клас зберігає ідентифікатор.

Інтерфейс Progress є функціональним інтерфейсом і призначений для отримання даних про кількість відсотків готового прогресу для певної дії.

Інтерфейс ProgressWithComment є функціональним інтерфейсом і призначений для отримання даних про кількість відсотків готового прогресу і коментаря щодо готового прогресу для певної дії.

Клас NodeException призначений для описання помилок що можуть виникнути при встановленні параметрів для класу Node.

Клас WayException призначений для описання помилок що можуть виникнути при встановленні параметрів для класу Way.

Клас XmlParseException призначений для опису помилок що можуть виникнути при аналізі тексту який має структуру xml.

Клас OsmDataException призначений для описання помилок що можуть виникнути при аналізі інформації з файлу osm.

Клас SaveWorldException призначений для описання помилок що можуть виникнути при спробі збереження світу Minecraft.

Клас XmlNode призначений для зберігання даних вузла у структурі xml. Клас має наступні методи.

Метод parseFromText(StringBuilder text, Progress lambda) призначений для встановлення даних для поточного вузла структури xml з тексту який передано у параметрі text і встановленні прогресу у екземпляр анонімного класу, який реалізує інтерфейс Progress.

Метод parseFromText(StringBuilder text, Progress lambda, int allLength) є приватним методом і викликається методом parseFromText(StringBuilder text, Progress lambda), даний метод також призначений для встановлення даних для поточного вузла структури xml з тексту який передано у параметрі text і встановленні прогресу у екземпляр анонімного класу, який реалізує інтерфейс Progress, але даний метод приймає параметр що визначає кількість символів у всьому тексті, який було передано до першого вузла структури xml.

Метод countInc(int allLength, Progress lambda) призначений для розрахунку поточного прогресу розбору структури xml у відсотках.

Метод getChildren(String name) призначений для отримання усіх дочірніх вузлів поточного вузла у вигляді списку які мають назву name.

Метод getAllChildren() призначений для отримання усіх дочірніх вузлів поточного вузла у вигляді списку.

Метод getParams() призначений для отримання усіх параметрів поточного вузла у вигляді хеш-мапи.

Метод getParam(String key) призначений для отримання значення параметра за назвою key.

Метод getName() призначений для отримання назви поточного вузла.

Метод getContent() призначений для отримання текстових даних поточного вузла, які не були використані у формуванні структури xml.

Клас OsmParser призначений для отримання даних з файлу osm та має конструктор, який приймає екземпляр класу XmlNode, що описує головний вузол структури xml файлу osm. Також клас має наступні методи.

Метод getObjectsByName(String name) призначений для отримання об’єктів мапи за назвою об’єкта.

Метод getRelationsByType(String type) призначений для отримання об’єктів

«relation» з типом type.

Метод getMembersByRole(XmlNode relation, String role) призначений для отримання з об’єкта «relation» усіх дочірніх об’єктів з типом «member» які мають роль role.

Метод getWaysByTagKey(String key) повертає список об’єктів «way» які позначені тегом зі значенням key.

Метод getObjectById(String id) повертає об’єкт за ідентифікатором з головного вузла структури xml.

Метод getRoot() повертає головний вузол структури xml.

Клас WorldGenerator є головним класом що має метод main. Клас призначений для відображення інтерфейсу і забезпечення взаємодії з користувачем. Важливі методи класу наведено далі.

Метод initComponents() виконується при створенні екземпляру класу і призначений для ініціалізації компонентів інтерфейсу.

Метод main(String args[]) є точкою входу до виконання програми.

Метод openOsmFileBtnActionPerformed(java.awt.event.ActionEvent evt) спрацьовує при натисканні на кнопку «Завантажити файл мапи» і призначений для завантаження файлу мапи і виведення поточного прогресу завантаження на форму.

Основний метод для перевірки введених параметрів і заповнення екземпляру класу Config з подальшим генеруванням і збереженням світу має назву generateWorldBtnActionPerformed(java.awt.event.ActionEvent evt).

РОЗДІЛ 4
ТЕСТУВАННЯ ДОДАТКУ ДЛЯ СТВОРЕННЯ ІГРОВИХ СВІТІВ

4.1 Загальне тестування

Для загального тестування було обрано і завантажено карту карту міста Трускавець на офіційному сайті проєкту Openstreetmap (рис. 4.1), також було завантажено файли для рельєфу поверхні землі, отриманих під час місії SRTM з роздільною здатністю близько 90 метрів. Після чого було завантажено файл міста Трускавець до додатку, процес завантаження і аналізу зображено на рисунку 4.2. Далі було проведено тестування програми з загальними налаштуваннями, значення для яких вказано у таблиці 4.1, з значеннями для типів будівель вказаними у таблиці 4.2, з значеннями для типів доріг вказаними у таблиці 4.3 Для усіх типів місцевостей налаштування було залишене за замовчуванням. Для усіх типів водойм було встановлено налаштування, що водойму даного типу треба генерувати. Скріншот додатку з усіма встановленими налаштуваннями наведено на рисунку 4.3.

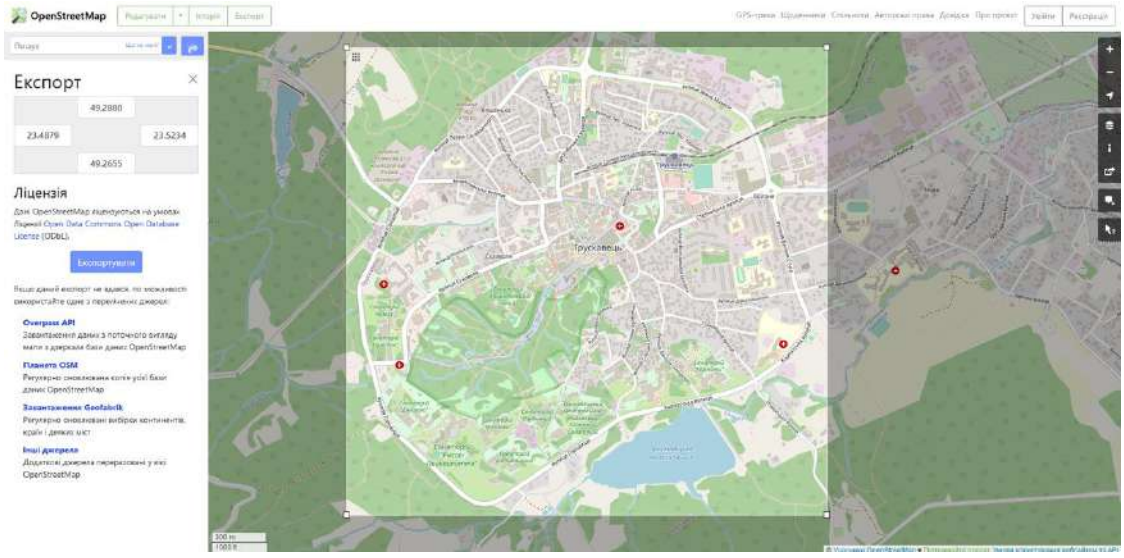

Рисунок 4.1 – Експорт даних мапи для міста Трусквець

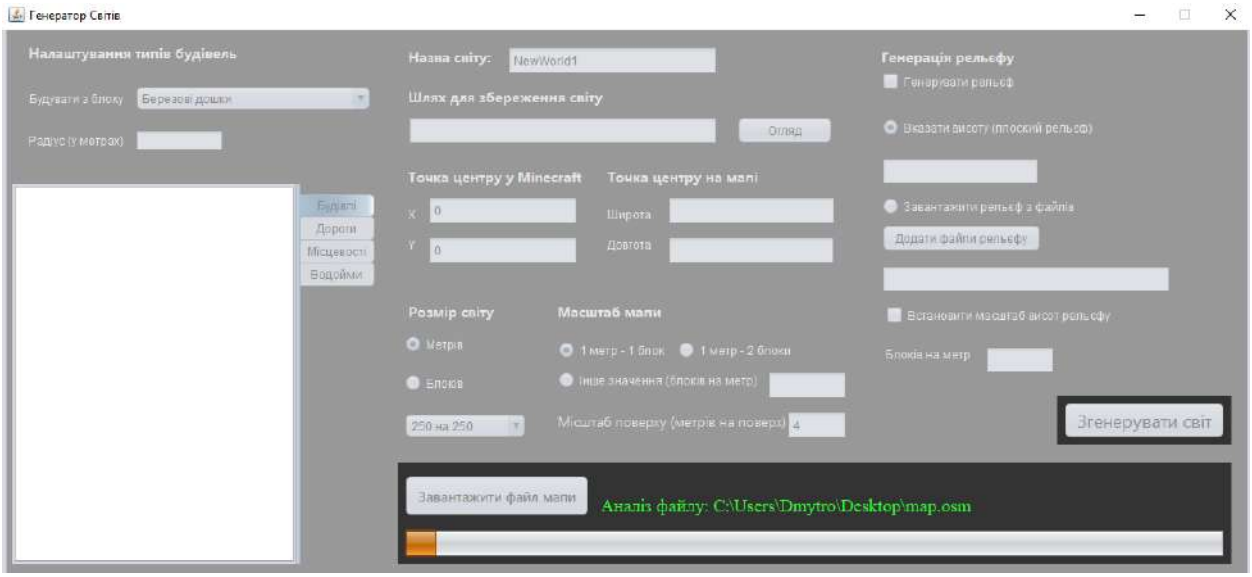

Рисунок 4.2 – Аналіз даних з файла мапи

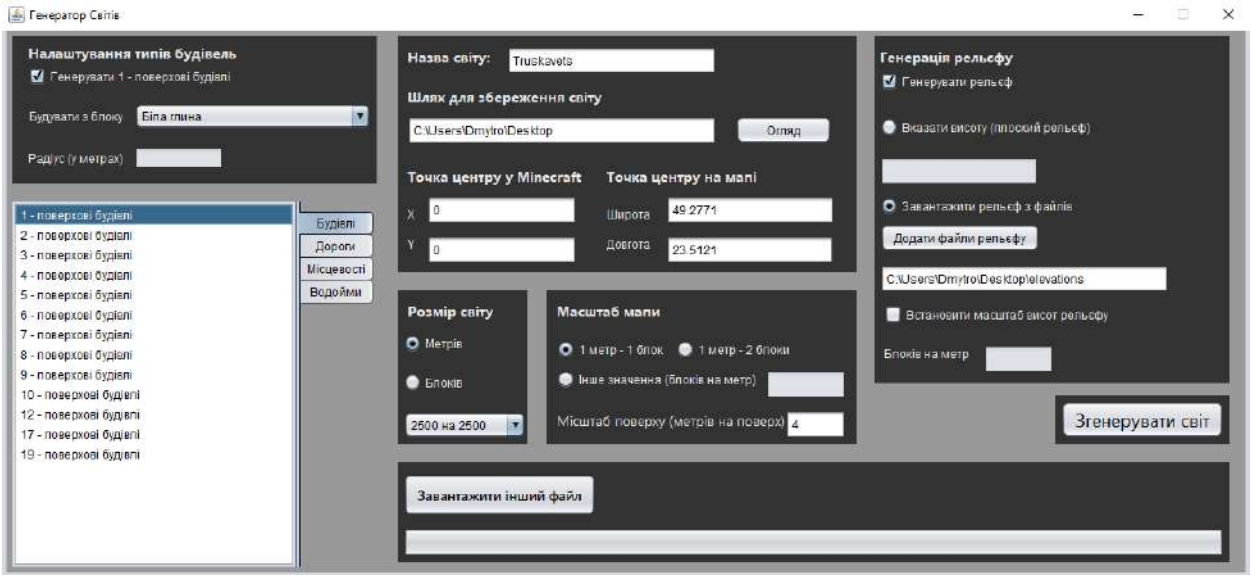

Рисунок 4.3 – Встановлені налаштування

Таблиця 4.1 – Значення загальних налаштувань

Назва налаштування	Значення
Назва світу	Truskavets
Шлях для збереження світу	C:\Users\Dmytro\Desktop
Точка центру у Minecraft (x)	0
Точка центру у Minecraft (y)	0
Точка центру на мапі (широта)	49.2771
Точка центру на мапі (довгота)	23.5121
Одиниця виміру розміру світу	Метр

Продовження таблиці 4.1

Назва налаштування	Значення
Розмір світу	2500 на 2500
Масштаб мапи	1 блок - 1 метр
Масштаб поверху (метрів на поверх)	4
Генерувати рельєф	Так
Спосіб отримання даних рельєфу	Завантажити рельєф з файлів
Шлях до директорії з файлами рельєфу	C:\Users\Dmytro\Desktop\elevations
Встановити масштаб висот рельєфу	Ні

Таблиця 4.2 – Значення налаштувань для кожного типу будівлі

Тип будівлі	Генерувати	Блок для генерації
1 - поверхові будівлі	Так	Біла глина
2 - поверхові будівлі	Так	Пісок
3 - поверхові будівлі	Так	Соснові дошки
4 - поверхові будівлі	Так	Блакитна глина
5 - поверхові будівлі	Так	Рожева шерсть
6 - поверхові будівлі	Так	Редстоунова руда
7 - поверхові будівлі	Так	Жовта шерсть
8 - поверхові будівлі	Так	Пурпурна шерсть
9 - поверхові будівлі	Так	Червона шерсть
10 - поверхові будівлі	Так	Коричнева глина
12 - поверхові будівлі	Так	Дошки з акації
17 - поверхові будівлі	Так	Лаймова шерсть
19 - поверхові будівлі	Так	Рожева глина

Таблиця 4.3 – Значення налаштувань для кожного типу дороги

Тип доріг	Генерувати	Блок для генерації	Радіус
Неспецифічні шляхи	Так	Каміння	3
Некласифіковані дороги	Так	Каміння	3
З’єднання з місцевими дорогами	Так	Чорна глина	3
Житлові дороги	Так	Чорна глина	3
Дороги службового призначення	Так	Каміння	3
Місцеві дороги	Так	Чорна глина	3
Автобусні дороги	Так	Каміння	3
Дороги що будуються	Так	Трава	3
Пішохідні дороги	Так	Гравій	3
Господарські дороги	Так	Земля	3
Сходи	Ні	-	-

Для продовження тестування було на тиснуто кнопку з написом «Згенерувати світ», після чого програма почала генерацію світу і виводила процес генерації (рис. 4.4). Після завершення генерації світу за вказаним шляхом з’явився каталог зі світом. Для перегляду результату генерації, було використано спеціальну програму яка відображає на екран світи гри Minecraft у високій якості. Відображення даного світу зображено на рисунках 4.5-4.8.

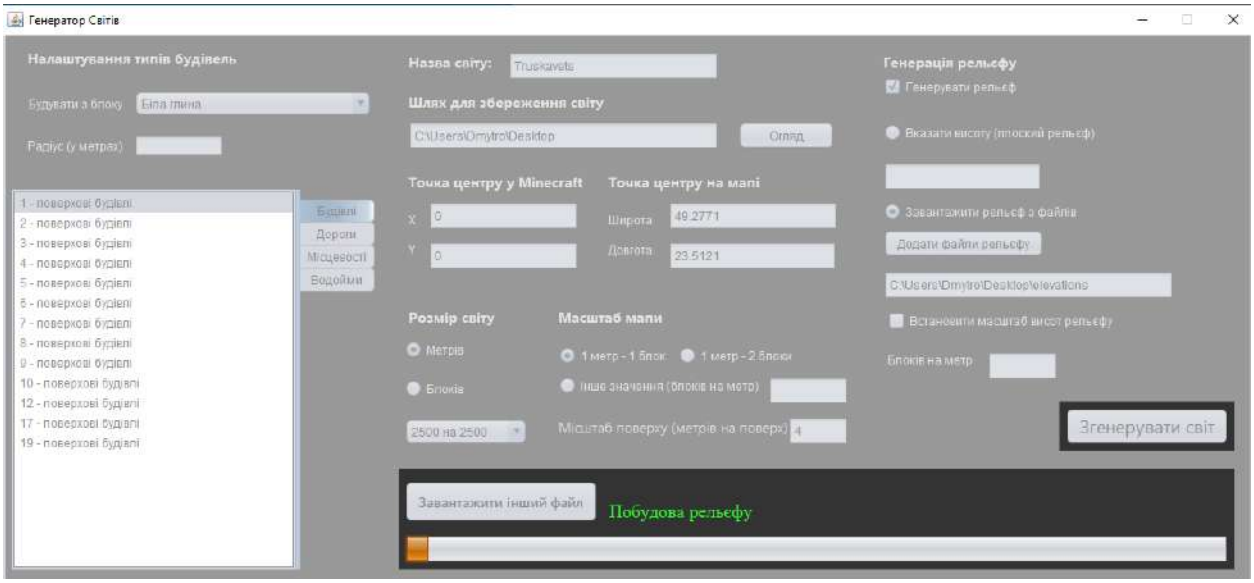

Рисунок 4.4 – Процес генерації світу

4.2 Тестування на обробку помилок

Для тестування на обробку помилок спочатку було завантажено файл мапи міста Трускавець у програму. Далі було обрано довільний тип дороги зі списку знайдених типів доріг і введено у поле для радіусу дороги некоректне значення, після введення некоректного значення поле одразу було підсвічено червоним кольором (рис. 4.5)

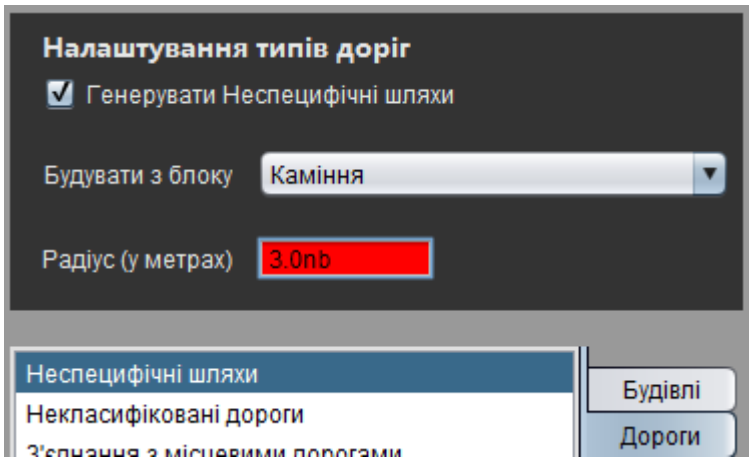

Рисунок 4.5 – Некоректне значення для радіусу типу дороги

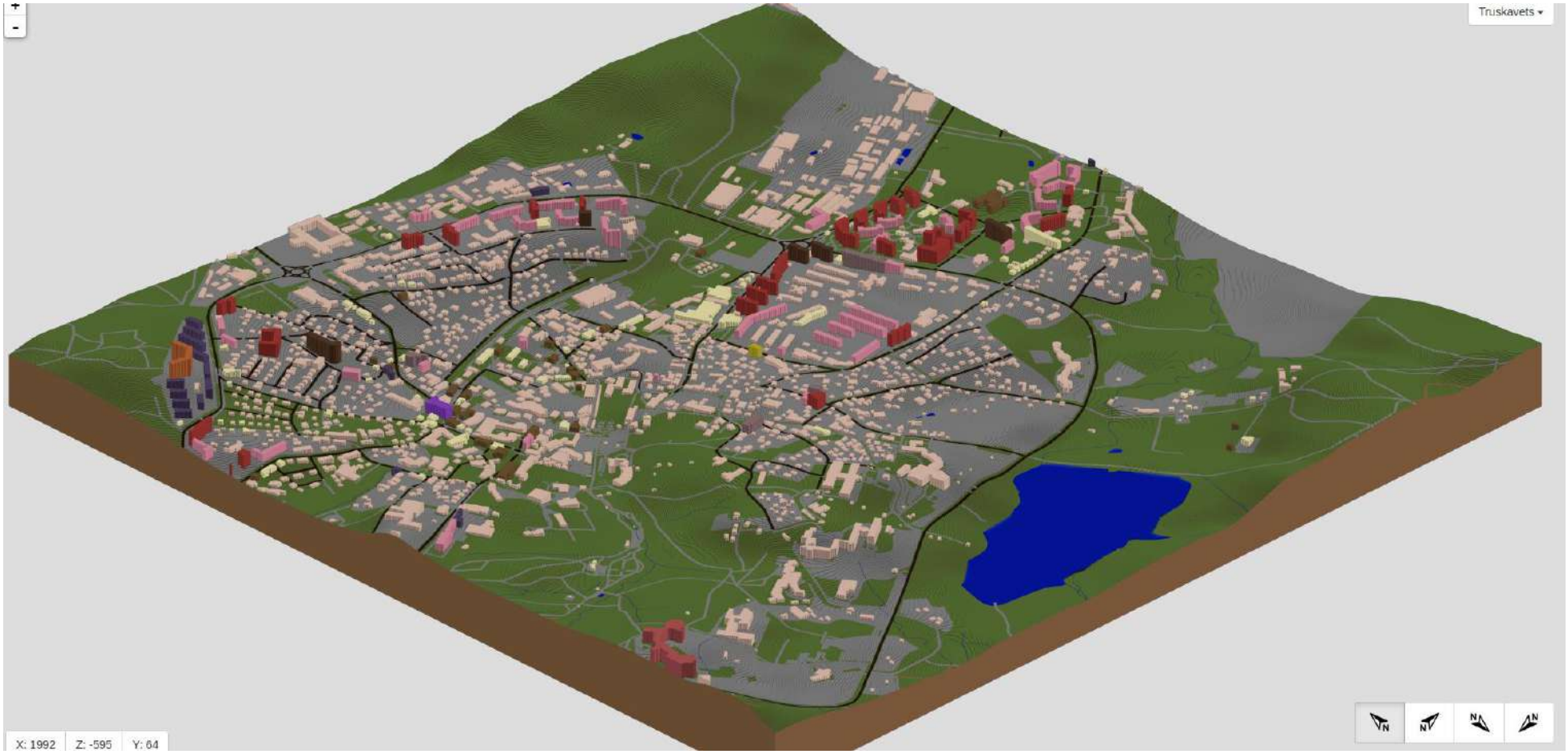

Рисунок 4.6 – Зображення згенерованого світу з Південно-Західного кута

Рисунок 4.6 – Зображення згенерованого світу з Південно-Східного кута

Рисунок 4.7 – Зображення згенерованого світу з Північно-Східного кута

Рисунок 4.8 – Зображення згенерованого світу з Північно-Західного кута

Для продовження тестування на обробку некоректно введених даних було введено в усі поля некоректні дані, після чого було натиснуто кнопку з написом «Згенерувати світ». Після натискання кнопки з’явилося вікно з вказанням на неправильно введені дані (рис. 4.9), також усі поля з некоректними значеннями було виділено червоним кольором (рис. 4.10).

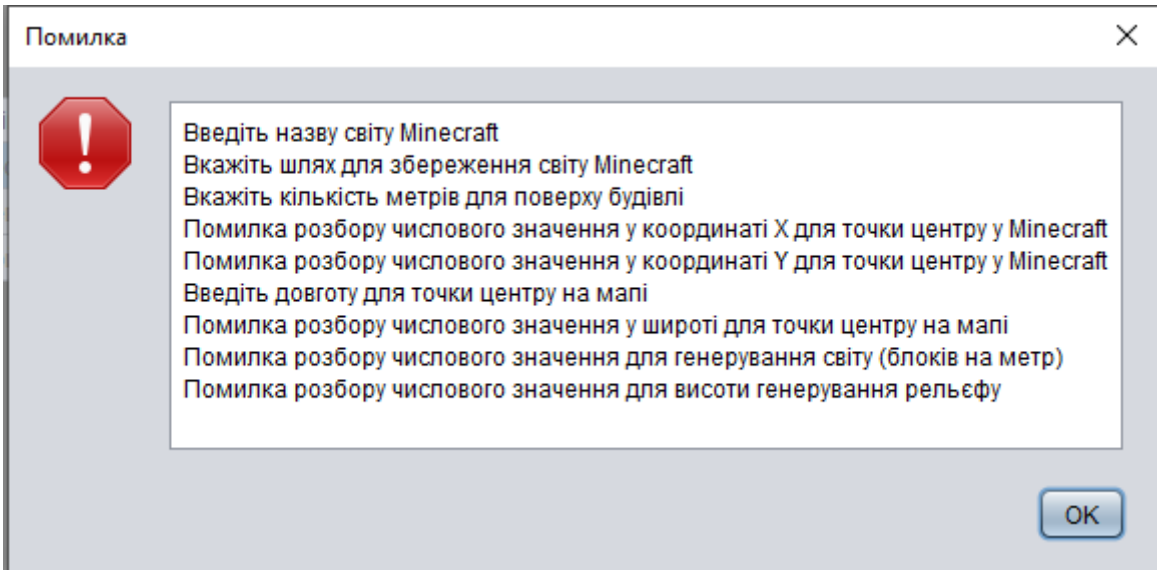

Рисунок 4.9 – Вікно з вказанням на неправильно введені дані

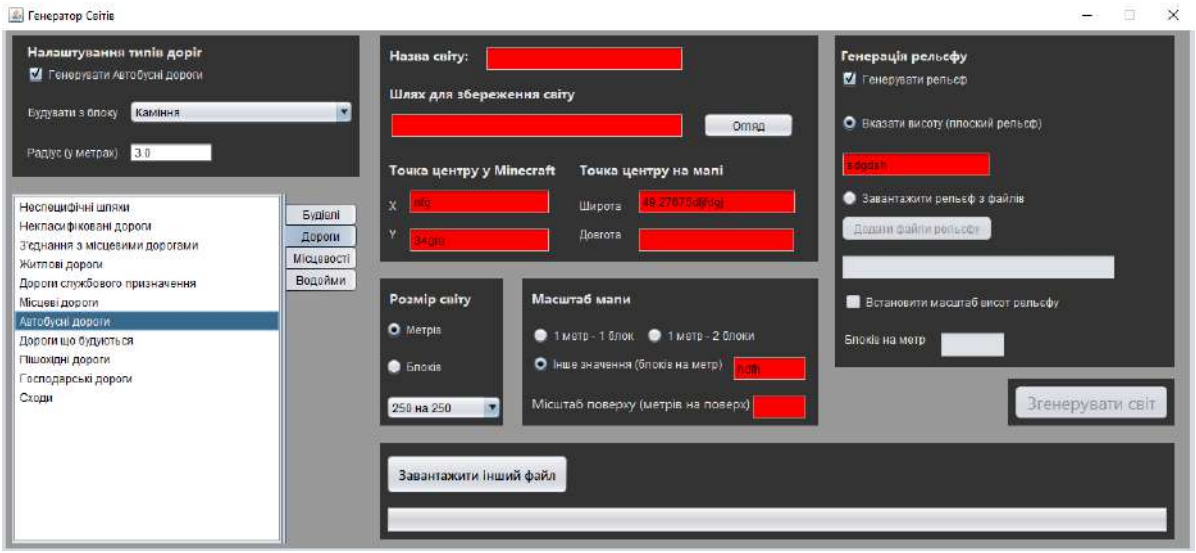

Рисунок 4.10 – Виділені червоним кольором поля з некоректними значеннями

Для завершення тестів обробки помилок, було створено копію файлу мапи міста Трускавець і спецально пошкоджено її. У копії файлу було спеціально змінено одне з коректних чисел на число що введено некоректно, далі було зроблено спробу завантажити даний файл для аналізу у програму. При спробі

аналізу файлу програма видала помилку яка вказує на неможливість проаналізувати файл (рис. 4.11).

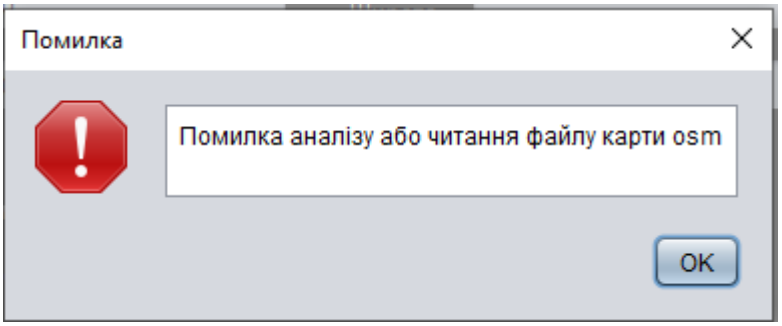

Рисунок 4.11 – Повідомлення про помилку аналізу або читання файлу osm

ВИСНОВКИ

Було проаналізовано доповнення які можуть допомогти генерувати ігрові світи для створення власних ігор або для серверу Minecraft і визначено їх переваги та недоліки. Додатково було сформовано мету і поставлено задачу розробити додаток для створення ігрових світів у Minecraft і створено детальний опис цього додатку, включаючи список налаштувань і список помилок, що можуть виникнути при використанні даного додатку. Для додатку було визначено що він буде брати інформацію з відкритих карт саме Openstreetmap та даних з карт отриманих під час місії SRTM.

Додаток було розроблено і протестовано на прикладах генерації певної території і на обробку помилок. Тестування показало, що додаток працює правильно і може обробляти помилки, інформуючи користувача яка саме причина була у виникненні помилки. Додатково додаток надає гнучкі налаштування для генерації світу.

Також даний додаток має можливості до покращення. До нього можна додати інші функції і зробити більш гнучким і оптимальним, можна застосувати механізми серіалізації, які присутні у Java, для зменшення обсягу оперативної пам’яті, яку додаток потребує для генерування особливо великих територій. Також можна додати додатковий функціонал, наприклад можна створити вікно, де користувач буде бачити територію, яку він завантажив у файлі osm, і зможе виділяти територію яку він саме хоче згенерувати. Не виключено покращення і у способі завантаження даних для генерації світу: можна зробити завантаження карт не тільки через файли, але і через інтернет, що покращать зручність використання.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ ТА ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Офіційний сайт Openstreetmap. [Електронний ресурс]. Режим доступу: https://wiki.openstreetmap.org/wiki/Main_Page. Дата доступу: 07.05.2023.

2. Формула гаверсинуса — Вікіпедія. [Електронний ресурс]. - Режим доступу: https://uk.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D0%BC%D1%83%D0%BB%D0%B0_%D0%B3%D0%B0%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%BD%D1%83%D1%81%D0%B0. - Дата доступу: 07.05.2023.

3. Документація бібліотеки J2Blocks [Електронний ресурс]. - Режим доступу: https://morbz.github.io/J2Blocks/index.html

4. Офіційний сайт Geotools [Електронний ресурс]. - Режим доступу: https://www.geotools.org/

5. Steven C. Chapra Numerical Methods for Engineers / Steven C. Chapra, Raymond P. Canale - McGraw-Hill Education, 2015. - 987 с.

6. Wolfgang Torge Geodesy / Wolfgang Torge, Jürgen Müller - De Gruyter , 2012. - 444 с.

Додаток А

Зауваження нормоконтролера

Таблиця А.1 — Зауваження нормоконтролера

Сторінка	Позначення	Зміст зауваження
Зміст		Службові слова РОЗДІЛ не потрібні
С. 9, 13-19		Невірні переліки
ПЗ		Нещільне заповнення сторінок
С. 51		Перелік джерел – не вірно оформлений , в ПЗ нема посилань на джерела

Додаток Б

Лістинг коду програми

Для файлу WorldGenerator.java

```
package org.minemapcreator.minemapcreator;
import java.awt.Color;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Set;
import javax.swing.DefaultListModel;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingWorker;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import org.geotools.data.DataSourceException;
import org.minemapcreator.minemapcreator.mapgenerator.Blocks;
import org.minemapcreator.minemapcreator.mapgenerator.BuildingConfig;
import org.minemapcreator.minemapcreator.mapgenerator.Config;
import org.minemapcreator.minemapcreator.mapgenerator.FlatLandscapeData;
import org.minemapcreator.minemapcreator.mapgenerator.HighwayConfig;
import org.minemapcreator.minemapcreator.mapgenerator.ILandscapeData;
import org.minemapcreator.minemapcreator.mapgenerator.InterpolatedLandscapeData;
import org.minemapcreator.minemapcreator.mapgenerator.LanduseConfig;
import org.minemapcreator.minemapcreator.mapgenerator.Map;
import org.minemapcreator.minemapcreator.mapgenerator.OsmData;
import org.minemapcreator.minemapcreator.mapgenerator.OsmDataException;
import org.minemapcreator.minemapcreator.mapgenerator.SaveWorldException;
import org.minemapcreator.minemapcreator.mapgenerator.Translator;
import org.minemapcreator.minemapcreator.mapgenerator.WaterConfig;
import org.minemapcreator.minemapcreator.math.Vector2D;
import org.minemapcreator.minemapcreator.xmlparser.XmlParseException;

public class WorldGenerator extends javax.swing.JFrame {

   public WorldGenerator() {
          initComponents();
   }

   @SuppressWarnings("unchecked")
   // <editor-fold defaultstate="collapsed" desc="Generated Code">
   private void initComponents() {

      sizeBtnGroup = new javax.swing.ButtonGroup();
      scaleBtnGroup = new javax.swing.ButtonGroup();
      landBtnGroup = new javax.swing.ButtonGroup();
      backgroundPanel = new javax.swing.JPanel();
```

```
listsTabs = new javax.swing.JTabbedPane();
buildingsTab = new javax.swing.JPanel();
buildingsListScrollPane = new javax.swing.JScrollPane();
buildingsList = new javax.swing.JList<>();
roadsTab = new javax.swing.JPanel();
roadsListScrollPane = new javax.swing.JScrollPane();
roadsList = new javax.swing.JList<>();
landusesTab = new javax.swing.JPanel();
landusesListScrollPane = new javax.swing.JScrollPane();
landusesList = new javax.swing.JList<>();
watersTab = new javax.swing.JPanel();
watersListScrollPane = new javax.swing.JScrollPane();
watersList = new javax.swing.JList<>();
jPanel1 = new javax.swing.JPanel();
progress = new javax.swing.JProgressBar();
openOsmFileBtn = new javax.swing.JButton();
statusLabel = new javax.swing.JLabel();
centerPropertiesPanel = new javax.swing.JPanel();
yField = new javax.swing.JTextField();
xField = new javax.swing.JTextField();
mapCenterMinecraft = new javax.swing.JLabel();
xLabel = new javax.swing.JLabel();
yLabel = new javax.swing.JLabel();
mapCenterLabel = new javax.swing.JLabel();
latField = new javax.swing.JTextField();
lonField = new javax.swing.JTextField();
setWorldPathField = new javax.swing.JTextField();
setPathForSaveBtn = new javax.swing.JButton();
setWorldNameLabel = new javax.swing.JLabel();
latLabel = new javax.swing.JLabel();
lonLabel1 = new javax.swing.JLabel();
setWorldPathLabel = new javax.swing.JLabel();
setWorldNameField = new javax.swing.JTextField();
scalePropertiesPanel = new javax.swing.JPanel();
sizeLabel = new javax.swing.JLabel();
sizeComboBox = new javax.swing.JComboBox<>();
metrsBtn = new javax.swing.JRadioButton();
blocksBtn = new javax.swing.JRadioButton();
oneMeterOneBlockBtn = new javax.swing.JRadioButton();
oneMeterTwoBlockBtn = new javax.swing.JRadioButton();
otherValueBtn = new javax.swing.JRadioButton();
scaleLabel = new javax.swing.JLabel();
levelsScaleField = new javax.swing.JTextField();
betweenPaneel = new javax.swing.JPanel();
otherValueField = new javax.swing.JTextField();
levelsScaleLabel = new javax.swing.JLabel();
landscapePanel = new javax.swing.JPanel();
elevationField = new javax.swing.JTextField();
enterElevationBtn = new javax.swing.JRadioButton();
addLandscapeFilesBtn = new javax.swing.JButton();
addLandscapePath = new javax.swing.JTextField();
loadLandscapeFromFileBtn = new javax.swing.JRadioButton();
landscapeScaleField = new javax.swing.JTextField();
generateLandscapeBackgroundPanel = new javax.swing.JPanel();
```

```
        generateLandscapeBtn = new javax.swing.JCheckBox();
        generateLandscapeLabel = new javax.swing.JLabel();
        landscapeScaleCheckbox = new javax.swing.JCheckBox();
        landscapeScaleLabel1 = new javax.swing.JLabel();
        btnGeneratePanel = new javax.swing.JPanel();
        generateWorldBtn = new javax.swing.JButton();
        typePanel = new javax.swing.JPanel();
        titleTypePanel = new javax.swing.JLabel();
        blockLabel = new javax.swing.JLabel();
        blockSelection = new javax.swing.JComboBox<>();
        roadRadius = new javax.swing.JTextField();
        radiusLabel = new javax.swing.JLabel();
        isNeeded = new javax.swing.JCheckBox();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Генератор Світів");
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowOpened(java.awt.event.WindowEvent evt) {
                formWindowOpened(evt);
            }
        });

        backgroundPanel.setBackground(new java.awt.Color(153, 153, 153));
        backgroundPanel.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));

        listsTabs.setBackground(new java.awt.Color(51, 51, 51));
        listsTabs.setForeground(new java.awt.Color(230, 230, 230));
        listsTabs.setTabPlacement(javax.swing.JTabbedPane.RIGHT);

        buildingsList.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
        buildingsListScrollPane.setViewportView(buildingsList);

        javax.swing.GroupLayout buildingsTabLayout = new javax.swing.GroupLayout(buildingsTab);
        buildingsTab.setLayout(buildingsTabLayout);
        buildingsTabLayout.setHorizontalGroup(
            buildingsTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(buildingsListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 284,
Short.MAX_VALUE)
        );
        buildingsTabLayout.setVerticalGroup(
            buildingsTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(buildingsListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 366,
Short.MAX_VALUE)
        );

        listsTabs.addTab("Будівлі", buildingsTab);

        roadsTab.setBackground(new java.awt.Color(224, 224, 224));

        roadsListScrollPane.setViewportView(roadsList);

        javax.swing.GroupLayout roadsTabLayout = new javax.swing.GroupLayout(roadsTab);
        roadsTab.setLayout(roadsTabLayout);
```

```
        roadsTabLayout.setHorizontalGroup(
            roadsTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(roadsListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 284, Short.MAX_VALUE)
        );
        roadsTabLayout.setVerticalGroup(
            roadsTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(roadsListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 366, Short.MAX_VALUE)
        );

        listsTabs.addTab("Дороги", roadsTab);

        landusesTab.setBackground(new java.awt.Color(224, 224, 224));

        landusesListScrollPane.setViewportView(landusesList);

        javax.swing.GroupLayout landusesTabLayout = new javax.swing.GroupLayout(landusesTab);
        landusesTab.setLayout(landusesTabLayout);
        landusesTabLayout.setHorizontalGroup(
            landusesTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(landusesListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 284,
Short.MAX_VALUE)
        );
        landusesTabLayout.setVerticalGroup(
            landusesTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(landusesListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 366,
Short.MAX_VALUE)
        );

        listsTabs.addTab("Місцевості", landusesTab);

        watersListScrollPane.setViewportView(watersList);

        javax.swing.GroupLayout watersTabLayout = new javax.swing.GroupLayout(watersTab);
        watersTab.setLayout(watersTabLayout);
        watersTabLayout.setHorizontalGroup(
            watersTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(watersListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 284,
Short.MAX_VALUE)
        );
        watersTabLayout.setVerticalGroup(
            watersTabLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(watersListScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 366,
Short.MAX_VALUE)
        );

        listsTabs.addTab("Водойми", watersTab);

        jPanel1.setBackground(new java.awt.Color(51, 51, 51));

        progress.setBackground(new java.awt.Color(255, 255, 255));
        progress.setForeground(new java.awt.Color(51, 51, 51));
        progress.setName(""); // NOI18N

        openOsmFileBtn.setFont(new java.awt.Font("Yu Gothic UI Semibold", 0, 14)); // NOI18N
```

```
        openOsmFileBtn.setText("Завантажити файл мапи");
        openOsmFileBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                openOsmFileBtnActionPerformed(evt);
            }
        });

        statusLabel.setFont(new java.awt.Font("Serif", 0, 18)); // NOI18N
        statusLabel.setForeground(new java.awt.Color(51, 255, 51));
        statusLabel.setToolTipText("");
        statusLabel.setMaximumSize(new java.awt.Dimension(79, 427));

        javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addComponent(openOsmFileBtn)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(statusLabel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(progress, javax.swing.GroupLayout.PREFERRED_SIZE, 824,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
            jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(12, 12, 12)
                .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(openOsmFileBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 42,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(statusLabel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(progress, javax.swing.GroupLayout.DEFAULT_SIZE, 29, Short.MAX_VALUE)
                .addContainerGap())
        );

        centerPropertiesPanel.setBackground(new java.awt.Color(51, 51, 51));
        centerPropertiesPanel.setToolTipText("");
        centerPropertiesPanel.setPreferredSize(new java.awt.Dimension(235, 130));
        centerPropertiesPanel.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        yField.setText("0");
        centerPropertiesPanel.add(yField, new org.netbeans.lib.awtextra.AbsoluteConstraints(30, 200, 150, -1));

        xField.setText("0");
        centerPropertiesPanel.add(xField, new org.netbeans.lib.awtextra.AbsoluteConstraints(30, 160, 150, -1));
```

```
    mapCenterMinecraft.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
    mapCenterMinecraft.setForeground(new java.awt.Color(230, 230, 230));
    mapCenterMinecraft.setText("Точка центру у Minecraft");
    centerPropertiesPanel.add(mapCenterMinecraft,                                    new
org.netbeans.lib.awtextra.AbsoluteConstraints(10, 130, -1, -1));

    xLabel.setForeground(new java.awt.Color(230, 230, 230));
    xLabel.setText("X");
    centerPropertiesPanel.add(xLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 170, -1, -1));

    yLabel.setForeground(new java.awt.Color(230, 230, 230));
    yLabel.setText("Y");
    centerPropertiesPanel.add(yLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 200, -1, -1));

    mapCenterLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
    mapCenterLabel.setForeground(new java.awt.Color(230, 230, 230));
    mapCenterLabel.setText("Точка центру на мапі");
    centerPropertiesPanel.add(mapCenterLabel,  new  org.netbeans.lib.awtextra.AbsoluteConstraints(210,
130, -1, 20));
    centerPropertiesPanel.add(latField, new org.netbeans.lib.awtextra.AbsoluteConstraints(270, 160, 169, -
1));
    centerPropertiesPanel.add(lonField, new org.netbeans.lib.awtextra.AbsoluteConstraints(270, 200, 168,
-1));

    setWorldPathField.setToolTipText("тм");
    centerPropertiesPanel.add(setWorldPathField,  new  org.netbeans.lib.awtextra.AbsoluteConstraints(10,
80, 310, -1));

    setPathForSaveBtn.setText("Огляд");
    setPathForSaveBtn.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        setPathForSaveBtnActionPerformed(evt);
      }
    });
    centerPropertiesPanel.add(setPathForSaveBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(340,
80, 96, -1));

    setWorldNameLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
    setWorldNameLabel.setForeground(new java.awt.Color(230, 230, 230));
    setWorldNameLabel.setText("Назва світу:");
    centerPropertiesPanel.add(setWorldNameLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(10,
10, -1, -1));

    latLabel.setForeground(new java.awt.Color(230, 230, 230));
    latLabel.setText("Широта");
    centerPropertiesPanel.add(latLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 170, -1, -
1));

    lonLabel1.setForeground(new java.awt.Color(230, 230, 230));
    lonLabel1.setText("Довгота");
    centerPropertiesPanel.add(lonLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 200, -1,
-1));
```

```
        setWorldPathLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        setWorldPathLabel.setForeground(new java.awt.Color(230, 230, 230));
        setWorldPathLabel.setText("Шлях для збереження світу");
        centerPropertiesPanel.add(setWorldPathLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 50, -1, -1));

        setWorldNameField.setToolTipText("тм");
        centerPropertiesPanel.add(setWorldNameField, new org.netbeans.lib.awtextra.AbsoluteConstraints(110, 10, 210, -1));

        scalePropertiesPanel.setBackground(new java.awt.Color(51, 51, 51));
        scalePropertiesPanel.setToolTipText("");
        scalePropertiesPanel.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());

        sizeLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        sizeLabel.setForeground(new java.awt.Color(230, 230, 230));
        sizeLabel.setText("Розмір світу");
        scalePropertiesPanel.add(sizeLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 10, -1, -1));

        sizeComboBox.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "250 на 250", "1000 на 1000", "2500 на 2500", "5000 на 5000", "10000 на 10000" }));
        scalePropertiesPanel.add(sizeComboBox, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 122, -1, -1));

        sizeBtnGroup.add(metrsBtn);
        metrsBtn.setForeground(new java.awt.Color(230, 230, 230));
        metrsBtn.setSelected(true);
        metrsBtn.setText("Метрів");
        scalePropertiesPanel.add(metrsBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 44, -1, -1));

        sizeBtnGroup.add(blocksBtn);
        blocksBtn.setForeground(new java.awt.Color(230, 230, 230));
        blocksBtn.setText("Блоків");
        scalePropertiesPanel.add(blocksBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 83, -1, -1));

        scaleBtnGroup.add(oneMeterOneBlockBtn);
        oneMeterOneBlockBtn.setForeground(new java.awt.Color(230, 230, 230));
        oneMeterOneBlockBtn.setSelected(true);
        oneMeterOneBlockBtn.setText("1 метр - 1 блок");
        oneMeterOneBlockBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                oneMeterOneBlockBtnActionPerformed(evt);
            }
        });
        scalePropertiesPanel.add(oneMeterOneBlockBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 50, -1, -1));

        scaleBtnGroup.add(oneMeterTwoBlockBtn);
        oneMeterTwoBlockBtn.setForeground(new java.awt.Color(230, 230, 230));
        oneMeterTwoBlockBtn.setText("1 метр - 2 блоки");
        oneMeterTwoBlockBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                oneMeterTwoBlockBtnActionPerformed(evt);
            }
```

```
        });
        scalePropertiesPanel.add(oneMeterTwoBlockBtn,                                                    new
org.netbeans.lib.awtextra.AbsoluteConstraints(280, 50, -1, -1));

        scaleBtnGroup.add(otherValueBtn);
        otherValueBtn.setForeground(new java.awt.Color(230, 230, 230));
        otherValueBtn.setText("Інше значення (блоків на метр)");
        otherValueBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                otherValueBtnActionPerformed(evt);
            }
        });
        scalePropertiesPanel.add(otherValueBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 80, -
1, -1));

        scaleLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        scaleLabel.setForeground(new java.awt.Color(230, 230, 230));
        scaleLabel.setText("Масштаб мапи");
        scalePropertiesPanel.add(scaleLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 10, -1, -
1));
        scalePropertiesPanel.add(levelsScaleField, new org.netbeans.lib.awtextra.AbsoluteConstraints(390, 120,
60, -1));

        betweenPaneel.setBackground(new java.awt.Color(153, 153, 153));

        javax.swing.GroupLayout betweenPaneelLayout = new javax.swing.GroupLayout(betweenPaneel);
        betweenPaneel.setLayout(betweenPaneelLayout);
        betweenPaneelLayout.setHorizontalGroup(
            betweenPaneelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 20, Short.MAX_VALUE)
        );
        betweenPaneelLayout.setVerticalGroup(
            betweenPaneelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 160, Short.MAX_VALUE)
        );

        scalePropertiesPanel.add(betweenPaneel, new org.netbeans.lib.awtextra.AbsoluteConstraints(130, 0,
20, 160));
        scalePropertiesPanel.add(otherValueField, new org.netbeans.lib.awtextra.AbsoluteConstraints(370, 80,
80, -1));

        levelsScaleLabel.setFont(new java.awt.Font("Segoe UI", 0, 14)); // NOI18N
        levelsScaleLabel.setForeground(new java.awt.Color(230, 230, 230));
        levelsScaleLabel.setText("Місштаб поверху (метрів на поверх)");
        scalePropertiesPanel.add(levelsScaleLabel, new org.netbeans.lib.awtextra.AbsoluteConstraints(160, 120,
-1, -1));

        landscapePanel.setBackground(new java.awt.Color(51, 51, 51));
        landscapePanel.setToolTipText("");
        landscapePanel.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());
        landscapePanel.add(elevationField, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 121, 158, -1));

        landBtnGroup.add(enterElevationBtn);
        enterElevationBtn.setForeground(new java.awt.Color(230, 230, 230));
```

```
        enterElevationBtn.setSelected(true);
        enterElevationBtn.setText("Вказати висоту (плоский рельєф)");
        enterElevationBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                enterElevationBtnActionPerformed(evt);
            }
        });
        landscapePanel.add(enterElevationBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 82, -1, -
1));

        addLandscapeFilesBtn.setText("Додати файли рельєфу");
        addLandscapeFilesBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                addLandscapeFilesBtnActionPerformed(evt);
            }
        });
        landscapePanel.add(addLandscapeFilesBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 188,
-1, -1));

        addLandscapePath.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                addLandscapePathActionPerformed(evt);
            }
        });
        landscapePanel.add(addLandscapePath, new org.netbeans.lib.awtextra.AbsoluteConstraints(6, 229, 290,
-1));

        landBtnGroup.add(loadLandscapeFromFileBtn);
        loadLandscapeFromFileBtn.setForeground(new java.awt.Color(230, 230, 230));
        loadLandscapeFromFileBtn.setText("Завантажити рельєф з файлів");
        loadLandscapeFromFileBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                loadLandscapeFromFileBtnActionPerformed(evt);
            }
        });
        landscapePanel.add(loadLandscapeFromFileBtn, new org.netbeans.lib.awtextra.AbsoluteConstraints(6,
161, -1, -1));
        landscapePanel.add(landscapeScaleField, new org.netbeans.lib.awtextra.AbsoluteConstraints(110, 310,
70, -1));

        generateLandscapeBackgroundPanel.setBackground(new java.awt.Color(51, 51, 51));

        generateLandscapeBtn.setForeground(new java.awt.Color(230, 230, 230));
        generateLandscapeBtn.setText("Генерувати рельєф");
        generateLandscapeBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                generateLandscapeBtnActionPerformed(evt);
            }
        });

        generateLandscapeLabel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
        generateLandscapeLabel.setForeground(new java.awt.Color(230, 230, 230));
        generateLandscapeLabel.setText("Генерація рельєфу");
```

```
    javax.swing.GroupLayout generateLandscapeBackgroundPanelLayout = new
javax.swing.GroupLayout(generateLandscapeBackgroundPanel);
    generateLandscapeBackgroundPanel.setLayout(generateLandscapeBackgroundPanelLayout);
    generateLandscapeBackgroundPanelLayout.setHorizontalGroup(

generateLandscapeBackgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
        .addGroup(generateLandscapeBackgroundPanelLayout.createSequentialGroup()
          .addContainerGap()
          .addGroup(generateLandscapeBackgroundPanelLayout.createParallelGroup(javax.swing.GroupLayo
ut.Alignment.LEADING)
            .addComponent(generateLandscapeBtn, javax.swing.GroupLayout.PREFERRED_SIZE, 280,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(generateLandscapeLabel))
          .addContainerGap(44, Short.MAX_VALUE))
    );
    generateLandscapeBackgroundPanelLayout.setVerticalGroup(

generateLandscapeBackgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
generateLandscapeBackgroundPanelLayout.createSequentialGroup()
          .addContainerGap(8, Short.MAX_VALUE)
          .addComponent(generateLandscapeLabel)
          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
          .addComponent(generateLandscapeBtn)
          .addContainerGap())
    );

    landscapePanel.add(generateLandscapeBackgroundPanel, new
org.netbeans.lib.awtextra.AbsoluteConstraints(0, 0, 330, 60));

    landscapeScaleCheckbox.setForeground(new java.awt.Color(230, 230, 230));
    landscapeScaleCheckbox.setText("Встановити масштаб висот рельєфу");
    landscapeScaleCheckbox.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        landscapeScaleCheckboxActionPerformed(evt);
      }
    });
    landscapePanel.add(landscapeScaleCheckbox, new org.netbeans.lib.awtextra.AbsoluteConstraints(10,
270, -1, -1));

    landscapeScaleLabel1.setForeground(new java.awt.Color(230, 230, 230));
    landscapeScaleLabel1.setText("Блоків на метр");
    landscapePanel.add(landscapeScaleLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 310,
-1, -1));

    btnGeneratePanel.setBackground(new java.awt.Color(51, 51, 51));

    generateWorldBtn.setFont(new java.awt.Font("Segoe UI", 0, 18)); // NOI18N
    generateWorldBtn.setText("Згенерувати світ");
    generateWorldBtn.setEnabled(false);
    generateWorldBtn.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
            generateWorldBtnActionPerformed(evt);
        }
    });

    javax.swing.GroupLayout btnGeneratePanelLayout = new javax.swing.GroupLayout(btnGeneratePanel);
    btnGeneratePanel.setLayout(btnGeneratePanelLayout);
    btnGeneratePanelLayout.setHorizontalGroup(
        btnGeneratePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
btnGeneratePanelLayout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(generateWorldBtn)
            .addContainerGap())
    );
    btnGeneratePanelLayout.setVerticalGroup(
        btnGeneratePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
btnGeneratePanelLayout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(generateWorldBtn)
            .addContainerGap())
    );

    typePanel.setBackground(new java.awt.Color(51, 51, 51));

    titleTypePanel.setFont(new java.awt.Font("Segoe UI", 1, 14)); // NOI18N
    titleTypePanel.setForeground(new java.awt.Color(230, 230, 230));
    titleTypePanel.setText("Налаштування типів будівель");

    blockLabel.setForeground(new java.awt.Color(230, 230, 230));
    blockLabel.setText("Будувати з блоку");

    radiusLabel.setForeground(new java.awt.Color(230, 230, 230));
    radiusLabel.setText("Радіус (у метрах)");

    isNeeded.setForeground(new java.awt.Color(230, 230, 230));

    javax.swing.GroupLayout typePanelLayout = new javax.swing.GroupLayout(typePanel);
    typePanel.setLayout(typePanelLayout);
    typePanelLayout.setHorizontalGroup(
        typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(typePanelLayout.createSequentialGroup()
            .addGap(16, 16, 16)
            .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(typePanelLayout.createSequentialGroup()
                    .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G)
                        .addComponent(blockLabel)
                        .addComponent(radiusLabel))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                    .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                        .addComponent(blockSelection,  0,  javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
```

```
                .addGroup(typePanelLayout.createSequentialGroup()
                  .addComponent(roadRadius, javax.swing.GroupLayout.PREFERRED_SIZE, 89,
javax.swing.GroupLayout.PREFERRED_SIZE)
                  .addGap(0, 0, Short.MAX_VALUE))))
            .addGroup(typePanelLayout.createSequentialGroup()
              .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                .addComponent(isNeeded)
                .addComponent(titleTypePanel))
              .addGap(0, 0, Short.MAX_VALUE)))
          .addContainerGap())
    );
    typePanelLayout.setVerticalGroup(
      typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(typePanelLayout.createSequentialGroup()
        .addContainerGap()
        .addComponent(titleTypePanel)
        .addGap(5, 5, 5)
        .addComponent(isNeeded)
        .addGap(18, 18, 18)
        .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(blockLabel)
          .addComponent(blockSelection, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addGroup(typePanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
          .addComponent(roadRadius, javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
          .addComponent(radiusLabel))
        .addContainerGap(15, Short.MAX_VALUE))
    );

    javax.swing.GroupLayout backgroundPanelLayout = new javax.swing.GroupLayout(backgroundPanel);
    backgroundPanel.setLayout(backgroundPanelLayout);
    backgroundPanelLayout.setHorizontalGroup(
      backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(backgroundPanelLayout.createSequentialGroup()
        .addContainerGap()
        .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
          .addComponent(listsTabs)
          .addComponent(typePanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 21, Short.MAX_VALUE)
        .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
          .addGroup(backgroundPanelLayout.createSequentialGroup()
            .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING, false)
              .addComponent(scalePropertiesPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
              .addComponent(centerPropertiesPanel, javax.swing.GroupLayout.PREFERRED_SIZE, 461,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
```

```
                .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING)
                    .addComponent(landscapePanel,                javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addGroup(backgroundPanelLayout.createSequentialGroup()
                        .addGap(0, 0, Short.MAX_VALUE)
                        .addComponent(btnGeneratePanel,          javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))))
                .addComponent(jPanel1,                       javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(16, 16, 16))
        );
        backgroundPanelLayout.setVerticalGroup(
            backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(backgroundPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADI
NG, false)
                    .addGroup(backgroundPanelLayout.createSequentialGroup()
                        .addGroup(backgroundPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.L
EADING, false)
                            .addGroup(backgroundPanelLayout.createSequentialGroup()
                                .addComponent(centerPropertiesPanel,     javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                                .addGap(18, 18, 18)
                                .addComponent(scalePropertiesPanel,  javax.swing.GroupLayout.PREFERRED_SIZE,  154,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(backgroundPanelLayout.createSequentialGroup()
                                .addComponent(landscapePanel,    javax.swing.GroupLayout.PREFERRED_SIZE,    348,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                                .addComponent(btnGeneratePanel,          javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                        .addGap(18, 18, 18)
                        .addComponent(jPanel1,                   javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGroup(backgroundPanelLayout.createSequentialGroup()
                        .addComponent(typePanel,                 javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(18, 18, 18)
                        .addComponent(listsTabs)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addComponent(backgroundPanel,               javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
```

```
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(backgroundPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void openOsmFileBtnActionPerformed(java.awt.event.ActionEvent evt) {

            JFileChooser chooser = new JFileChooser();
            chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
            chooser.setMultiSelectionEnabled(false);
            int resp = chooser.showOpenDialog(this);

            String path = "";

            if(resp == JFileChooser.APPROVE_OPTION){
                path = chooser.getSelectedFile().getAbsolutePath();
            }else{
                return;
            }

            err = false;
            openOsmFileBtn.setEnabled(false);
            generateWorldBtn.setEnabled(false);
            config = new Config();
            config.setOsmPath(path);
            resetAll();
            disableAll();
            setAllObjects();

            ArrayList<String> errors = new ArrayList<>();

                SwingWorker worker = new SwingWorker<Void, Integer>() {
                        @Override
                        protected Void doInBackground() {
                            try {
                                osmData = new OsmData(config.getOsmPath(), (progress, comment) -> {
                                    setProgress(progress);
                                    statusLabel.setText(comment);
                                });
                            } catch (OsmDataException | XmlParseException ex) {
                                errors.add("Помилка аналізу або читання файлу карти osm");
                                showError(errors);
                                openOsmFileBtn.setEnabled(true);
                                generateWorldBtn.setEnabled(false);
                                err = true;
                                return null;
                            }
```

```
    return null;
}

@Override
protected void done(){

    setProgress(0);
    statusLabel.setText("");

    if(err){
        return;
    }

    enableAll();
    setAllObjects();

    if(!errors.isEmpty())
        return;

    Vector2D min = osmData.getMinPoint();
    Vector2D max = osmData.getMaxPoint();

    latField.setText(String.valueOf((min.getY() + max.getY()) / 2.0));
    lonField.setText(String.valueOf((min.getX() + max.getX()) / 2.0));

    try{
        bTypesArr = osmData.getBuildingTypes();
        hTypesArr = osmData.getHighwayTypes();
        lTypesArr = osmData.getLanduseTypes();
        rTypesArr = osmData.getReservoirTypes();
    }catch(OsmDataException e){
        if(!errors.isEmpty()){
            showError(errors);
            enableAll();
            openOsmFileBtn.setEnabled(true);
            generateWorldBtn.setEnabled(true);
        }
    }

    bTypes = new ArrayList<>();
    hTypes = new ArrayList<>();
    lTypes = new ArrayList<>();
    rTypes = new ArrayList<>();

    bParams = new HashMap(bTypes.size());
    hParams = new HashMap(bTypes.size());
    lParams = new HashMap(bTypes.size());
    rParams = new HashMap(bTypes.size());

    for(String str : bTypesArr){
        bTypes.add(Translator.getBuildingTypeTranslate(str));

        BuildingConfig buildingConfig = new BuildingConfig(
```

```
            config.getDefaultBuildingConfig().getBlockName(),
            config.getDefaultBuildingConfig().isIsNeeded());
        bParams.put(str, buildingConfig);
}

for(String str : hTypesArr){
        hTypes.add(Translator.getRoadTypeTranslate(str));

        HighwayConfig highwayConfig = new HighwayConfig(
            config.getDefaultRoadConfig().getBlockName(),
            config.getDefaultRoadConfig().getRadius(),
            config.getDefaultRoadConfig().isIsNeeded());
        hParams.put(str, highwayConfig);
}

for(String str : lTypesArr){
        lTypes.add(Translator.getLanduseTypeTranslate(str));

        LanduseConfig landuseConfig = new LanduseConfig(
            config.getDefaultLanduseConfig().getBlockName(),
            config.getDefaultLanduseConfig().isIsNeeded());
        lParams.put(str, landuseConfig);
}

for(String str : rTypesArr){
        rTypes.add(Translator.getWaterTypeTranslate(str));

        WaterConfig waterConfig = new WaterConfig(
            config.getDefaultWaterConfig().isIsNeeded());
        rParams.put(str, waterConfig);
}

DefaultListModel<String> bModel = (DefaultListModel<String>) buildingsList.getModel();
bModel.clear();
bModel.addAll(bTypes);

DefaultListModel<String> hModel = (DefaultListModel<String>) roadsList.getModel();
hModel.clear();
hModel.addAll(hTypes);

DefaultListModel<String> lModel = (DefaultListModel<String>) landusesList.getModel();
lModel.clear();
lModel.addAll(lTypes);

DefaultListModel<String> rModel = (DefaultListModel<String>) watersList.getModel();
rModel.clear();
rModel.addAll(rTypes);

listsTabs.setSelectedIndex(0);
list = Lists.BUILDINGS;
setListTabs();
generateWorldBtn.setEnabled(true);
openOsmFileBtn.setEnabled(true);
openOsmFileBtn.setText("Завантажити інший файл");
```

```
                }

            };

            worker.addPropertyChangeListener(listener -> {
                if(listener.getPropertyName().equals("progress")){
                   progress.setValue((int) worker.getProgress());
                }
            });

            worker.execute();

    }

    private void oneMeterOneBlockBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setOtherValueField();
    }

    private void oneMeterTwoBlockBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setOtherValueField();
    }

    private void otherValueBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setOtherValueField();
    }

    private void enterElevationBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setLandscapeElevationField();
       setLandscapeInputFromFile();
       setLandscapeScale();
    }

    private void addLandscapePathActionPerformed(java.awt.event.ActionEvent evt) {
       // TODO add your handling code here:
    }

    private void loadLandscapeFromFileBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setLandscapeElevationField();
       setLandscapeInputFromFile();
       setLandscapeScale();
    }

    private void generateLandscapeBtnActionPerformed(java.awt.event.ActionEvent evt) {
       setLandObjects();
    }

    private void landscapeScaleCheckboxActionPerformed(java.awt.event.ActionEvent evt) {
       setLandscapeScale();
    }

    private void generateWorldBtnActionPerformed(java.awt.event.ActionEvent evt) {

       openOsmFileBtn.setEnabled(false);
       generateWorldBtn.setEnabled(false);
```

```
disableAll();

setWorldNameField.setBackground(Color.WHITE);
setWorldPathField.setBackground(Color.WHITE);
levelsScaleField.setBackground(Color.WHITE);
xField.setBackground(Color.WHITE);
yField.setBackground(Color.WHITE);
lonField.setBackground(Color.WHITE);
latField.setBackground(Color.WHITE);
otherValueField.setBackground(Color.WHITE);
elevationField.setBackground(Color.WHITE);
addLandscapePath.setBackground(Color.WHITE);
landscapeScaleField.setBackground(Color.WHITE);

String worldName = setWorldNameField.getText();
String worldPath = setWorldPathField.getText();
String levelsScale = levelsScaleField.getText();
String xStr = xField.getText();
String yStr = yField.getText();
String lonStr = lonField.getText();
String latStr = latField.getText();

ArrayList<String> errors = new ArrayList();

if(worldName.isBlank()){
   errors.add("Введіть назву світу Minecraft");
   setWorldNameField.setBackground(Color.RED);
}else{
   config.setWorldName(worldName);
}

if(worldPath.isBlank()){
   errors.add("Вкажіть шлях для збереження світу Minecraft");
   setWorldPathField.setBackground(Color.RED);
}else{
   config.setSavePath(worldPath);
}

if(levelsScale.isBlank()){
   errors.add("Вкажіть кількість метрів для поверху будівлі");
   levelsScaleField.setBackground(Color.RED);
}else{
   try{
      double value = Double.valueOf(levelsScale);

      if(value <= 0){
         errors.add("Введіть значення більше нуля у кількості метрів для поверху будівлі");
         levelsScaleField.setBackground(Color.RED);
      }else{
         config.setLevelsScale(value);
      }
   }catch(NumberFormatException e){
      errors.add("Помилка розбору числового значення у кількості метрів для поверху будівлі");
```

```
            levelsScaleField.setBackground(Color.RED);
        }
    }

    if(xStr.isBlank()){
        errors.add("Введіть дані координати X для точки центру у Minecraft");
        xField.setBackground(Color.RED);
    }else{
        try{
            int x = Integer.valueOf(xStr);
            config.setCentreX(x);
        }catch(NumberFormatException e){
            errors.add("Помилка розбору числового значення у координаті X для точки центру у Minecraft");
            xField.setBackground(Color.RED);
        }
    }

    if(yStr.isBlank()){
        errors.add("Введіть дані координати Y для точки центру у Minecraft");
        yField.setBackground(Color.RED);
    }else{
        try{
            int y = Integer.valueOf(yStr);
            config.setCentreY(y);
        }catch(NumberFormatException e){
            errors.add("Помилка розбору числового значення у координаті Y для точки центру у Minecraft");
            yField.setBackground(Color.RED);
        }
    }

    if(lonStr.isBlank()){
        errors.add("Введіть довготу для точки центру на мапі");
        lonField.setBackground(Color.RED);
    }else{
        try{
            double lonCentre = Double.valueOf(lonStr);

            if(lonCentre < -180 || lonCentre > 180){
                errors.add("Введіть значення у довготі для точки центру на мапі в діапазоні [-180; 180]");
                lonField.setBackground(Color.RED);
            }else{
                config.setCentreLon(lonCentre);
            }
        }catch(NumberFormatException e){
            errors.add("Помилка розбору числового значення у довготі для точки центру на мапі");
            lonField.setBackground(Color.RED);
        }
    }

    if(latStr.isBlank()){
        errors.add("Введіть широту для точки центру на мапі");
        latField.setBackground(Color.RED);
    }else{
        try{
```

```
            double latCentre = Double.valueOf(latStr);

            if(latCentre < -180 || latCentre > 180){
               errors.add("Введіть значення у широті для точки центру на мапі в діапазоні (-90; 90)");
               latField.setBackground(Color.RED);
            }else{
               config.setCentreLat(latCentre);
            }
         }catch(NumberFormatException e){
            errors.add("Помилка розбору числового значення у широті для точки центру на мапі");
            latField.setBackground(Color.RED);
         }
      }

      if(oneMeterOneBlockBtn.isSelected()){
         config.setScale(1.0);
      }else{
         if(oneMeterTwoBlockBtn.isSelected()){
            config.setScale(2.0);
         }else{
            if(otherValueBtn.isSelected()){
               String worldScale = otherValueField.getText();

               if(worldScale.isBlank()){
                  errors.add("Введіть масштаб для генерування світу (блоків на метр)");
                  otherValueField.setBackground(Color.RED);
               }else{
                  try{
                     double scale = Double.valueOf(worldScale);

                     if(scale <= 0){
                        errors.add("Введіть значення більше нуля для генерування світу (блоків на метр)");
                        otherValueField.setBackground(Color.RED);
                     }else{
                        config.setScale(scale);
                     }
                  }catch(NumberFormatException e){
                     errors.add("Помилка розбору числового значення для генерування світу (блоків на
метр)");
                     otherValueField.setBackground(Color.RED);
                  }
               }
            }
         }
      }

      if(generateLandscapeBtn.isSelected()){
         if(enterElevationBtn.isSelected()){
            String elevationStr = elevationField.getText();

            if(elevationStr.isBlank()){
               errors.add("Введіть висоту для генерування рельєфу");
               elevationField.setBackground(Color.RED);
            }else{
```

```
        try{
            double elevation = Double.valueOf(elevationStr);
            landData = new FlatLandscapeData(elevation);
        }catch(NumberFormatException e){
            errors.add("Помилка розбору числового значення для висоти генерування рельєфу");
            elevationField.setBackground(Color.RED);
        }
    }

}else{
    if(loadLandscapeFromFileBtn.isSelected()){
        if(landscapeScaleCheckbox.isSelected()){
            String landscapePath = addLandscapePath.getText();
            String landscapeScale = landscapeScaleField.getText();
            double landscapeScaleValue = 1;
            int localErrorsCount = 0;

            if(landscapePath.isBlank()){
                localErrorsCount++;
                errors.add("Введіть шлях для завантаження файлів рельєфу");
                addLandscapePath.setBackground(Color.RED);
            }

            if(landscapeScale.isBlank()){
                localErrorsCount++;
                errors.add("Введіть масштаб висот рельєфу");
                landscapeScaleField.setBackground(Color.RED);
            }else{
                try{
                    landscapeScaleValue = Double.valueOf(landscapeScaleField.getText());
                }catch(NumberFormatException e){
                    localErrorsCount++;
                    errors.add("Помилка розбору числового значення для масштабу висот рельєфу");
                    landscapeScaleField.setBackground(Color.RED);
                }
            }

            if(localErrorsCount == 0){
                try {
                    landData = new InterpolatedLandscapeData(landscapePath, landscapeScaleValue);
                }catch (DataSourceException ex) {
                    errors.add("Помилка аналізу або читання файлів GeoTIFF");
                                addLandscapePath.setBackground(Color.RED);
                }
                            catch (IOException ex) {
                    errors.add("Помилка аналізу або читання файлів GeoTIFF");
                                addLandscapePath.setBackground(Color.RED);
                }
            }

        }else{
            String landscapePath = addLandscapePath.getText();
            if(landscapePath.isBlank()){
                errors.add("Введіть шлях для завантаження файлів рельєфу");
```

```
                addLandscapePath.setBackground(Color.RED);
            }else{
                try {
                    landData = new InterpolatedLandscapeData(landscapePath);
                }catch (DataSourceException ex) {
                    errors.add("Помилка аналізу або читання файлів GeoTIFF");
                                addLandscapePath.setBackground(Color.RED);
                }
                            catch (IOException ex) {
                    errors.add("Помилка аналізу або читання файлів рельєфу");
                                addLandscapePath.setBackground(Color.RED);
                }
            }
        }
    }
  }
}else{
  landData = new FlatLandscapeData(2);
}

if(sizeComboBox.getSelectedIndex() == 0){
  config.setWorldSize(250);
}
if(sizeComboBox.getSelectedIndex() == 1){
  config.setWorldSize(1000);
}
if(sizeComboBox.getSelectedIndex() == 2){
  config.setWorldSize(2500);
}
if(sizeComboBox.getSelectedIndex() == 3){
  config.setWorldSize(5000);
}
if(sizeComboBox.getSelectedIndex() == 4){
  config.setWorldSize(10000);
}

config.setbParams(bParams);
config.setlParams(lParams);
config.sethParams(hParams);
config.setwParams(rParams);

if(!errors.isEmpty()){
  showError(errors);
  enableAll();
  openOsmFileBtn.setEnabled(true);
  generateWorldBtn.setEnabled(false);
  return;
}

try{
  Map map = new Map(config, osmData, landData);
  progress.setValue(0);

  SwingWorker worker = new SwingWorker<Void, Integer>() {
```

```
        @Override
        protected Void doInBackground() {
            map.setBuildProcess((currentProgress, comment) -> {
                setProgress(currentProgress);
                statusLabel.setText(comment);
            });

            map.setSaveProcess((currentProgress, comment) -> {
                setProgress(currentProgress);
                statusLabel.setText(comment);
            });

            try {
                map.build();
            } catch (SaveWorldException ex) {
                        errors.add("Виникла невідома помилка при збереженні світу");
                        if(!errors.isEmpty()){
                            showError(errors);
                            enableAll();
                            openOsmFileBtn.setEnabled(true);
                            generateWorldBtn.setEnabled(true);
                        }
                return null;
            }

            return null;
        }

        @Override
        protected void done(){
            enableAll();
            openOsmFileBtn.setEnabled(true);
            generateWorldBtn.setEnabled(true);
        }
    };

    worker.addPropertyChangeListener(listener -> {
        if(listener.getPropertyName().equals("progress")){
            progress.setValue((int) worker.getProgress());
        }
    });

    worker.execute();

  }catch(OsmDataException e){
        errors.add("Помилка аналізу файлу мапи .osm Можливо файл було пошкоджено");
        if(!errors.isEmpty()){
                showError(errors);
                openOsmFileBtn.setEnabled(true);
                generateWorldBtn.setEnabled(true);
        }
  }

}
```

```
private void formWindowOpened(java.awt.event.WindowEvent evt) {
   resetAll();
        disableAll();

        backgroundDisabledColor = new Color(153, 153, 153);
        backgroundEnabledColor = new Color(51, 51, 51);

        buildingsList.setModel(new DefaultListModel<String>());
        roadsList.setModel(new DefaultListModel<String>());
        landusesList.setModel(new DefaultListModel<String>());
        watersList.setModel(new DefaultListModel<String>());

        Set<String> blockNames = Blocks.getBlockNames();
        for(String str : blockNames){
           blockSelection.addItem(str);
        }

        buildingsList.addListSelectionListener(e -> {
           if(!e.getValueIsAdjusting()){
                if(buildingsList.getSelectedIndex() == -1) return;

                currentBuildings = bParams.get(bTypesArr.get(buildingsList.getSelectedIndex()));

                isNeeded.setText("Генерувати " + buildingsList.getSelectedValue());
                isNeeded.setSelected(currentBuildings.isIsNeeded());

                blockSelection.setSelectedItem(currentBuildings.getBlockName());
                roadRadius.setText("");
           }
        });


        roadsList.addListSelectionListener(e -> {
           if(!e.getValueIsAdjusting()){
                if(roadsList.getSelectedIndex() == -1) return;

                currentHighways = hParams.get(hTypesArr.get(roadsList.getSelectedIndex()));

                isNeeded.setText("Генерувати " + roadsList.getSelectedValue());
                isNeeded.setSelected(currentHighways.isIsNeeded());

                blockSelection.setSelectedItem(currentHighways.getBlockName());
                roadRadius.setText(String.valueOf(currentHighways.getRadius()));
           }
        });

        landusesList.addListSelectionListener(e -> {
           if(!e.getValueIsAdjusting()){
                if(landusesList.getSelectedIndex() == -1) return;

                currentLanduses = lParams.get(lTypesArr.get(landusesList.getSelectedIndex()));

                isNeeded.setText("Генерувати " + landusesList.getSelectedValue());
```

```
                isNeeded.setSelected(currentLanduses.isIsNeeded());

                blockSelection.setSelectedItem(currentLanduses.getBlockName());
                roadRadius.setText("");
            }
        });

        watersList.addListSelectionListener(e -> {
            if(!e.getValueIsAdjusting()){
                if(watersList.getSelectedIndex() == -1) return;

                currentWaters = rParams.get(rTypesArr.get(watersList.getSelectedIndex()));

                isNeeded.setText("Генерувати " + watersList.getSelectedValue());
                isNeeded.setSelected(currentWaters.isIsNeeded());

                blockSelection.setSelectedItem(blockSelection.getItemAt(0));
                roadRadius.setText("");
            }
        });

        isNeeded.addActionListener(l -> {
            if(list == WorldGenerator.Lists.BUILDINGS){
                if(currentBuildings != null){
                    currentBuildings.setIsNeeded(isNeeded.isSelected());
                }
            }

            if(list == WorldGenerator.Lists.LANDUSES){
                if(currentLanduses != null){
                    currentLanduses.setIsNeeded(isNeeded.isSelected());
                }
            }

            if(list == WorldGenerator.Lists.ROADS){
                if(currentHighways != null){
                    currentHighways.setIsNeeded(isNeeded.isSelected());
                }
            }

            if(list == WorldGenerator.Lists.WATERS){
                if(currentBuildings != null){
                    currentBuildings.setIsNeeded(isNeeded.isSelected());
                }
            }
        });

        blockSelection.addActionListener(l -> {
            if(list == WorldGenerator.Lists.BUILDINGS){
                if(currentBuildings != null){

currentBuildings.setBlockName(blockSelection.getItemAt(blockSelection.getSelectedIndex()));
                }
            }
```

```
            if(list == WorldGenerator.Lists.LANDUSES){
                if(currentLanduses != null){

currentLanduses.setBlockName(blockSelection.getItemAt(blockSelection.getSelectedIndex()));
                }
            }

            if(list == WorldGenerator.Lists.ROADS){
                if(currentHighways != null){

currentHighways.setBlockName(blockSelection.getItemAt(blockSelection.getSelectedIndex()));
                }
            }

        });

        roadRadius.getDocument().addDocumentListener(new DocumentListener(){

            @Override
            public void insertUpdate(DocumentEvent e) {
                if(list == WorldGenerator.Lists.ROADS){
                  if(currentHighways != null){
                        try{
                          float radius = Float.valueOf(roadRadius.getText());
                          currentHighways.setRadius(radius);
                          roadRadius.setBackground(Color.WHITE);
                        }catch(NumberFormatException exception){
                          currentHighways.setRadius(config.getDefaultRoadConfig().getRadius());
                          roadRadius.setBackground(Color.RED);
                        }
                  }
                }
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                if(list == WorldGenerator.Lists.ROADS){
                  if(currentHighways != null){
                        try{
                          float radius = Float.valueOf(roadRadius.getText());
                          currentHighways.setRadius(radius);
                          roadRadius.setBackground(Color.WHITE);
                        }catch(NumberFormatException exception){
                          currentHighways.setRadius(config.getDefaultRoadConfig().getRadius());
                          roadRadius.setBackground(Color.RED);
                        }
                  }
                }
            }

            @Override
            public void changedUpdate(DocumentEvent e) {
```

```
            }

        });

        listsTabs.addChangeListener(e -> {
            setListTabs();
        });
    }

    private void setPathForSaveBtnActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        chooser.setMultiSelectionEnabled(false);
        int resp = chooser.showOpenDialog(this);
        String path = "";

        if(resp == JFileChooser.APPROVE_OPTION){
            path = chooser.getSelectedFile().getAbsolutePath();
            setWorldPathField.setText(path);
        }else{
            return;
        }
    }

    private void addLandscapeFilesBtnActionPerformed(java.awt.event.ActionEvent evt) {
        JFileChooser chooser = new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        chooser.setMultiSelectionEnabled(false);
        int resp = chooser.showOpenDialog(this);
        String path = "";

        if(resp == JFileChooser.APPROVE_OPTION){
            path = chooser.getSelectedFile().getAbsolutePath();
            addLandscapePath.setText(path);
        }else{
            return;
        }
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(WorldGenerator.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
        } catch (InstantiationException ex) {
```

```
java.util.logging.Logger.getLogger(WorldGenerator.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(WorldGenerator.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(WorldGenerator.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }

        java.awt.EventQueue.invokeLater(new Runnable() {
           public void run() {
                new WorldGenerator().setVisible(true);
           }
        });

    }

    private void showError(ArrayList<String> errors){
        String error = "";

        for(String e : errors){
           error += (e + "\n");
        }

        JTextArea area = new JTextArea(error);
        JScrollPane scroll = new JScrollPane(area);
        JOptionPane.showMessageDialog(this, scroll, "Помилка", JOptionPane.ERROR_MESSAGE);
    }

    private void setListTabs(){
        switch(listsTabs.getSelectedIndex()){
                case 0:
                   list = WorldGenerator.Lists.BUILDINGS;
                   titleTypePanel.setText("Налаштування типів будівель");
                   isNeededEnabled = true;
                   blockSelectionEnabled = true;
                   roadRadiusEnabled = false;

                   setIsNeeded();
                   setBlockSelection();
                   setRoadRadius();
                   break;

                case 1:
                   list = WorldGenerator.Lists.ROADS;
                   titleTypePanel.setText("Налаштування типів доріг");
                   isNeededEnabled = true;
                   blockSelectionEnabled = true;
                   roadRadiusEnabled = true;

                   setIsNeeded();
```

```
                setBlockSelection();
                setRoadRadius();
                break;

            case 2:
                list = WorldGenerator.Lists.LANDUSES;
                titleTypePanel.setText("Налаштування типів місцевостей");
                isNeededEnabled = true;
                blockSelectionEnabled = true;
                roadRadiusEnabled = false;

                setIsNeeded();
                setBlockSelection();
                setRoadRadius();
                break;

            case 3:
                list = WorldGenerator.Lists.WATERS;
                titleTypePanel.setText("Налаштування типів водойм");
                isNeededEnabled = true;
                blockSelectionEnabled = false;
                roadRadiusEnabled = false;

                setIsNeeded();
                setBlockSelection();
                setRoadRadius();
                break;
        }
    }

    private void disableAll(){
        enabledSettings = false;
        setAllObjects();
    }

    private void enableAll(){
        enabledSettings = true;
        setAllObjects();
    }

    private void setAllObjects(){
        setListObjects();
        setLandObjects();
        setMainSettingsObjects();
    }

    private void setListObjects(){
        if(enabledSettings){
            typePanel.setBackground(backgroundEnabledColor);
        }else{
            typePanel.setBackground(backgroundDisabledColor);
        }

        listsTabs.setEnabled(enabledSettings);
```

```
        buildingsList.setEnabled(enabledSettings);
        landusesList.setEnabled(enabledSettings);
        roadsList.setEnabled(enabledSettings);
        watersList.setEnabled(enabledSettings);

        isNeeded.setVisible(enabledSettings);
        isNeeded.setEnabled(enabledSettings);

        setBlockSelection();
        setRoadRadius();
}

private void setMainSettingsObjects(){
        if(enabledSettings){
           centerPropertiesPanel.setBackground(backgroundEnabledColor);
           scalePropertiesPanel.setBackground(backgroundEnabledColor);
           generateLandscapeBackgroundPanel.setBackground(backgroundEnabledColor);
        }else{
           centerPropertiesPanel.setBackground(backgroundDisabledColor);
           scalePropertiesPanel.setBackground(backgroundDisabledColor);
           generateLandscapeBackgroundPanel.setBackground(backgroundDisabledColor);
        }

        setWorldNameField.setEnabled(enabledSettings);
        xField.setEnabled(enabledSettings);
        yField.setEnabled(enabledSettings);
        latField.setEnabled(enabledSettings);
        lonField.setEnabled(enabledSettings);
        setWorldPathField.setEnabled(enabledSettings);
        setPathForSaveBtn.setEnabled(enabledSettings);

        metrsBtn.setEnabled(enabledSettings);
        blocksBtn.setEnabled(enabledSettings);
        oneMeterOneBlockBtn.setEnabled(enabledSettings);
        oneMeterTwoBlockBtn.setEnabled(enabledSettings);
        otherValueBtn.setEnabled(enabledSettings);
        sizeComboBox.setEnabled(enabledSettings);
        levelsScaleField.setEnabled(enabledSettings);
        setOtherValueField();
}

private void setOtherValueField(){
        otherValueField.setEnabled(otherValueBtn.isSelected() && enabledSettings);
}

private void setLandObjects(){
        generateLandscapeBtn.setEnabled(enabledSettings);

        if(enabledSettings && generateLandscapeBtn.isSelected()){
           landscapePanel.setBackground(backgroundEnabledColor);
        }else{
           landscapePanel.setBackground(backgroundDisabledColor);
        }
```

```
        enterElevationBtn.setEnabled(generateLandscapeBtn.isSelected() && enabledSettings);
        loadLandscapeFromFileBtn.setEnabled(generateLandscapeBtn.isSelected() && enabledSettings);
        setLandscapeElevationField();
        setLandscapeInputFromFile();
        setLandscapeScale();
    }

    private void setLandscapeElevationField(){
        elevationField.setEnabled(
                enterElevationBtn.isSelected() && generateLandscapeBtn.isSelected() && enabledSettings);
    }

    private void setLandscapeInputFromFile(){
        addLandscapeFilesBtn.setEnabled(
                loadLandscapeFromFileBtn.isSelected()  &&  generateLandscapeBtn.isSelected()  &&
enabledSettings);
        addLandscapePath.setEnabled(
                loadLandscapeFromFileBtn.isSelected()  &&  generateLandscapeBtn.isSelected()  &&
enabledSettings);
    }

    private void setLandscapeScale(){
        landscapeScaleCheckbox.setEnabled(
                loadLandscapeFromFileBtn.isSelected()  &&  generateLandscapeBtn.isSelected()  &&
enabledSettings);
        landscapeScaleField.setEnabled(
                landscapeScaleCheckbox.isSelected() && loadLandscapeFromFileBtn.isSelected()
                        && generateLandscapeBtn.isSelected() && enabledSettings);
    }

    private void setIsNeeded(){
        isNeeded.setEnabled(isNeededEnabled && enabledSettings);
    }

    private void setBlockSelection(){
        blockSelection.setEnabled(blockSelectionEnabled && enabledSettings);
    }

    private void setRoadRadius(){
        roadRadius.setEnabled(roadRadiusEnabled && enabledSettings);
    }

    private void resetAll(){
        list = WorldGenerator.Lists.NONE;
        setWorldNameField.setText("NewWorld1");
        xField.setText("0");
        yField.setText("0");
        latField.setText("");
        lonField.setText("");
        setWorldPathField.setText("");
        metrsBtn.setSelected(true);
        oneMeterOneBlockBtn.setSelected(true);
        sizeComboBox.setSelectedIndex(0);
        levelsScaleField.setText("");
```

```
        generateLandscapeBtn.setSelected(false);
        enterElevationBtn.setSelected(true);
        elevationField.setText("");
        addLandscapePath.setText("");
        landscapeScaleCheckbox.setSelected(false);
        landscapeScaleField.setText("");
        levelsScaleField.setText("4");

    }

    private OsmData osmData;
    private ILandscapeData landData;
    private Config config;

    private Color backgroundDisabledColor;
    private Color backgroundEnabledColor;

    private String osmFilePath = null;

    private ArrayList<String> bTypesArr;
    private ArrayList<String> hTypesArr;
    private ArrayList<String> lTypesArr;
    private ArrayList<String> rTypesArr;

    private ArrayList<String> bTypes;
    private ArrayList<String> hTypes;
    private ArrayList<String> lTypes;
    private ArrayList<String> rTypes;

    private HashMap<String, BuildingConfig> bParams;
    private HashMap<String, HighwayConfig> hParams;
    private HashMap<String, LanduseConfig> lParams;
    private HashMap<String, WaterConfig> rParams;

    private BuildingConfig currentBuildings = null;
    private HighwayConfig currentHighways = null;
    private LanduseConfig currentLanduses = null;
    private WaterConfig currentWaters = null;

    private boolean enabledSettings = false;
    private boolean err = false;

    private boolean isNeededEnabled = false;
    private boolean blockSelectionEnabled = false;
    private boolean roadRadiusEnabled = false;

    private enum Lists {BUILDINGS, ROADS, LANDUSES, WATERS, NONE};

    private Lists list = Lists.NONE;

    // Variables declaration - do not modify
    private javax.swing.JButton addLandscapeFilesBtn;
    private javax.swing.JTextField addLandscapePath;
    private javax.swing.JPanel backgroundPanel;
```

```
private javax.swing.JPanel betweenPaneel;
private javax.swing.JLabel blockLabel;
private javax.swing.JComboBox<String> blockSelection;
private javax.swing.JRadioButton blocksBtn;
private javax.swing.JPanel btnGeneratePanel;
private javax.swing.JList<String> buildingsList;
private javax.swing.JScrollPane buildingsListScrollPane;
private javax.swing.JPanel buildingsTab;
private javax.swing.JPanel centerPropertiesPanel;
private javax.swing.JTextField elevationField;
private javax.swing.JRadioButton enterElevationBtn;
private javax.swing.JPanel generateLandscapeBackgroundPanel;
private javax.swing.JCheckBox generateLandscapeBtn;
private javax.swing.JLabel generateLandscapeLabel;
private javax.swing.JButton generateWorldBtn;
private javax.swing.JCheckBox isNeeded;
private javax.swing.JPanel jPanel1;
private javax.swing.ButtonGroup landBtnGroup;
private javax.swing.JPanel landscapePanel;
private javax.swing.JCheckBox landscapeScaleCheckbox;
private javax.swing.JTextField landscapeScaleField;
private javax.swing.JLabel landscapeScaleLabel1;
private javax.swing.JList<String> landusesList;
private javax.swing.JScrollPane landusesListScrollPane;
private javax.swing.JPanel landusesTab;
private javax.swing.JTextField latField;
private javax.swing.JLabel latLabel;
private javax.swing.JTextField levelsScaleField;
private javax.swing.JLabel levelsScaleLabel;
private javax.swing.JTabbedPane listsTabs;
private javax.swing.JRadioButton loadLandscapeFromFileBtn;
private javax.swing.JTextField lonField;
private javax.swing.JLabel lonLabel1;
private javax.swing.JLabel mapCenterLabel;
private javax.swing.JLabel mapCenterMinecraft;
private javax.swing.JRadioButton metrsBtn;
private javax.swing.JRadioButton oneMeterOneBlockBtn;
private javax.swing.JRadioButton oneMeterTwoBlockBtn;
private javax.swing.JButton openOsmFileBtn;
private javax.swing.JRadioButton otherValueBtn;
private javax.swing.JTextField otherValueField;
private javax.swing.JProgressBar progress;
private javax.swing.JLabel radiusLabel;
private javax.swing.JTextField roadRadius;
private javax.swing.JList<String> roadsList;
private javax.swing.JScrollPane roadsListScrollPane;
private javax.swing.JPanel roadsTab;
private javax.swing.ButtonGroup scaleBtnGroup;
private javax.swing.JLabel scaleLabel;
private javax.swing.JPanel scalePropertiesPanel;
private javax.swing.JButton setPathForSaveBtn;
private javax.swing.JTextField setWorldNameField;
private javax.swing.JLabel setWorldNameLabel;
private javax.swing.JTextField setWorldPathField;
```

```
    private javax.swing.JLabel setWorldPathLabel;
    private javax.swing.ButtonGroup sizeBtnGroup;
    private javax.swing.JComboBox<String> sizeComboBox;
    private javax.swing.JLabel sizeLabel;
    private javax.swing.JLabel statusLabel;
    private javax.swing.JLabel titleTypePanel;
    private javax.swing.JPanel typePanel;
    private javax.swing.JList<String> watersList;
    private javax.swing.JScrollPane watersListScrollPane;
    private javax.swing.JPanel watersTab;
    private javax.swing.JTextField xField;
    private javax.swing.JLabel xLabel;
    private javax.swing.JTextField yField;
    private javax.swing.JLabel yLabel;
    // End of variables declaration
}
```

Для файлу Area.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import org.minemapcreator.minemapcreator.math.MapMath;
import org.minemapcreator.minemapcreator.math.Vector2D;
import org.minemapcreator.minemapcreator.math.Vector3D;


public class Area {

    private Vector2D pointA;
    private Vector2D pointB;
    private Vector2D pointC;
    private Vector2D pointD;

    private Vector2D pointRadiansA;
    private Vector2D pointRadiansB;
    private Vector2D pointRadiansC;
    private Vector2D pointRadiansD;

    private Vector3D pointXYZA;
    private Vector3D pointXYZB;
    private Vector3D pointXYZC;
    private Vector3D pointXYZD;

    private int width;
    private int height;

    private double widthD;
    private double heightD;

    private double scale;

    public Area(Vector2D pointA, Vector2D pointC, double scale) {
        this.pointA = pointA;
```

```
        this.pointC = pointC;
        this.scale = scale;
        setPoints();
        setWidthAndHeight();
    }



    private void setWidthAndHeight(){
        this.width = (int) Math.round(MapMath.getDistance(pointRadiansA, pointRadiansD) * scale);
        this.height = (int) Math.round(MapMath.getDistance(pointRadiansA, pointRadiansB) * scale);

        this.widthD = MapMath.getDistance(pointRadiansA, pointRadiansD) * scale;
        this.heightD = MapMath.getDistance(pointRadiansA, pointRadiansB) * scale;
    }

    private void setPoints(){
        pointRadiansA = MapMath.pointToRadians(pointA);
        pointRadiansC = MapMath.pointToRadians(pointC);

        pointXYZA = MapMath.lonRLatRToXYZ(pointRadiansA);
        pointXYZC = MapMath.lonRLatRToXYZ(pointRadiansC);

        Vector3D center = MapMath.getCenter(pointXYZA, pointXYZC);

        pointXYZD = MapMath.getAnotherPoint(center, pointXYZA, Math.cos(pointRadiansA.getY()));
        pointXYZB = MapMath.getAnotherPoint(center, pointXYZC, Math.cos(pointRadiansC.getY()));

        pointRadiansB = MapMath.XYZToLonRLatR(pointXYZB);
        pointRadiansD = MapMath.XYZToLonRLatR(pointXYZD);

        pointB = MapMath.pointToDegrees(pointRadiansB);
        pointD = MapMath.pointToDegrees(pointRadiansD);

    }

    public Vector2D getXYFromLonLatRad(Vector2D lonLatPoint){
        Vector2D point =  MapMath.getXYFromLonLatRad(lonLatPoint, pointRadiansA, pointRadiansB,
pointRadiansC, pointRadiansD);
        return new Vector2D(point.getX() * this.scale, point.getY() * this.scale);
    }

    public Vector2D getPointA() {
        return pointA;
    }

    public Vector2D getPointB() {
        return pointB;
    }

    public Vector2D getPointC() {
        return pointC;
    }
```

```
public Vector2D getPointD() {
	return pointD;
}

public Vector2D getPointRadiansA() {
	return pointRadiansA;
}

public Vector2D getPointRadiansB() {
	return pointRadiansB;
}

public Vector2D getPointRadiansC() {
	return pointRadiansC;
}

public Vector2D getPointRadiansD() {
	return pointRadiansD;
}

public Vector3D getPointXYZA() {
	return pointXYZA;
}

public Vector3D getPointXYZB() {
	return pointXYZB;
}

public Vector3D getPointXYZC() {
	return pointXYZC;
}

public Vector3D getPointXYZD() {
	return pointXYZD;
}

public int getWidth(){
	return width;
}

public int getHeight(){
	return height;
}

public double getWidthD() {
	return widthD;
}

public double getHeightD() {
	return heightD;
}

public double getScale() {
	return scale;
```

```
    }

}
```

Для файлу BlockLine.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import net.morbz.minecraft.blocks.IBlock;


public class BlockLine {

    private IBlock block;
    private int elevation;
    private int height;

    public BlockLine(IBlock block, int elevation) {
        this(block, elevation, 0);
    }

    public BlockLine(IBlock block, int elevation, int height) {
        this.block = block;
        this.elevation = elevation;
        this.height = height;
    }

    public IBlock getBlock() {
        return block;
    }

    public int getElevation() {
        return elevation;
    }

    public int getHeight() {
        return height;
    }

}
```

Для файлу Blocks.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.util.HashMap;
import java.util.Set;
import net.morbz.minecraft.blocks.CustomBlock;
import net.morbz.minecraft.blocks.IBlock;


public class Blocks {
```

```
private static HashMap<String, IBlock> blocks = null;

private static void init(){
	blocks = new HashMap();

	blocks.put("Порожнеча", new CustomBlock(0, 0, 0));
	blocks.put("Каміння", new CustomBlock(1, 0, 0));
	blocks.put("Граніт", new CustomBlock(1, 1, 0));
	blocks.put("Полірований граніт", new CustomBlock(1, 2, 0));
	blocks.put("Діорит", new CustomBlock(1, 3, 0));
	blocks.put("Полірований діорит", new CustomBlock(1, 4, 0));
	blocks.put("Андезіт", new CustomBlock(1, 5, 0));
	blocks.put("Полірований андезіт", new CustomBlock(1, 6, 0));
	blocks.put("Трава", new CustomBlock(2, 0, 0));
	blocks.put("Земля", new CustomBlock(3, 0, 0));
	blocks.put("Груба земля", new CustomBlock(3, 1, 0));
	blocks.put("Подзол", new CustomBlock(3, 2, 0));
	blocks.put("Булижник", new CustomBlock(4, 0, 0));
	blocks.put("Дубові дошки", new CustomBlock(5, 0, 0));
	blocks.put("Соснові дошки", new CustomBlock(5, 1, 0));
	blocks.put("Березові дошки", new CustomBlock(5, 2, 0));
	blocks.put("Тропічні дошки", new CustomBlock(5, 3, 0));
	blocks.put("Дошки з акації", new CustomBlock(5, 4, 0));
	blocks.put("Дошки з темного дуба", new CustomBlock(5, 5, 0));
	blocks.put("Бедрок", new CustomBlock(7, 0, 0));
	blocks.put("Пісок", new CustomBlock(12, 0, 0));
	blocks.put("Червоний пісок", new CustomBlock(12, 1, 0));
	blocks.put("Гравій", new CustomBlock(13, 0, 0));
	blocks.put("Золота руда", new CustomBlock(14, 0, 0));
	blocks.put("Залізна руда", new CustomBlock(15, 0, 0));
	blocks.put("Вугільна руда", new CustomBlock(16, 0, 0));
	blocks.put("Дубова древесина", new CustomBlock(17, 0, 0));
	blocks.put("Соснова древесина", new CustomBlock(17, 1, 0));
	blocks.put("Березова древесина", new CustomBlock(17, 2, 0));
	blocks.put("Древесина тропічного дерева", new CustomBlock(17, 3, 0));
	blocks.put("Дубове листя", new CustomBlock(18, 0, 0));
	blocks.put("Соснове листя", new CustomBlock(18, 1, 0));
	blocks.put("Березове листя", new CustomBlock(18, 2, 0));
	blocks.put("Листя тропічного дерева", new CustomBlock(18, 3, 0));
	blocks.put("Губка", new CustomBlock(19, 0, 0));
	blocks.put("Мокра губка", new CustomBlock(19, 1, 0));
	blocks.put("Скло", new CustomBlock(20, 0, 0));
	blocks.put("Лазурітова руда", new CustomBlock(21, 0, 0));
	blocks.put("Лазурітовий блок", new CustomBlock(22, 0, 0));
	blocks.put("Піщанник", new CustomBlock(24, 0, 0));
	blocks.put("Оброблений піщанник", new CustomBlock(24, 1, 0));
	blocks.put("Нотний блок", new CustomBlock(25, 0, 0));
	blocks.put("Біла шерсть", new CustomBlock(35, 0, 0));
	blocks.put("Помаранчева шерсть", new CustomBlock(35, 1, 0));
	blocks.put("Маджентова шерсть", new CustomBlock(35, 2, 0));
	blocks.put("Блакитна шерсть", new CustomBlock(35, 3, 0));
	blocks.put("Жовта шерсть", new CustomBlock(35, 4, 0));
	blocks.put("Лаймова шерсть", new CustomBlock(35, 5, 0));
	blocks.put("Рожева шерсть", new CustomBlock(35, 6, 0));
```

```
blocks.put("Сіра шерсть", new CustomBlock(35, 7, 0));
blocks.put("Світла сіра шерсть", new CustomBlock(35, 8, 0));
blocks.put("Цианова шерсть", new CustomBlock(35, 9, 0));
blocks.put("Пурпурна шерсть", new CustomBlock(35, 10, 0));
blocks.put("Синя шерсть", new CustomBlock(35, 11, 0));
blocks.put("Коричнева шерсть", new CustomBlock(35, 12, 0));
blocks.put("Зелена шерсть", new CustomBlock(35, 13, 0));
blocks.put("Червона шерсть", new CustomBlock(35, 14, 0));
blocks.put("Чорна шерсть", new CustomBlock(35, 15, 0));
blocks.put("Блок золота", new CustomBlock(41, 0, 0));
blocks.put("Блок заліза", new CustomBlock(42, 0, 0));
blocks.put("Подвійна плита з каменя", new CustomBlock(43, 0, 0));
blocks.put("Подвійна прита з піщанника", new CustomBlock(43, 1, 0));
blocks.put("Цегла", new CustomBlock(45, 0, 0));
blocks.put("Обсидіан", new CustomBlock(49, 0, 0));
blocks.put("Діамантова руда", new CustomBlock(56, 0, 0));
blocks.put("Діамантовий блок", new CustomBlock(57, 0, 0));
blocks.put("Верстак", new CustomBlock(58, 0, 0));
blocks.put("Пічка", new CustomBlock(61, 0, 0));
blocks.put("Редстоунова руда", new CustomBlock(73, 0, 0));
blocks.put("Глина", new CustomBlock(82, 0, 0));
blocks.put("Гарбуз", new CustomBlock(86, 0, 0));
blocks.put("Пекельний камінь", new CustomBlock(87, 0, 0));
blocks.put("Світильний каміннь", new CustomBlock(89, 0, 0));
blocks.put("Біле скло", new CustomBlock(95, 0, 0));
blocks.put("Помаранчеве скло", new CustomBlock(95, 1, 0));
blocks.put("Маджентове скло", new CustomBlock(95, 2, 0));
blocks.put("Блакитне скло", new CustomBlock(95, 3, 0));
blocks.put("Жовте скло", new CustomBlock(95, 4, 0));
blocks.put("Лаймове скло", new CustomBlock(95, 5, 0));
blocks.put("Рожеве скло", new CustomBlock(95, 6, 0));
blocks.put("Сіре скло", new CustomBlock(95, 7, 0));
blocks.put("Світле сіре скло", new CustomBlock(95, 8, 0));
blocks.put("Цианове скло", new CustomBlock(95, 9, 0));
blocks.put("Пурпурне скло", new CustomBlock(95, 10, 0));
blocks.put("Блакитне скло", new CustomBlock(95, 11, 0));
blocks.put("Коричневе скло", new CustomBlock(95, 12, 0));
blocks.put("Зелене скло", new CustomBlock(95, 13, 0));
blocks.put("Червоне скло", new CustomBlock(95, 14, 0));
blocks.put("Чорне скло", new CustomBlock(95, 15, 0));
blocks.put("Оброблена кам'яна цегла", new CustomBlock(98, 3, 0));
blocks.put("Камінь ендерсвіту", new CustomBlock(121, 0, 0));
blocks.put("Лампа", new CustomBlock(123, 0, 0));
blocks.put("Ізумрудна руда", new CustomBlock(129, 0, 0));
blocks.put("Ізумрудний блок", new CustomBlock(133, 0, 0));
blocks.put("Блок редстоуна", new CustomBlock(152, 0, 0));
blocks.put("Біла глина", new CustomBlock(159, 0, 0));
blocks.put("Помаранчева глина", new CustomBlock(159, 1, 0));
blocks.put("Глина маджентового кольору", new CustomBlock(159, 2, 0));
blocks.put("Блакитна глина", new CustomBlock(159, 3, 0));
blocks.put("Жовта глина", new CustomBlock(159, 4, 0));
blocks.put("Лаймова глина", new CustomBlock(159, 5, 0));
blocks.put("Рожева глина", new CustomBlock(159, 6, 0));
blocks.put("Сіра глина", new CustomBlock(159, 7, 0));
```

```
        blocks.put("Світла сіра глина", new CustomBlock(159, 8, 0));
        blocks.put("Цианова глина", new CustomBlock(159, 9, 0));
        blocks.put("Пурпурна глина", new CustomBlock(159, 10, 0));
        blocks.put("Блакитна глина", new CustomBlock(159, 11, 0));
        blocks.put("Коричнева глина", new CustomBlock(159, 12, 0));
        blocks.put("Зелена глина", new CustomBlock(159, 13, 0));
        blocks.put("Червона глина", new CustomBlock(159, 14, 0));
        blocks.put("Чорна глина", new CustomBlock(159, 15, 0));

    }

    public static IBlock getBlock(String name){
        if(blocks == null)
            init();

        return blocks.get(name);
    }

    public static Set<String> getBlockNames(){
        if(blocks == null)
            init();

        return blocks.keySet();
    }
}
```

Для файлу Building.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class Building {

    private Way buildingPoligon;

    private int levels = 1;

    public Building(Way buildingPoligon) {
        this.buildingPoligon = buildingPoligon;
        if(buildingPoligon.getTag("building:levels") != null){
            levels = Integer.valueOf(buildingPoligon.getTag("building:levels"));
        }
    }

    public Way getBuildingPoligon() {
        return buildingPoligon;
    }

    public int getLevels() {
        return levels;
    }

}
```

Для файлу BuildingConfig.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import net.morbz.minecraft.blocks.IBlock;


public class BuildingConfig {

   private IBlock block;
   private boolean isNeeded;

   private String blockName = null;

   public BuildingConfig(IBlock block, boolean isNeeded) {
         this.block = block;
         this.isNeeded = isNeeded;
   }

   public BuildingConfig(String blockName, boolean isNeeded) {
         if(Blocks.getBlockNames().contains(blockName))
            this.block = Blocks.getBlock(blockName);
         else
            System.err.println("BlockName does not exist");

         this.blockName = blockName;
         this.isNeeded = isNeeded;
   }

   public IBlock getBlock() {
            return block;
   }

   public boolean isIsNeeded() {
         return isNeeded;
   }

   public String getBlockName() {
         return blockName;
   }

   public void setIsNeeded(boolean isNeeded) {
         this.isNeeded = isNeeded;
   }

   public void setBlockName(String blockName) {
         this.blockName = blockName;
         this.block = Blocks.getBlock(blockName);
   }

}
```

Для файлу Config.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.util.HashMap;

public class Config {

   private HashMap<String, BuildingConfig> bParams;
   private HashMap<String, HighwayConfig> hParams;
   private HashMap<String, LanduseConfig> lParams;
   private HashMap<String, WaterConfig> wParams;

   private BuildingConfig defaultBuildingConfig;
   private HighwayConfig defaultRoadConfig;
   private LanduseConfig defaultLanduseConfig;
   private WaterConfig defaultWaterConfig;

   private String worldName;
   private String savePath;
   private String osmPath;
   private String landscapePath;

   private double scale;
   private double levelsScale;
   private double landscapeScale;

   private int centreX;
   private int centreY;

   private double centreLon;
   private double centreLat;

   private int worldSize;
   private Unit unit;

   public enum Unit{BLOCK, METER};

   public Config() {


         bParams = new HashMap<String, BuildingConfig>();
         hParams = new HashMap<String, HighwayConfig>();
         lParams = new HashMap<String, LanduseConfig>();
         wParams = new HashMap<String, WaterConfig>();

         defaultBuildingConfig = new BuildingConfig("Жовта шерсть", true);
         defaultRoadConfig = new HighwayConfig("Каміння", 3, true);
         defaultLanduseConfig = new LanduseConfig("Булижник", true);
         defaultWaterConfig = new WaterConfig(true);

         scale = 1;
         landscapeScale = 1;
```

```
        levelsScale = 4;
        worldSize = 250;
}

public HashMap<String, HighwayConfig> getHighwayParams() {
        return hParams;
}

public HashMap<String, LanduseConfig> getLanduseParams() {
        return lParams;
}

public String getSavePath() {
        return savePath;
}

public void setSavePath(String savePath) {
        this.savePath = savePath;
}

public String getWorldName() {
        return worldName;
}

public void setWorldName(String worldName) {
        this.worldName = worldName;
}

public double getScale() {
        return scale;
}

public double getLandscapeScale() {
        return landscapeScale;
}

public void setLandscapeScale(double landscapeScale) {
        this.landscapeScale = landscapeScale;
}

public void setScale(double scale) {
        this.scale = scale;
}

public String getOsmPath() {
        return osmPath;
}

public void setOsmPath(String osmPath) {
        this.osmPath = osmPath;
}

public String getLandscapePath() {
        return landscapePath;
```

```
}

public void setLandscapePath(String landscapePath) {
	this.landscapePath = landscapePath;
}

public HashMap<String, BuildingConfig> getBuildingParams() {
	return bParams;
}

public void setBuildingParams(HashMap<String, BuildingConfig> bParams) {
	this.bParams = bParams;
}

public HashMap<String, WaterConfig> getWaterParams() {
	return wParams;
}

public void setWaterParams(HashMap<String, WaterConfig> wParams) {
	this.wParams = wParams;
}

public BuildingConfig getDefaultBuildingConfig() {
	return defaultBuildingConfig;
}

public void setDefaultBuildingConfig(BuildingConfig defaultBuildingConfig) {
	this.defaultBuildingConfig = defaultBuildingConfig;
}

public HighwayConfig getDefaultRoadConfig() {
	return defaultRoadConfig;
}

public void setDefaultRoadConfig(HighwayConfig defaultRoadConfig) {
	this.defaultRoadConfig = defaultRoadConfig;
}

public LanduseConfig getDefaultLanduseConfig() {
	return defaultLanduseConfig;
}

public void setDefaultLanduseConfig(LanduseConfig defaultLanduseConfig) {
	this.defaultLanduseConfig = defaultLanduseConfig;
}

public WaterConfig getDefaultWaterConfig() {
	return defaultWaterConfig;
}

public void setDefaultWaterConfig(WaterConfig defaultWaterConfig) {
	this.defaultWaterConfig = defaultWaterConfig;
}
```

```
public HashMap<String, BuildingConfig> getbParams() {
	return bParams;
}

public void setbParams(HashMap<String, BuildingConfig> bParams) {
	this.bParams = bParams;
}

public HashMap<String, HighwayConfig> gethParams() {
	return hParams;
}

public void sethParams(HashMap<String, HighwayConfig> hParams) {
	this.hParams = hParams;
}

public HashMap<String, LanduseConfig> getlParams() {
	return lParams;
}

public void setlParams(HashMap<String, LanduseConfig> lParams) {
	this.lParams = lParams;
}

public HashMap<String, WaterConfig> getwParams() {
	return wParams;
}

public void setwParams(HashMap<String, WaterConfig> wParams) {
	this.wParams = wParams;
}

public double getLevelsScale() {
	return levelsScale;
}

public void setLevelsScale(double levelsScale) {
	this.levelsScale = levelsScale;
}

public int getCentreX() {
	return centreX;
}

public void setCentreX(int centreX) {
	this.centreX = centreX;
}

public int getCentreY() {
	return centreY;
}

public void setCentreY(int centreY) {
	this.centreY = centreY;
```

```
    }

    public double getCentreLon() {
        return centreLon;
    }

    public void setCentreLon(double centreLon) {
        this.centreLon = centreLon;
    }

    public double getCentreLat() {
        return centreLat;
    }

    public void setCentreLat(double centreLat) {
        this.centreLat = centreLat;
    }

    public int getWorldSize() {
        return worldSize;
    }

    public void setWorldSize(int worldSize) {
        this.worldSize = worldSize;
    }

    public Unit getUnit() {
        return unit;
    }

    public void setUnit(Unit unit) {
        this.unit = unit;
    }

}
```

Для файлу FlatLandscapeData.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class FlatLandscapeData implements ILandscapeData {

    private double elevation;

    public FlatLandscapeData(double elevation){
        this.elevation = elevation;
    }

    @Override
    public double[][] getLandscapeFromArea(Area area) {
        double landscape[][] = new double[area.getWidth()][area.getHeight()];
```

```
        for(int i = 0; i < landscape.length; i++)
           for(int j = 0; j < landscape[0].length; j++)
                landscape[i][j] = elevation;

        return landscape;
   }

}
```

Для файлу Highway.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class Highway {

   private Way road;

   public Highway(Way road) {
        this.road = road;
   }

   public Way getRoad() {
        return road;
   }

}
```

Для файлу HighwayConfig.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import net.morbz.minecraft.blocks.IBlock;


public class HighwayConfig {

   private IBlock block;
   private float radius;
   private boolean isNeeded;

   private String blockName = null;

   public HighwayConfig(IBlock block, float radius) {
        this(block, radius, true);
   }

   public HighwayConfig(IBlock block, float radius, boolean isNeeded) {
        this.block = block;
        this.radius = radius;
        this.isNeeded = isNeeded;
   }
```

```
    public HighwayConfig(String blockName, float radius, boolean isNeeded) {
        if(Blocks.getBlockNames().contains(blockName))
            this.block = Blocks.getBlock(blockName);
        else
            System.err.println("BlockName does not exist");

        this.radius = radius;
        this.isNeeded = isNeeded;
        this.blockName = blockName;
    }

    public IBlock getBlock() {
            return block;
    }

    public float getRadius() {
            return radius;
    }

    public boolean isIsNeeded() {
        return isNeeded;
    }

    public String getBlockName() {
        return blockName;
    }

    public void setIsNeeded(boolean isNeeded) {
        this.isNeeded = isNeeded;
    }

    public void setBlockName(String blockName) {
        this.blockName = blockName;
        this.block = Blocks.getBlock(blockName);
    }

    public void setRadius(float radius) {
        this.radius = radius;
    }

}
```

Для файлу ILandscapeData.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public interface ILandscapeData {

    double[][] getLandscapeFromArea(Area area);

}
```

Для файлу InterpolatedLandscapeData.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.util.ArrayList;
import org.geotools.data.DataSourceException;
import org.minemapcreator.minemapcreator.math.MapMath;
import org.minemapcreator.minemapcreator.math.Vector2D;
import org.opengis.coverage.PointOutsideCoverageException;


public class InterpolatedLandscapeData implements ILandscapeData {

   private ArrayList<LandscapeChunk> chunks;

   private double scale;

   private double rel;

   public InterpolatedLandscapeData(String path) throws DataSourceException, IOException {
         this(path, 1.0);

   }

   public InterpolatedLandscapeData(String path, double scale) throws DataSourceException, IOException {
         this.scale = scale;
         this.rel = 0;
         setChunks(path);

   }

   private void setChunks(String path) throws DataSourceException, IOException {

         File files[] = new File(path).listFiles(new FilenameFilter(){
            @Override
            public boolean accept(File dir, String name) {
                  return name.toLowerCase().endsWith(".tif");
            }

         });

         if(files != null){
            chunks = new ArrayList();
            for(File file : files){
                  LandscapeChunk chunk = new LandscapeChunk(file);

                  if(chunk.getChunkHeight() / chunk.getRadHeight() > rel)
                     rel = chunk.getChunkHeight() / chunk.getRadHeight();

                  chunks.add(chunk);
```

```
        }
      }
}

@Override
public double[][] getLandscapeFromArea(Area area) {
      double landscape[][] = new double[area.getWidth()][area.getHeight()];

      Vector2D pointAR = area.getPointRadiansA();
      Vector2D pointBR = area.getPointRadiansB();
      Vector2D pointCR = area.getPointRadiansC();

      double height = Math.ceil((pointBR.getY() - pointAR.getY()) * rel);
      double width = Math.ceil(height * (double) area.getWidth() / (double) area.getHeight());

      double elevations[][] = new double[(int) width][(int) height];

      double x = pointAR.getX();
      double y = pointAR.getY();

      double deltaX = (pointCR.getX() - pointBR.getX()) / width;
      double deltaY = (pointBR.getY() - pointAR.getY()) / height;

      double average = 0;
      double count = 0;

      for (int i = 0; i < elevations.length; i++, x += deltaX, y = pointAR.getY()) {
        for (int j = 0; j < elevations[0].length; j++, y += deltaY) {
            double elevation = getElevation(Math.toDegrees(x), Math.toDegrees(y));
            elevations[i][j] = elevation;

            if(!Double.isNaN(elevations[i][j])){
              average += elevations[i][j];
              count++;
            }
        }
      }

      average /= count;

      if(average == 0)
        average = 2.0;

      for (int i = 0; i < elevations.length; i++) {
        for (int j = 0; j < elevations[0].length; j++) {
            if(Double.isNaN(elevations[i][j])){
              elevations[i][j] = average;
            }
        }
      }

      double handledElevations[][] = new double[(int) area.getWidth()][(int) area.getHeight()];
      double elevationsInterpolationX[][] = new double[(int) area.getWidth()][elevations[0].length];
```

```
for (int j = 0; j < (int) height; j++) {
   double line[] = new double[(int) width];

   for (int k = 0; k < (int) width; k++) {
         line[k] = elevations[k][j];
   }

   double scale = (width - 1) / (double) area.getWidth();

   for (int i = 0; i < elevationsInterpolationX.length; i++) {
         double p0;
         double p1 = Math.floor(i * scale);
         double p2 = p1 + 1;
         double p3;

         if ((int) p1 == 0) {
            p0 = p1;
         } else {
            p0 = p1 - 1;
         }

         if ((int) p2 == line.length - 1) {
            p3 = line.length - 1;
         } else {
            p3 = p2 + 1;
         }

         if((int) p1 == line.length - 1){
            p3 = p1;
            p2 = p1;
            p1 = p2 - 1;
            p0 = p1 - 1;
         }

         elevationsInterpolationX[i][j] = MapMath.getInterpolatedValue(new double[]{
            line[(int) (p0)],
            line[(int) (p1)],
            line[(int) (p2)],
            line[(int) (p3)]}, (i ) * scale - Math.floor((i ) * scale));
   }

}

for (int i = 0; i < elevationsInterpolationX.length; i++) {

   double scale = (height - 1) / (double) area.getHeight();

   for (int j = 0; j < handledElevations[0].length; j++) {
         double p0;
         double p1 = Math.floor(j * scale);
         double p2 = p1 + 1;
         double p3;

         if ((int) p1 == 0) {
```

```
            p0 = p1;
         } else {
            p0 = p1 - 1;
         }

         if ((int) p2 == elevationsInterpolationX[i].length - 1) {
            p3 = elevationsInterpolationX[i].length - 1;
         } else {
            p3 = p2 + 1;
         }

         if((int) p1 == elevationsInterpolationX[i].length - 1){
            p3 = p1;
            p2 = p1;
            p1 = p2 - 1;
            p0 = p1 - 1;
         }

         handledElevations[i][j] = MapMath.getInterpolatedValue(new double[]{
            elevationsInterpolationX[i][(int) (p0)],
            elevationsInterpolationX[i][(int) (p1)],
            elevationsInterpolationX[i][(int) (p2)],
            elevationsInterpolationX[i][(int) (p3)]}, (j ) * scale - Math.floor((j ) * scale));
   }

}

for (int i = 0; i < handledElevations.length; i++) {
   for (int j = 0; j < handledElevations[0].length; j++) {
         handledElevations[i][j] *= this.scale;
   }
}

double min = Double.MAX_VALUE;
double max = Double.MIN_VALUE;

for (int i = 0; i < handledElevations.length; i++) {
   for (int j = 0; j < handledElevations[0].length; j++) {

         if (handledElevations[i][j] < min) {
            min = handledElevations[i][j];
         }

         if (handledElevations[i][j] > max) {
            max = handledElevations[i][j];
         }
   }
}

double scale = 1;
if (max - min > 255) {
   scale = 255.0 / (max - min + 1);
}
```

```
        for (int i = 0; i < handledElevations.length; i++) {
            for (int j = 0; j < handledElevations[0].length; j++) {
                handledElevations[i][j] = (handledElevations[i][j] - min + 1) * scale;
            }
        }

        min = Double.MAX_VALUE;
        max = Double.MIN_VALUE;

        for (int i = 0; i < handledElevations.length; i++) {
            for (int j = 0; j < handledElevations[0].length; j++) {
                if (handledElevations[i][j] < min) {
                    min = handledElevations[i][j];
                }

                if (handledElevations[i][j] > max) {
                    max = handledElevations[i][j];
                }
            }
        }

        for (int i = 0; i < handledElevations.length; i++) {
            for (int j = 0; j < handledElevations[0].length; j++) {
                landscape[i][j] = (int) (handledElevations[i][j] * 255 / (max - min + 1));
            }
        }

        return landscape;
    }

    public double getElevation(double lon, double lat){
        double elevation = Double.NaN;

        for(LandscapeChunk chunk : chunks){
            try{
                elevation = chunk.getElevation(lon, lat);
            }catch(PointOutsideCoverageException e){

            }
        }

        return elevation;
    }

}
```

Для файлу LandscapeChunk.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.io.File;
import java.io.IOException;
import org.geotools.coverage.grid.GridCoverage2D;
```

```
import org.geotools.data.DataSourceException;
import org.geotools.gce.geotiff.GeoTiffReader;
import org.geotools.geometry.DirectPosition2D;
import org.opengis.coverage.PointOutsideCoverageException;


public class LandscapeChunk {

   private String path;

   private File file;
   private GridCoverage2D cov;

   private double lonResolution;
   private double latResolution;
   private double lonResolutionRadians;
   private double latResolutionRadians;

   private double chunkWidth;
   private double chunkHeight;
   private double degWidth;
   private double degHeight;
   private double radWidth;
   private double radHeight;

   public LandscapeChunk(File file) throws DataSourceException, IOException{
         this.file = file;
         GeoTiffReader reader = new GeoTiffReader(file);
         cov = reader.read(null);

         double lonResolution = cov.getEnvelope2D().getSpan(0) / cov.getRenderedImage().getWidth();
         double latResolution = cov.getEnvelope2D().getSpan(1) / cov.getRenderedImage().getHeight();

         double lonResolutionRadians = Math.toRadians(cov.getEnvelope2D().getSpan(0)) /
cov.getRenderedImage().getWidth();
         double latResolutionRadians = Math.toRadians(cov.getEnvelope2D().getSpan(1)) /
cov.getRenderedImage().getHeight();

         chunkWidth = cov.getRenderedImage().getWidth();
         chunkHeight = cov.getRenderedImage().getHeight();

         degWidth = cov.getEnvelope2D().getSpan(0);
         degHeight = cov.getEnvelope2D().getSpan(1);

         radWidth = Math.toRadians(cov.getEnvelope2D().getSpan(0));
         radHeight = Math.toRadians(cov.getEnvelope2D().getSpan(1));

   }

   public double getElevation(double lon, double lat) throws PointOutsideCoverageException{
         double elevation[] = new double[1];

         try{
            cov.evaluate(new DirectPosition2D(lon, lat).getDirectPosition(), elevation);
```

```
            }catch(PointOutsideCoverageException e){
               throw e;
            }
            return elevation[0];
    }

    public double getLonResolution() {
            return lonResolution;
    }

    public double getLatResolution() {
            return latResolution;
    }

    public double getLonResolutionRadians() {
            return lonResolutionRadians;
    }

    public double getLatResolutionRadians() {
            return latResolutionRadians;
    }

    public double getChunkWidth() {
            return chunkWidth;
    }

    public double getChunkHeight() {
            return chunkHeight;
    }

    public double getDegWidth() {
            return degWidth;
    }

    public double getDegHeight() {
            return degHeight;
    }

    public double getRadWidth() {
            return radWidth;
    }

    public double getRadHeight() {
            return radHeight;
    }

}
```

Для файлу Landuse.java

```
package org.minemapcreator.minemapcreator.mapgenerator;
```

```
public class Landuse {

   private Way area;

   public Landuse(Way area) {
         this.area = area;
   }

   public Way getArea() {
         return area;
   }

}
```

Для файлу LanduseConfig.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import net.morbz.minecraft.blocks.IBlock;

public class LanduseConfig {

   private IBlock block;
   private boolean isNeeded;

   private String blockName = null;

   public LanduseConfig(IBlock block, boolean isNeeded) {
         this.block = block;
         this.isNeeded = isNeeded;
   }

   public LanduseConfig(String blockName, boolean isNeeded) {
         if(Blocks.getBlockNames().contains(blockName))
            this.block = Blocks.getBlock(blockName);
         else
            System.err.println("BlockName does not exist");

         this.isNeeded = isNeeded;
         this.blockName = blockName;
   }

   public IBlock getBlock() {
            return block;
   }

   public boolean isIsNeeded() {
         return isNeeded;
   }

   public String getBlockName() {
         return blockName;
   }
```

```
    public void setIsNeeded(boolean isNeeded) {
        this.isNeeded = isNeeded;
    }

    public void setBlockName(String blockName) {
        this.blockName = blockName;
        this.block = Blocks.getBlock(blockName);
    }

}
```

Для файлу Map.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import net.morbz.minecraft.blocks.CustomBlock;
import net.morbz.minecraft.blocks.IBlock;
import net.morbz.minecraft.level.EmptyGenerator;
import net.morbz.minecraft.level.GameType;
import net.morbz.minecraft.world.DefaultLayers;
import net.morbz.minecraft.world.World;
import org.minemapcreator.minemapcreator.math.MapMath;
import org.minemapcreator.minemapcreator.math.MapUtill;
import org.minemapcreator.minemapcreator.math.Vector2D;

public class Map {

    private Config config;

    private OsmData osmData;
    private ILandscapeData landData;

    private Area area;

    private double landscape[][];
    private BlockLine builds[][];
    private BlockLine water[][];
    private IBlock lands[][];
    private IBlock roads[][];

    private ProgressWithComment buildProcess;
    private ProgressWithComment saveProcess;

    private Vector2D xyLonLatBias;

    public Map(Config config, OsmData osmData, ILandscapeData landData) throws OsmDataException {
        this.config = config;
        this.osmData = osmData;
        this.landData = landData;
```

```
        area = new Area(osmData.getMinPoint(), osmData.getMaxPoint(), config.getScale());
        xyLonLatBias = area.getXYFromLonLatRad(MapMath.pointToRadians(new
Vector2D(config.getCentreLon(), config.getCentreLat())));

        //this.width = area.getWidth();
        //this.height = area.getHeight();

        this.landscape = new double[area.getWidth()][area.getHeight()];
        this.builds = new BlockLine[area.getWidth()][area.getHeight()];
        this.water = new BlockLine[area.getWidth()][area.getHeight()];
        this.roads = new IBlock[area.getWidth()][area.getHeight()];
        this.lands = new IBlock[area.getWidth()][area.getHeight()];

        this.landscape = landData.getLandscapeFromArea(area);
        fillReservoirs();
        fillLanduses();
        fillRoads();
        fillBuildings();
    }

    private void setBlock(int x, int y, int z, IBlock block, World world){
        int xBias = (int) Math.round(xyLonLatBias.getX());
        int yBias = (int) Math.round(xyLonLatBias.getY());

        int pointX = x + config.getCentreX() - xBias;
        int pointY = y;
        int pointZ = area.getHeight() - 1 - z + config.getCentreY() - yBias;

        int worldSideHalf = 0;

        if(config.getUnit() == Config.Unit.BLOCK){
            worldSideHalf = config.getWorldSize() / 2;
        }else{
            worldSideHalf = (int) Math.round((config.getWorldSize() * config.getScale()) / 2.0);
        }

        if(pointY < 0 || pointY > 255)
            return;

        if(pointX < config.getCentreX() - worldSideHalf || pointX > config.getCentreX() + worldSideHalf - 1)
            return;

        if(pointZ < config.getCentreY() - worldSideHalf || pointZ > config.getCentreY() + worldSideHalf - 1)
            return;

        world.setBlock(pointX, y, pointZ, block);
    }

    public void build() throws SaveWorldException {
        String path = config.getSavePath();
        DefaultLayers layers = new DefaultLayers();
        net.morbz.minecraft.level.Level level = new net.morbz.minecraft.level.Level(config.getWorldName(),
new EmptyGenerator());
```

```
        level.setMapFeatures(false);
        level.setAllowCommands(true);
        level.setGameType(GameType.CREATIVE);
        World world = new World(level, layers);

        int count = 0;
        int all = landscape.length + lands.length + water.length + roads.length + builds.length;

        if(buildProcess != null){
           buildProcess.set(0, "");
        }

        for (int i = 0; i < landscape.length; i++) {
           for (int j = 0; j < landscape[0].length; j++) {
                int elevation = (int) landscape[i][j];
                for (int k = 0; k < elevation; k++) {
                   setBlock(i, k, j, new CustomBlock(3, 0, 0), world);
                }
                setBlock(i, elevation - 1, j, new CustomBlock(2, 0, 0), world);
           }
           count++;
           if(buildProcess != null){
                buildProcess.set((int) Math.round((double) count / (double) all * 100.0), "Побудова
рельєфу");
           }
        }

        for (int i = 0; i < lands.length; i++) {
           for (int j = 0; j < lands[0].length; j++) {
                if (lands[i][j] != null) {
                   int elevation = (int) landscape[i][j];
                   setBlock(i, elevation - 1, j, lands[i][j], world);
                }
           }
           count++;
           if(buildProcess != null){
                buildProcess.set((int) Math.round((double) count / (double) all * 100.0), "Побудова
земельних ділянок");
           }
        }

        for (int i = 0; i < water.length; i++) {
           for (int j = 0; j < water[0].length; j++) {
                if (water[i][j] != null) {
                   int elevation = (int) water[i][j].getElevation();
                   if(elevation == -1){
                        elevation = (int) landscape[i][j];
                   }

                   for(int k = elevation; k >= landscape[i][j]; k--)
                        setBlock(i, elevation - 1, j, water[i][j].getBlock(), world);
                }
           }
           count++;
```

```
            if(buildProcess != null){
                buildProcess.set((int) Math.round((double) count / (double) all * 100.0), "Побудова водойм");
            }
        }

        for (int i = 0; i < roads.length; i++) {
            for (int j = 0; j < roads[0].length; j++) {
                if (roads[i][j] != null) {
                    int elevation = (int) landscape[i][j];
                    setBlock(i, elevation - 1, j, roads[i][j], world);
                }
            }
            count++;
            if(buildProcess != null){
                buildProcess.set((int) Math.round((double) count / (double) all * 100.0), "Побудова доріг");
            }
        }

        for (int i = 0; i < builds.length; i++) {
            for (int j = 0; j < builds[0].length; j++) {
                if (builds[i][j] != null) {
                    int elevation = (int) builds[i][j].getElevation();
                    int height = (int) builds[i][j].getHeight();
                    for(int k = (int) Math.floor(landscape[i][j]); k <= height + elevation; k++){
                        setBlock(i, k, j, builds[i][j].getBlock(), world);
                    }
                }
            }
            count++;
            if(buildProcess != null){
                buildProcess.set((int) Math.round((double) count / (double) all * 100.0), "Побудова будівель");
            }
        }

        try {
            world.save(path, saveProcess);

        } catch (IOException ex) {
            throw new SaveWorldException();
        }
    }

    private void fillReservoirs() throws OsmDataException{
        HashMap<String, WaterConfig> wParams = config.getWaterParams();


        for (String type : osmData.getReservoirTypes()) {
            IBlock block = config.getDefaultWaterConfig().getBlock();
            boolean isNeeded = config.getDefaultWaterConfig().isIsNeeded();

            if (wParams.get(type) != null) {
                block = wParams.get(type).getBlock();
```

```
                isNeeded = wParams.get(type).isIsNeeded();
            }

            if(!isNeeded) continue;

            for (Reservoir reservoir : osmData.getReservoirs().get(type)) {
                drawReservoir(reservoir);
            }
        }

}

private void fillLanduses() throws OsmDataException {
        HashMap<String, LanduseConfig> lParams = config.getLanduseParams();


        for (String type : osmData.getLanduseTypes()) {
            IBlock block = config.getDefaultLanduseConfig().getBlock();
            boolean isNeeded = config.getDefaultLanduseConfig().isIsNeeded();

            if (lParams.get(type) != null) {
                block = lParams.get(type).getBlock();
                isNeeded = lParams.get(type).isIsNeeded();
            }

            if(!isNeeded) continue;

            for (Landuse landuse : osmData.getLanduses().get(type)) {
                drawLanduse(landuse, block);
            }
        }

}

private void fillRoads() throws OsmDataException {
        HashMap<String, HighwayConfig> hParams = config.getHighwayParams();


        for (String type : osmData.getHighwayTypes()) {
            IBlock block = config.getDefaultRoadConfig().getBlock();
            float radius = config.getDefaultRoadConfig().getRadius();
            boolean isNeeded = config.getDefaultRoadConfig().isIsNeeded();

            if (hParams.get(type) != null) {
                block = hParams.get(type).getBlock();
                radius = hParams.get(type).getRadius();
                isNeeded = hParams.get(type).isIsNeeded();
            }

            if(!isNeeded) continue;

            for (Highway highway : osmData.getHighways().get(type)) {
                drawRoad(highway, radius, block);
            }
```

```
        }

}

private void fillBuildings() throws OsmDataException{

        HashMap<String, BuildingConfig> bParams = config.getBuildingParams();


        for (String type : osmData.getBuildingTypes()) {
           IBlock block = config.getDefaultBuildingConfig().getBlock();
           boolean isNeeded = config.getDefaultBuildingConfig().isIsNeeded();

           if (bParams.get(type) != null) {
                 block = bParams.get(type).getBlock();
                 isNeeded = bParams.get(type).isIsNeeded();
           }

           if(!isNeeded) continue;

           for (Building building : osmData.getBuildings().get(type)) {
                 drawBuilding(building, building.getLevels(), block);
           }
        }

}

private void setRoadPoint(int x, int y, IBlock block) {
        setPointOnArray(x, y, roads, block);
}

private void setPointOnArray(int x, int y, IBlock array[][], IBlock block) {
        if (x < 0 || x >= array.length || y < 0 || y >= array[0].length) {
           return;
        }

        array[x][y] = block;
}

private void drawCircle(float x, float y, float radius, IBlock block) {
        float x0 = x - radius;
        float y0 = y - radius;

        for (float i = x0; i <= x0 + radius * 2; i++) {
           for (float j = y0; j <= y0 + radius * 2; j++) {
                 if (((i - x) * (i - x) + (j - y) * (j - y)) <= radius * radius) {
                    setRoadPoint((int) i, (int) j, block);

                 }
           }
        }
}

private void drawRoad(Highway road, float radius, IBlock block) {
```

```
        ArrayList<Node> nodes = road.getRoad().getPoints();
        Node nodeArray[] = new Node[nodes.size()];
        nodes.toArray(nodeArray);

        ArrayList<float[]> points = new ArrayList<float[]>();

        for (int i = 0; i < nodeArray.length; i++) {
          Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
          points.add(new float[]{(float) xy.getX(), (float) xy.getY()});
        }

        for (int i = 0; i < points.size() - 1; i++) {
          drawRoadLine((int) (points.get(i)[0]), (int) (points.get(i)[1]), (int) (points.get(i + 1)[0]), (int)
(points.get(i + 1)[1]), (int) Math.round(radius * config.getScale()), block);
        }
    }

    private void drawRoadLine(float x1, float y1, float x2, float y2, float radius, IBlock block) {
        MapUtill.drawLine((int) Math.round(x1), (int) Math.round(y1), (int) Math.round(x2), (int)
Math.round(y2), (x, y) -> {
          setRoadPoint(x, y, block);
          drawCircle(x, y, radius, block);
        });
    }

    private void drawReservoir(Reservoir reservoir){
        if(reservoir.getType() == Reservoir.ReservoirType.AREA && reservoir.getRelation() != null){
          HashMap<String, Way> polygons = reservoir.getRelation().getWays().get("outer");

          double elevation = 0;

          for(java.util.Map.Entry<String, Way> relkv : polygons.entrySet()){
              ArrayList<Node> nodes = relkv.getValue().getPoints();
              Node nodeArray[] = new Node[nodes.size()];
              nodes.toArray(nodeArray);

              ArrayList<Vector2D> points = new ArrayList<>();

              for (int i = 0; i < nodeArray.length; i++) {
                Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
                points.add(xy);
              }

              elevation = Math.max(elevation, getOptimalElevationForReservoir(points));
          }

          IBlock waterBlock = new CustomBlock(9, 0, 0);
          BlockLine w = new BlockLine(waterBlock, (int) Math.round(elevation));

          for(java.util.Map.Entry<String, Way> relkv : polygons.entrySet()){
              ArrayList<Node> nodes = relkv.getValue().getPoints();
              Node nodeArray[] = new Node[nodes.size()];
              nodes.toArray(nodeArray);
```

```
                ArrayList<Vector2D> points = new ArrayList<>();

                for (int i = 0; i < nodeArray.length; i++) {
                    Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
                    points.add(xy);
                }

                MapUtill.drawPolygon(points, (int x, int y) -> {
                    if (x < 0 || x >= water.length || y < 0 || y >= water[0].length) {
                        return;
                    }

                    water[x][y] = w;
                });
            }
            return;
        }


        ArrayList<Node> nodes = reservoir.getArea().getPoints();
        Node nodeArray[] = new Node[nodes.size()];
        nodes.toArray(nodeArray);

        ArrayList<Vector2D> points = new ArrayList<>();

        for (int i = 0; i < nodeArray.length; i++) {
            Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
            points.add(xy);
        }

        if(reservoir.getType() == Reservoir.ReservoirType.AREA){
            double elevation = getOptimalElevationForReservoir(points);

            IBlock waterBlock = new CustomBlock(9, 0, 0);
            BlockLine w = new BlockLine(waterBlock, (int) Math.round(elevation));

            MapUtill.drawPolygon(points, (int x, int y) -> {
                if (x < 0 || x >= water.length || y < 0 || y >= water[0].length) {
                    return;
                }

                water[x][y] = w;
            });
        }

        if(reservoir.getType() == Reservoir.ReservoirType.RIVER){
            for(int i = 0; i < points.size() - 1; i++){
                IBlock waterBlock = new CustomBlock(9, 0, 0);
                BlockLine w = new BlockLine(waterBlock, -1);

                drawReservoirLine((int) points.get(i).getX(), (int) points.get(i).getY(), (int) points.get(i +
1).getX(), (int) points.get(i + 1).getY(), w);
            }
        }
```

```
        if(reservoir.getType() == Reservoir.ReservoirType.DOT){
            IBlock waterBlock = new CustomBlock(9, 0, 0);
            BlockLine w = new BlockLine(waterBlock, -1);

            setWater((int)points.get(0).getX(), (int) points.get(0).getY(), w);
            System.out.println("water");

        }

    }

    private double getOptimalElevationForBuilding(int xBias, int yBias, int area[][]){

        double average = 0;
        double D = 0;
        int count = 0;

        for(int i = 0; i < area.length; i++){
            for(int j = 0; j < area[0].length; j++){
                if(area[i][j] == 1/** || area[i][j] == 0**/){
                    if(i + xBias < 0 || i + xBias >= landscape.length || j + yBias < 0 || j + yBias >=
landscape[0].length)
                        continue;

                    average += landscape[i + xBias][j + yBias];
                    count++;
                }
            }
        }

        if(count == 0) return 0;

        average /= (double) count;
        count = 0;

        for(int i = 0; i < area.length; i++){
            for(int j = 0; j < area[0].length; j++){
                if(area[i][j] == 1/** || area[i][j] == 0**/){
                    if(i + xBias < 0 || i + xBias >= landscape.length || j + yBias < 0 || j + yBias >=
landscape[0].length)
                        continue;

                    D += Math.pow((average - landscape[i + xBias][j + yBias]), 2);
                    count++;
                }
            }
        }

        D /= count;

        return average + (Math.sqrt(D) * 3);
    }
```

```
    private double getOptimalElevationForReservoir(ArrayList<Vector2D> polygon){

        int points[][] = new int[polygon.size()][2];

        for (int i = 0; i < polygon.size(); i++) {
            points[i][0] = (int) Math.round(polygon.get(i).getX());
            points[i][1] = (int) Math.round(polygon.get(i).getY());
        }

        int maxX = Integer.MIN_VALUE;
        int maxY = Integer.MIN_VALUE;
        int minX = Integer.MAX_VALUE;
        int minY = Integer.MAX_VALUE;

        for (int i = 0; i < points.length; i++) {
            maxX = Math.max(maxX, points[i][0]);
            maxY = Math.max(maxY, points[i][1]);
            minX = Math.min(minX, points[i][0]);
            minY = Math.min(minY, points[i][1]);
        }

        int width = maxX - minX + 1;
        int height = maxY - minY + 1;
        int canvas[][] = new int[width][height];

        for (int i = 0; i < points.length - 1; i++) {
            MapUtill.drawLineToArray(points[i][0] - minX, points[i][1] - minY, points[i + 1][0] - minX, points[i +
1][1] - minY, canvas);
        }

        int count = 0;
        double average = 0;
        double disp = 0;

        for(int i = 0; i < canvas.length; i++){
            for(int j = 0; j < canvas[0].length; j++){
                    if(canvas[i][j] == 1){
                        average += landscape[i + minX][j + minY];
                        count++;
                    }
            }
        }

        average /= count;

        return average;
    }

    private void drawLanduse(Landuse landuse, IBlock block) {
        ArrayList<Node> nodes = landuse.getArea().getPoints();
        Node nodeArray[] = new Node[nodes.size()];
        nodes.toArray(nodeArray);

        ArrayList<Vector2D> points = new ArrayList<>();
```

```
        for (int i = 0; i < nodeArray.length; i++) {
            Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
            points.add(xy);
        }

        MapUtill.drawPolygon(points, (int x, int y) -> {
            if (x < 0 || x >= lands.length || y < 0 || y >= lands[0].length) {
                return;
            }

            lands[x][y] = block;
        });
    }

    private void drawBuilding(Building building, int levels, IBlock block){
        ArrayList<Node> nodes = building.getBuildingPoligon().getPoints();
        Node nodeArray[] = new Node[nodes.size()];
        nodes.toArray(nodeArray);

        ArrayList<Vector2D> points = new ArrayList<>();
        ArrayList<Vector2D> pointsWithBeas = new ArrayList<>();

        for (int i = 0; i < nodeArray.length; i++) {
            Vector2D xy = area.getXYFromLonLatRad(nodeArray[i].getLonLatRadians());
            points.add(xy);
        }

        int maxX = Integer.MIN_VALUE;
        int maxY = Integer.MIN_VALUE;
        int minX = Integer.MAX_VALUE;
        int minY = Integer.MAX_VALUE;

        for (int i = 0; i < points.size(); i++) {
            maxX = (int) Math.round(Math.max(maxX, points.get(i).getX()));
            maxY = (int) Math.round(Math.max(maxY, points.get(i).getY()));
            minX = (int) Math.round(Math.min(minX, points.get(i).getX()));
            minY = (int) Math.round(Math.min(minY, points.get(i).getY()));
        }

        int width = maxX - minX + 1;
        int height = maxY - minY + 1;

        int canvas[][] = new int[width][height];

        for(int i = 0; i < points.size(); i++){
            pointsWithBeas.add(new Vector2D(points.get(i).getX() - minX, points.get(i).getY() - minY));
        }

        MapUtill.drawPolygon(pointsWithBeas, (int x , int y) -> {
            canvas[(int) Math.round(x)][(int) Math.round(y) ] = 1;
        });

        double optimalElevation = getOptimalElevationForBuilding(minX, minY, canvas);
```

```
        BlockLine b = new BlockLine(block, (int) Math.round(optimalElevation), (int) Math.round(levels *
config.getLevelsScale()));

        MapUtill.drawPolygon(points, (int x, int y) -> {
          if (x < 0 || x >= builds.length || y < 0 || y >= builds[0].length) {
                return;
          }

          builds[x][y] = b;
        });
  }



  private void drawReservoirLine(int x1, int y1, int x2, int y2, BlockLine blockLine){
        MapUtill.drawLine(x1, y1, x2, y2, (int x, int y) -> {
          setWater(x, y, blockLine);
        });
  }

  private void setWater(int x, int y, BlockLine blockLine){
        if(x >= water.length || y >= water[0].length)
          return;

        water[x][y] = blockLine;
  }

  public void setSaveProcess(ProgressWithComment saveProcess) {
        this.saveProcess = saveProcess;
  }

  public void setBuildProcess(ProgressWithComment buildProcess) {
        this.buildProcess = buildProcess;
  }

}
```

Для файлу Node.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import org.minemapcreator.minemapcreator.math.MapMath;
import org.minemapcreator.minemapcreator.math.Vector2D;


public class Node {

  private long id;

  private double lat;
  private double lon;

  public Node(String id, String lat, String lon) throws NodeException{
```

```
            try{
               this.id = Long.valueOf(id);
               this.lat = Double.valueOf(lat);
               this.lon = Double.valueOf(lon);
            }catch(NumberFormatException e){
               throw new NodeException();
            }
   }

   public long getId() {
            return id;
   }

   public Vector2D getLonLat(){
      return new Vector2D(lon, lat);
   }

   public Vector2D getLonLatRadians(){
      return MapMath.pointToRadians(getLonLat());
   }

   public double getLat() {
            return lat;
   }

   public double getLon() {
            return lon;
   }

}
```

Для файлу NodeException.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class NodeException extends Exception{

}
```

Для файлу OsmData.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.HashSet;
```

```
import org.minemapcreator.minemapcreator.math.Vector2D;
import org.minemapcreator.minemapcreator.xmlparser.XmlNode;
import org.minemapcreator.minemapcreator.xmlparser.XmlParseException;
import org.minemapcreator.minemapcreator.xmlparser.OsmParser;


public class OsmData {

   private OsmParser parser;

   private HashMap<String, ArrayList<Building>> buildings = null;
   private HashMap<String,  ArrayList<Reservoir>> reservoirs = null;
   private HashMap<String, ArrayList<Highway>> highways = null;
   private HashMap<String, ArrayList<Landuse>> landuses = null;
   private HashSet<String> highwayTypes = null;
   private HashSet<String> landuseTypes = null;
   private HashSet<String> buildingTypes = null;
   private HashSet<String> reservoirTypes = null;

   private HashMap<String, Node> nodes = null;
   private HashMap<String, Way> ways = null;
   private HashMap<String, Relation> relations = null;

   private Vector2D minPoint;
   private Vector2D maxPoint;

   private ProgressWithComment progress;

   public OsmData(String path, ProgressWithComment progress) throws OsmDataException,
XmlParseException {
         this.progress = progress;
         XmlNode node = readXml(path);

         parser = new OsmParser(node);
         XmlNode bounds = parser.getObjectsByName("bounds").get(0);

         try{
            minPoint = new Vector2D(Double.valueOf(bounds.getParam("minlon")),
Double.valueOf(bounds.getParam("minlat")));
            maxPoint = new Vector2D(Double.valueOf(bounds.getParam("maxlon")),
Double.valueOf(bounds.getParam("maxlat")));
         }catch(NumberFormatException e){
            throw new OsmDataException();
         }
         setMapObjects();

   }

   private XmlNode readXml(String path) throws XmlParseException{
         StringBuilder text = new StringBuilder();

         if(progress != null){
            progress.set(0, "Завантаження файлу: " + path);
         }
```

```
        try {
            File file = new File(path);
            BufferedReader bReader;

            long size = file.length();
            long reads = 0;
            char[] buffer = new char[16384];
            int readChars;

            bReader = new BufferedReader(new FileReader(file));

            while((readChars = bReader.read(buffer)) != -1) {
                reads += readChars;
                text.append(buffer);

                if(progress != null){
                    progress.set((int) Math.round((double) reads / (double) size * 100.0), "Завантаження файлу: " + path);
                }
            }
            bReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        if(progress != null){
            progress.set(100, "Завантаження файлу: " + path);
        }

        XmlNode node = new XmlNode();

        if(progress != null){
            progress.set(0, "Аналіз файлу: " + path);
        }

        node.parseFromText(text, (progress) -> {
            this.progress.set(progress, "Аналіз файлу: " + path);
        });

        return node;

    }

    private HashMap<String, Node> getNodes() throws NodeException {
        if(nodes != null)
            return nodes;

        nodes = new HashMap<String, Node>();
        ArrayList<XmlNode> nodeList = parser.getObjectsByName("node");

        for(XmlNode node : nodeList) {
            nodes.put(node.getParam("id"),new Node(node.getParam("id"), node.getParam("lat"), node.getParam("lon")));
```

```
        }

        return nodes;
    }

    private HashMap<String, Way> getWays() throws WayException {
        if(ways != null)
            return ways;

        ways = new HashMap<String, Way>();
        ArrayList<XmlNode> wayList = parser.getObjectsByName("way");
        HashMap<String, Node> nodes;
        try {
            nodes = getNodes();

            for(XmlNode way : wayList) {
                ArrayList<XmlNode> nds = way.getChildren("nd");
                ArrayList<XmlNode> tags = way.getChildren("tag");
                ArrayList<Node> points = new ArrayList<Node>();
                HashMap<String, String> kv = new HashMap<String, String>();

                for(XmlNode node : nds) {
                    points.add(nodes.get(node.getParam("ref")));
                }

                for(XmlNode tag : tags) {
                    kv.put(tag.getParam("k"), tag.getParam("v"));
                }

                ways.put(way.getParam("id"), new Way(way.getParam("id"), points, kv));
            }

        } catch (NodeException ex) {
            throw new WayException();
        }

        return ways;
    }

    private HashMap<String, Relation> getRelations() throws  WayException {
        if(relations != null)
            return relations;

        HashMap<String, ArrayList<String>> needs = new HashMap<String, ArrayList<String>>();

        relations = new HashMap<String, Relation>();
        ArrayList<XmlNode> relationList = parser.getObjectsByName("relation");
        HashMap<String, Way> ways = getWays();

        for(XmlNode relation : relationList) {
            ArrayList<XmlNode> members = relation.getChildren("member");
            ArrayList<XmlNode> tags = relation.getChildren("tag");
           HashMap<String, HashMap<String, Way>> lines = new HashMap<String, HashMap<String, Way>>();
            HashMap<String, HashMap<String, Relation>> children = new HashMap<String, HashMap<String,
```

```
Relation>>();
            HashMap<String, String> kv = new HashMap<String, String>();
            Relation result;

            for(XmlNode member : members) {

                  if(member.getParam("type").equals("way")){
                    if(lines.get(member.getParam("role")) == null)
                          lines.put(member.getParam("role"), new HashMap<String, Way>());

                    lines.get(member.getParam("role")).put(member.getParam("ref"),
ways.get(member.getParam("ref")));
                    continue;
                  }
            }

            for(XmlNode tag : tags) {
                  kv.put(tag.getParam("k"), tag.getParam("v"));
            }

            relations.put(relation.getParam("id"),
                  new Relation(
                        relation.getParam("id"),
                        lines,
                        null,
                        kv));
        }

        return relations;
    }

    private void setMapObjects() throws OsmDataException {
        buildings = new HashMap<String, ArrayList<Building>>();
        reservoirs = new HashMap<String, ArrayList<Reservoir>>();
        highways = new HashMap<String, ArrayList<Highway>>();
        landuses = new HashMap<String, ArrayList<Landuse>>();
        highwayTypes = new HashSet<String>();
        landuseTypes = new HashSet<String>();
        buildingTypes = new HashSet<String>();
        reservoirTypes = new HashSet<String>();

        try {
            setRelationObjects();
            setWayObjects();
        } catch (WayException ex) {
            throw new OsmDataException();
        }
    }

    private void setRelationObjects() throws WayException{
        HashMap<String, Relation> relations = getRelations();

        for(java.util.Map.Entry<String, Relation> relkv : relations.entrySet()) {
            HashMap<String, String> tags = relkv.getValue().getTags();
```

```
            for(java.util.Map.Entry<String, String> tag : tags.entrySet()) {
                if(tag.getKey().equals("water") && tag.getValue().equals("reservoir")) {
                    if(!reservoirs.containsKey(tag.getValue()))
                        reservoirs.put(tag.getValue(), new ArrayList<Reservoir>());

                    reservoirs.get(tag.getValue()).add(new Reservoir(relkv.getValue()));
                    reservoirTypes.add(tag.getValue());
                }
            }
        }
    }

    private void setWayObjects() throws WayException {
        HashMap<String, Way> ways = getWays();

        for(java.util.Map.Entry<String, Way> waykv : ways.entrySet()) {
            HashMap<String, String> tags = waykv.getValue().getTags();
            for(java.util.Map.Entry<String, String> tag : tags.entrySet()) {
                if(tag.getKey().equals("building")) {
                    Building building = new Building(waykv.getValue());

                    if(!buildings.containsKey(String.valueOf(building.getLevels())))
                        buildings.put(String.valueOf(building.getLevels()), new ArrayList<Building>());

                    buildings.get(String.valueOf(building.getLevels())).add(building);
                    buildingTypes.add(String.valueOf(building.getLevels()));
                    continue;
                }else{
                    if(tag.getKey().equals("highway")) {
                        String tagv = waykv.getValue().getTag("highway");

                        if(highways.get(tagv) == null) {
                            highways.put(tagv, new ArrayList<Highway>());
                        }

                        highwayTypes.add(tagv);
                        Highway highway = new Highway(waykv.getValue());
                        highways.get(tagv).add(highway);
                        continue;
                    }else {
                        if(tag.getKey().equals("landuse")) {

                            String tagv = waykv.getValue().getTag("landuse");

                            if(tag.getValue().equals("basin") || tag.getValue().equals("reservoir")){
                                if(!reservoirs.containsKey(tag.getValue()))
                                    reservoirs.put(tag.getValue(), new ArrayList<Reservoir>());

                                reservoirs.get(tag.getValue()).add(new Reservoir(waykv.getValue()));
                                reservoirTypes.add(tag.getValue());
                                continue;
                            }

                            if(landuses.get(tagv) == null) {
```

```
                        landuses.put(tagv, new ArrayList<Landuse>());
                    }

                    landuseTypes.add(tagv);

                    Landuse landuse = new Landuse(waykv.getValue());
                    landuses.get(tagv).add(landuse);
                    continue;
                }else{
                    if(tag.getKey().equals("water") || tag.getKey().equals("waterway")){
                        if(!reservoirs.containsKey(tag.getValue()))
                            reservoirs.put(tag.getValue(), new ArrayList<Reservoir>());

                        reservoirs.get(tag.getValue()).add(new Reservoir(waykv.getValue()));
                        reservoirTypes.add(tag.getValue());

                        continue;
                    }
                    if(tag.getKey().equals("natural")){
                        if(tag.getValue().equals("water") && tag.getValue().equals("waterland")){
                            if(!reservoirs.containsKey(tag.getValue()))
                                reservoirs.put(tag.getValue(), new ArrayList<Reservoir>());

                            reservoirs.get(tag.getValue()).add(new Reservoir(waykv.getValue()));
                            reservoirTypes.add(tag.getValue());

                        }
                        continue;
                    }
                }
            }
        }
    }
}

public HashMap<String, ArrayList<Highway>> getHighways() throws OsmDataException {
    if(highways == null)
        setMapObjects();


    return highways;
}

public HashMap<String, ArrayList<Landuse>> getLanduses() throws OsmDataException {
    if(landuses == null)
        setMapObjects();

    return landuses;
}

public Vector2D getMinPoint() {
    return minPoint;
}
```

```
public Vector2D getMaxPoint() {
	return maxPoint;
}

public HashMap<String, ArrayList<Building>> getBuildings() throws OsmDataException {
	if(buildings == null)
	  setMapObjects();

	return buildings;
}

public HashMap<String, ArrayList<Reservoir>> getReservoirs() throws OsmDataException {
	if(reservoirs == null)
	  setMapObjects();

	return reservoirs;
}

public ArrayList<String> getBuildingTypes() throws OsmDataException {
	if(buildingTypes == null)
	  setMapObjects();

	ArrayList<String> result = new ArrayList<>();

	for(String str : buildingTypes){
	  result.add(str);
	}

	result.sort(new Comparator<String>(){
	  @Override
	  public int compare(String o1, String o2) {
		return Integer.valueOf(o1) - Integer.valueOf(o2);
	  }

	});

	return result;
}

public ArrayList<String> getReservoirTypes() throws OsmDataException {
	if(reservoirTypes == null)
	  setMapObjects();

	ArrayList<String> result = new ArrayList<>();

	for(String str : reservoirTypes){
	  result.add(str);
	}

	return result;
}

public ArrayList<String> getHighwayTypes() throws OsmDataException {
```

```
        if(highwayTypes == null)
            setMapObjects();

        ArrayList<String> result = new ArrayList<>();

        for(String str : highwayTypes){
            result.add(str);
        }

        return result;
    }

    public ArrayList<String> getLanduseTypes() throws OsmDataException {
        if(landuseTypes == null)
            setMapObjects();

        ArrayList<String> result = new ArrayList<>();

        for(String str : landuseTypes){
            result.add(str);
        }

        return result;
    }
}
```

Для файлу OsmDataException.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class OsmDataException extends Exception {

}
```

Для файлу PointSetter.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public interface PointSetter {

    void setPoint(int x, int y);

}
```

Для файлу Progress.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public interface Progress {
```

```
    void set(int progress);

}
```

Для файлу ProgressWithComment.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public interface ProgressWithComment {

    void set(int progress, String comment);

}
```

Для файлу Relation.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.util.HashMap;


public class Relation {

    private long id;

    private HashMap<String, HashMap<String, Way>> ways;
    private HashMap<String, HashMap<String, Relation>> relations;
    private HashMap<String, String> tags;

    public Relation(String id, HashMap<String, HashMap<String, Way>> ways, HashMap<String,
HashMap<String, Relation>> relations, HashMap<String, String> tags) throws NumberFormatException{
            try{
                this.id = Long.valueOf(id);
            }catch(NumberFormatException e){
                throw e;
            }
            this.ways = ways;
            this.relations = relations;
            this.tags = tags;
    }

    public long getId() {
            return id;
    }

    public HashMap<String, HashMap<String, Way>> getWays() {
            return ways;
    }

    public HashMap<String, HashMap<String, Relation>> getRelations() {
            return relations;
```

```
    }

    public HashMap<String, String> getTags() {
        return tags;
    }

}
```

Для файлу Reservoir.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class Reservoir {

    private Way area;
    private Relation relation;

    private ReservoirType type;

    public enum ReservoirType{
        AREA, RIVER, DOT
    }

    public Reservoir(Relation relation) {
        this.relation = relation;
        type = ReservoirType.AREA;
    }

    public Reservoir(Way area) {
        this.area = area;
        if(area.getPoints().size() == 1){
            type = ReservoirType.DOT;
        }else{
            if(area.getPoints().get(0).getId() == area.getPoints().get(area.getPoints().size() - 1).getId()){
                type = ReservoirType.AREA;
            }else{
                type = ReservoirType.RIVER;
            }
        }
    }

    public Reservoir(Way area, ReservoirType type) {
        this.area = area;
        this.type = type;
    }

    public Way getArea() {
        return area;
    }

    public Relation getRelation(){
        return relation;
```

```
    }

    public ReservoirType getType(){
        return type;
    }

}
```

Для файлу SaveWorldException.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class SaveWorldException extends Exception {

}
```

Для файлу Translator.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.util.HashMap;


public class Translator {

    private static HashMap<String, String> roadTypes;
    private static HashMap<String, String> landuseTypes;
    private static HashMap<String, String> waterTypes;

    private static boolean initRoads = false;
    private static boolean initLanduses = false;
    private static boolean initWaters = false;

    private static HashMap<String, String> getRoadTypes(){
        if(roadTypes != null) return roadTypes;

        roadTypes = new HashMap();

        roadTypes.put("motorway", "Автостради");
        roadTypes.put("trunk", "Магістралі");
        roadTypes.put("primary", "Головні дороги");
        roadTypes.put("secondary", "Міжміські дороги");
        roadTypes.put("tertiary", "Місцеві дороги");
        roadTypes.put("unclassified", "Некласифіковані дороги");
        roadTypes.put("residential", "Житлові дороги");
        roadTypes.put("motorway_link", "З'єднання з автострадами");
        roadTypes.put("trunk_link", "З'єднання з магістралями");
        roadTypes.put("primary_link", "З'єднання з головними дорогами");
        roadTypes.put("secondary_link", "З'єднання з міжміськими дорогами");
        roadTypes.put("tertiary_link", "З'єднання з місцевими дорогами");
        roadTypes.put("living_street", "Дороги житлових місцевостей");
        roadTypes.put("service", "Дороги службового призначення");
```

```
        roadTypes.put("pedestrian", "Пішохідні дороги");
        roadTypes.put("track", "Господарські дороги");
        roadTypes.put("bus_guideway", "Автобусні колії");
        roadTypes.put("escape", "Дороги для аварійних зупинок");
        roadTypes.put("raceway", "Дороги для перегонів");
        roadTypes.put("road", "Невизначені дороги");
        roadTypes.put("busway", "Автобусні дороги");
        roadTypes.put("footway", "Автобусні дороги");
        roadTypes.put("bridleway", "Дороги для конів");
        roadTypes.put("steps", "Сходи");
        roadTypes.put("corridor", "Дороги через будівлі");
        roadTypes.put("path", "Неспецифічні шляхи");
        roadTypes.put("cycleway", "Велосипедні доріжки");
        roadTypes.put("proposed", "Заплановані дороги");
        roadTypes.put("construction", "Дороги що будуються");

        return roadTypes;
    }

    private static HashMap<String, String> getLanduseTypes(){
        if(landuseTypes != null) return landuseTypes;

        landuseTypes = new HashMap();

        landuseTypes.put("commercial", "Комерційні місцевості");
        landuseTypes.put("construction", "Місцевості будівництва");
        landuseTypes.put("education", "Місцевості освіти");
        landuseTypes.put("fairground", "Місцевості ярмарку");
        landuseTypes.put("industrial", "Індустріальні місцевості");
        landuseTypes.put("residential", "Житлові масиви");
        landuseTypes.put("retail", "Роздрібні підприємства");
        landuseTypes.put("institutional", "Інституційні місцевості");
        landuseTypes.put("aquaculture", "Аква-ферми");
        landuseTypes.put("allotments", "Ділянки для місцевих");
        landuseTypes.put("farmland", "Сільськогосподарські угіддя");
        landuseTypes.put("farmyard", "Ділянки з господарськими будівлями");
        landuseTypes.put("flowerbed", "Території для квітів");
        landuseTypes.put("forest", "Ліси та посадки");
        landuseTypes.put("greenhouse_horticulture", "Території теплиць");
        landuseTypes.put("meadow", "Луги та пасовища");
        landuseTypes.put("orchard", "Плодові дерева та кущі");
        landuseTypes.put("plant_nursery", "Виробництво рослин");
        landuseTypes.put("vineyard", "Виноградники");
        landuseTypes.put("brownfield", "Території для забудов");
        landuseTypes.put("cemetery", "Цвинтарі");
        landuseTypes.put("depot", "Депо");
        landuseTypes.put("garages", "Гаражі");
        landuseTypes.put("grass", "Трава");
        landuseTypes.put("greenfield", "Території запланованих забудов");
        landuseTypes.put("landfill", "Звалище");
        landuseTypes.put("religious", "Релігійні території");
        landuseTypes.put("village_green", "Громадська земля у селі");
        landuseTypes.put("winter_sports", "Зона зимового спорту");
        landuseTypes.put("recreation_ground", "Зона загального відпочинку");
```

```
        return landuseTypes;
}

private static HashMap<String, String> getWaterTypes(){
        if(waterTypes != null) return waterTypes;

        waterTypes = new HashMap();

        waterTypes.put("basin", "Басейни");
        waterTypes.put("reservoir", "Водосховища");
        waterTypes.put("salt_pond", "Соляні стави");
        waterTypes.put("river", "Річки");
        waterTypes.put("oxbow", "Заводі");
        waterTypes.put("canal", "Канали");
        waterTypes.put("ditch", "Канави");
        waterTypes.put("lock", "Шлюзові камери");
        waterTypes.put("fish_pass", "Зони приходу риби");
        waterTypes.put("lake", "Озера");
        waterTypes.put("pond", "Стави");
        waterTypes.put("basin", "Басейни");
        waterTypes.put("lagoon", "Лагуни");
        waterTypes.put("stream_pool", "Басейни струмків");
        waterTypes.put("reflecting_pool", "Невеликі басейни");
        waterTypes.put("moat", "Рови");
        waterTypes.put("wastewater", "Стічні води");
        waterTypes.put("wastewater", "Береги річок");
        waterTypes.put("stream", "Потоки");
        waterTypes.put("tidal_channel", "Припливні канали ї");
        waterTypes.put("drain", "Дренажі");
        waterTypes.put("fairway", "Судноплавні маршрути");
        waterTypes.put("dock", "Зони для кораблів");
        waterTypes.put("boatyard", "Місця будівництва суден");
        waterTypes.put("dam", "Дамби");
        waterTypes.put("weir", "Водозливи");
        waterTypes.put("waterfall", "Водоспади");
        waterTypes.put("lock_gate", "Шлюзи");

        return waterTypes;
}

public static String getBuildingTypeTranslate(String type){
        return type + " - поверхові будівлі";
}

public static String getRoadTypeTranslate(String type){
        if(getRoadTypes().containsKey(type))
           return roadTypes.get(type);

        return type;
}

public static String getLanduseTypeTranslate(String type){
        if(getLanduseTypes().containsKey(type))
```

```
            return landuseTypes.get(type);

        return type;
    }

    public static String getWaterTypeTranslate(String type){
        if(getWaterTypes().containsKey(type))
            return waterTypes.get(type);

        return type;
    }

}
```

Для файлу WaterConfig.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import net.morbz.minecraft.blocks.IBlock;


public class WaterConfig {

    private IBlock block;
    private boolean isNeeded;

    public WaterConfig(boolean isNeeded) {
        this.isNeeded = isNeeded;
    }

    public boolean isIsNeeded() {
        return isNeeded;
    }

    public IBlock getBlock() {
        return block;
    }



}
```

Для файлу Way.java

```
package org.minemapcreator.minemapcreator.mapgenerator;

import java.util.ArrayList;
import java.util.HashMap;


public class Way {

    private long id;
```

```
    private ArrayList<Node> points;
    private HashMap<String, String> tags;

    public Way(String id, ArrayList<Node> points, HashMap<String, String> tags) throws WayException {
            try{
                this.id = Long.valueOf(id);
            }catch(NumberFormatException e){
                throw new WayException();
            }
            this.points = points;
            this.tags = tags;
    }

    public long getId() {
            return id;
    }

    public ArrayList<Node> getPoints(){
            return points;
    }

    public HashMap<String, String> getTags() {
            return tags;
    }

    public String getTag(String key) {
            return tags.get(key);
    }

    public boolean containsTag(String key) {
            return tags.containsKey(key);
    }

}
```

Для файлу WayException.java

```
package org.minemapcreator.minemapcreator.mapgenerator;


public class WayException extends Exception{

}
```

Для файлу MapMath.java

```
package org.minemapcreator.minemapcreator.math;


public class MapMath {

    public static Vector3D lonRLatRToXYZ(Vector2D point){
```

```
        return new Vector3D(Math.cos(point.getX()) * Math.abs(Math.cos(point.getY())),
                            Math.sin(point.getX()) * Math.abs(Math.cos(point.getY())),
                           Math.sin(point.getY()));
}

public static Vector2D XYZToLonRLatR(Vector3D point){
        double pointLatR = Math.asin(point.getZ());
        double pointLonR;
        if((point.getY() / Math.cos(pointLatR)) >= 0){
           pointLonR = Math.acos(point.getX() / Math.cos(pointLatR));
        }else{
           pointLonR = -Math.acos(point.getX() / Math.cos(pointLatR));
        }

        return new Vector2D(pointLonR, pointLatR);
}

public static Vector3D getCenter(Vector3D point1, Vector3D point2){
        return new Vector3D(
                (point1.getX() + point2.getX()) / 2,
                (point1.getY() + point2.getY()) / 2,
                (point1.getZ() + point2.getZ()) / 2);
}

public static Vector2D pointToRadians(Vector2D point){
        return new Vector2D(point.getX() * Math.PI / 180, point.getY() * Math.PI / 180);
}

public static Vector2D pointToDegrees(Vector2D point){
        return new Vector2D(point.getX() * 180 / Math.PI, point.getY() * 180 / Math.PI);
}

public static Vector3D getAnotherPoint(Vector3D center, Vector3D point, double z){

        double cos = z;

        double x1 = point.getX();
        double y1 = point.getY();
        double z1 = point.getZ();

        double x0 = center.getX();
        double y0 = center.getY();
        double z0 = center.getZ();

        double nx, ny, nz;

        double m = z1 - z0;
        double d = Math.sqrt((x1 - x0) * (x1 - x0) + (y1 - y0) * (y1 - y0) + m * m);

        double k = (x0 * x0 + cos * cos + y0 * y0 + m * m - d * d);
        double a = (4 * x0 * x0 + 4 * y0 * y0);
        double b = -(4 * x0 * k);
        double c = (k * k - 4 * y0 * y0 * cos * cos);
```

```
        double xq1, xq2, yq1, yq2;

        double s[] = solveSquareEq(a, b, c);

        xq1 = s[0];
        xq2 = s[1];

        if((x1 - xq1) * (x1 - xq1) > (x1 - xq2) * (x1 - xq2)){
           nx = xq1;
        }else{
           nx = xq2;
        }

        a = 1;
        b = -2 * y0;
        c = y0 * y0 + (nx - x0) * (nx - x0) + m * m - d * d;

        s = solveSquareEq(a, b, c);

        yq1 = s[0];
        yq2 = s[1];

        if((y1 - yq1) * (y1 - yq1) > (y1 - yq2) * (y1 - yq2)){
           ny = yq1;
        }else{
           ny = yq2;
        }

        nz = z1;

        return new Vector3D(nx, ny, nz);
    }

    private static double[] solveSquareEq(double a, double b , double c){
        double D = b * b - 4 * a * c;
        if(D == 0){
           return new double[]{(-b + Math.sqrt(D)) / (2 * a)};
        }

        if(D > 0){
           return new double[]{(-b + Math.sqrt(D)) / (2 * a), (-b - Math.sqrt(D)) / (2 * a)};
        }

        return null;
    }

    public static double getDistance(Vector2D pointRadians1, Vector2D pointRadians2){
        double distance =
                1000 * 2.0 * 6371 *
                Math.asin(Math.sqrt(Math.pow(Math.sin(
                        (pointRadians1.getY() - pointRadians2.getY()) / 2.0), 2) +
                        Math.cos(pointRadians2.getY()) *
                                Math.cos(pointRadians1.getY()) *
                                Math.pow(Math.sin((pointRadians1.getX() - pointRadians2.getX()) / 2.0), 2)
```

```
                ));
        return distance;
    }

    public static Vector2D getXYFromLonLat(Vector2D point, Vector2D pointAR, Vector2D pointBR, Vector2D pointCR, Vector2D pointDR){
        return getXYFromLonLatRad(pointToRadians(point), pointAR, pointBR, pointCR, pointDR);
    }

    public static Vector2D getXYFromLonLatRad(Vector2D point, Vector2D pointAR, Vector2D pointBR, Vector2D pointCR, Vector2D pointDR){
        double ap = getDistance(pointAR, point);
        double bp = getDistance(pointBR, point);
        double ab = getDistance(pointAR, pointBR);
        double dp = getDistance(pointDR, point);
        double ad = getDistance(pointAR, pointDR);

        double x = getHighOfTriangeB(ap, ab, bp);
        double y = getHighOfTriangeB(ap, ad, dp);

        return new Vector2D(x, y);
    }

    public static boolean isInside(Vector2D pointRadians, Vector2D pointRadiansA, Vector2D pointRadiansB, Vector2D pointRadiansC, Vector2D pointRadiansD){

        double ap = getDistance(pointRadiansA, pointRadians);
        double bp = getDistance(pointRadiansB, pointRadians);
        double cp = getDistance(pointRadiansC, pointRadians);
        double dp = getDistance(pointRadiansD, pointRadians);

        double ac = getDistance(pointRadiansA, pointRadiansC);
        double bd = getDistance(pointRadiansB, pointRadiansD);

        return Math.abs((ap * ap + cp * cp) - (bp * bp + dp * dp)) < 1.5;
    }

    private static double getHighOfTriangeB(double a, double b, double c){
        double p = (a + b + c) / 2.0;
        double s = Math.sqrt(p * (p - a) * (p - b) * (p - c));

        return (2 * s) / b;
    }

    public static double getHypotenuse(double a, double b){
        return Math.sqrt(a * a + b * b);
    }

    public static double getInterpolatedValue(double[] points, double x) {
        double f0 = points[1];
        double f1 = points[2];
        double fd0 = (points[2] - points[0]) / (2);
        double fd1 = (points[3] - points[1]) / (2);
```

```
        double a = 2 * f0 - 2 * f1 + fd0 + fd1;
        double b = -3 * f0 + 3 * f1 - 2 * fd0 - fd1;
        double c = fd0;
        double d = f0;

        return a * x * x * x + b * x * x + c * x + d;
    }

}
```

Для файлу MapUtill.java

```
package org.minemapcreator.minemapcreator.math;

import java.util.ArrayList;
import org.minemapcreator.minemapcreator.mapgenerator.PointSetter;


public class MapUtill {

    public static void drawPolygon(ArrayList<Vector2D> polygon, PointSetter function) {

        int points[][] = new int[polygon.size()][2];

        for (int i = 0; i < polygon.size(); i++) {
            points[i][0] = (int) Math.round(polygon.get(i).getX());
            points[i][1] = (int) Math.round(polygon.get(i).getY());
        }

        int maxX = Integer.MIN_VALUE;
        int maxY = Integer.MIN_VALUE;
        int minX = Integer.MAX_VALUE;
        int minY = Integer.MAX_VALUE;

        for (int i = 0; i < points.length; i++) {
            maxX = Math.max(maxX, points[i][0]);
            maxY = Math.max(maxY, points[i][1]);
            minX = Math.min(minX, points[i][0]);
            minY = Math.min(minY, points[i][1]);
        }

        int width = maxX - minX + 1;
        int height = maxY - minY + 1;
        int canvas[][] = new int[width][height];

        for (int i = 0; i < points.length - 1; i++) {
            MapUtill.drawLineToArray(points[i][0] - minX, points[i][1] - minY, points[i + 1][0] - minX, points[i +
1][1] - minY, canvas);
        }

        boolean isChange = false;
        boolean canChange = false;
```

```
do {
   isChange = false;

   for (int i = 0; i < width; i++) {
         canChange = true;
         for (int j = 0; j < height; j++) {
            if (canvas[i][j] == 1) {
                  canChange = false;
            }

            if (canvas[i][j] == -1) {
                  canChange = true;
            }

            if (canvas[i][j] == 0 && canChange) {
                  isChange = true;
                  canvas[i][j] = -1;
            }
         }
   }

   for (int i = 0; i < width; i++) {
         canChange = true;
         for (int j = height - 1; j >= 0; j--) {
            if (canvas[i][j] == 1) {
                  canChange = false;
            }

            if (canvas[i][j] == -1) {
                  canChange = true;
            }

            if (canvas[i][j] == 0 && canChange) {
                  isChange = true;
                  canvas[i][j] = -1;
            }
         }
   }

   for (int i = 0; i < height; i++) {
         canChange = true;
         for (int j = 0; j < width; j++) {
            if (canvas[j][i] == 1) {
                  canChange = false;
            }

            if (canvas[j][i] == -1) {
                  canChange = true;
            }

            if (canvas[j][i] == 0 && canChange) {
                  isChange = true;
                  canvas[j][i] = -1;
            }
```

```
                }
            }

            for (int i = 0; i < height; i++) {
                canChange = true;
                for (int j = width - 1; j >= 0; j--) {
                    if (canvas[j][i] == 1) {
                        canChange = false;
                    }

                    if (canvas[j][i] == -1) {
                        canChange = true;
                    }

                    if (canvas[j][i] == 0 && canChange) {
                        isChange = true;
                        canvas[j][i] = -1;
                    }
                }
            }

        } while (isChange);

        for (int i = 0; i < width; i++) {
            for (int j = 0; j < height; j++) {
                if(canvas[i][j] == 1 || canvas[i][j] == 0)
                    function.setPoint(i + minX, j + minY);
            }
        }
    }

    public static void drawLine(int x1, int y1, int x2, int y2, PointSetter function) {
        float xb, yb, xe, ye;
        int x, y;

        if (y1 > y2) {
            xb = x1;
            yb = y1;
            xe = x2;
            ye = y2;
        } else {
            xb = x2;
            yb = y2;
            xe = x1;
            ye = y1;
        }

        float xeb = xe - xb;
        float yeb = ye - yb;
        double s = 0;

        if (xeb != 0 && yeb != 0) {
            s = yeb / xeb;
        }
```

```
        if (Math.abs(s) < 1) {
            if (xeb != 0) {
                if (xb < xe) {
                    for (float i = xb; i <= xe; i++) {
                        x = (int) (i);
                        y = (int) (s * (i - xb) + yb);
                        function.setPoint(x, y);
                    }
                } else {
                    for (float i = xb; i >= xe; i--) {
                        x = (int) (i);
                        y = (int) (s * (i - xb) + yb);
                        function.setPoint(x, y);
                    }
                }
            } else {
                if (yb < ye) {
                    for (float i = yb; i <= ye; i++) {
                        x = (int) (xb);
                        y = (int) (i);
                        function.setPoint(x, y);
                    }
                } else {
                    for (float i = yb; i >= ye; i--) {
                        x = (int) (xb);
                        y = (int) (i);
                        function.setPoint(x, y);
                    }
                }
            }
        } else {
            if (yb < ye) {
                for (float i = yb; i <= ye; i++) {
                    x = (int) ((i - yb) * (1 / s) + xb);
                    y = (int) (i);
                    function.setPoint(x, y);
                }
            } else {
                for (float i = yb; i >= ye; i--) {
                    x = (int) ((i - yb) * (1 / s) + xb);
                    y = (int) (i);
                    function.setPoint(x, y);
                }
            }
        }
    }

    public static void drawLineToArray(int x1, int y1, int x2, int y2, int array[][]) {
        drawLine(x1, y1, x2, y2, (int x, int y) -> {
            array[x][y] = 1;
        });
    }
```

```
}
```

Для файлу Vector2D.java

```
package org.minemapcreator.minemapcreator.math;


public class Vector2D {

   private double x;
   private double y;

   public Vector2D() {
          this(0, 0);
   }

   public Vector2D(double x, double y) {
          this.x = x;
          this.y = y;
   }

   public double getX() {
          return x;
   }

   public void setX(double x) {
          this.x = x;
   }

   public double getY() {
          return y;
   }

   public void setY(double y) {
          this.y = y;
   }
}
```

Для файлу Vector3D.java

```
package org.minemapcreator.minemapcreator.math;


public class Vector3D {

   private double x;
   private double y;
   private double z;

   public Vector3D() {
          this(0, 0,0);
   }
```

```
    public Vector3D(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getZ() {
        return z;
    }

    public void setX(double x) {
        this.x = x;
    }

    public void setY(double y) {
        this.y = y;
    }

    public void setZ(double z) {
        this.z = z;
    }

}
```

Для файлу OsmParser.java

```
package org.minemapcreator.minemapcreator.xmlparser;

import java.util.ArrayList;


public class OsmParser {

    private XmlNode root;

    public OsmParser(XmlNode root) {
        this.root = root;
    }

    public ArrayList<XmlNode> getObjectsByName(String name){
        return root.getChildren(name);
    }

    public ArrayList<XmlNode> getRelationsByType(String type){
        ArrayList nodes = new ArrayList<XmlNode>();
```

```
		ArrayList<XmlNode> relations = root.getChildren("relation");

		for(XmlNode relation : relations) {
			ArrayList<XmlNode> tags = relation.getChildren("tag");
			for(XmlNode tag : tags) {
				if(tag.getParam("k").equals("type")) {
					if(tag.getParam("v").equals(type))
						nodes.add(relation);
					break;
				}
			}
		}

		return nodes;
	}

	public ArrayList<XmlNode> getMembersByRole(XmlNode relation, String role){
		ArrayList nodes = new ArrayList<XmlNode>();
		ArrayList<XmlNode> members = relation.getChildren("member");

		for(XmlNode member : members) {
			if(member.getParam("role").equals(role))
				nodes.add(member);
		}

		return nodes;
	}

	public ArrayList<XmlNode> getWaysByTagKey(String key){
		ArrayList nodes = new ArrayList<XmlNode>();
		ArrayList<XmlNode> children = root.getAllChildren();

		for(XmlNode node : children) {
			if(node.getName().equals("way")) {
				ArrayList<XmlNode> tags = node.getChildren("tag");

				for(XmlNode tag : tags) {
					if(tag.getParam("k").equals(key)){
						nodes.add(node);
					}
				}
			}
		}
		return nodes;
	}

	public XmlNode getObjectById(String id) {
		ArrayList<XmlNode> nodes = root.getAllChildren();
		for(XmlNode current : nodes) {
			if(current.getParam("id") == null) {
				continue;
			}
			if(current.getParam("id").equals(id)) {
				return current;
```

```
                }
            }
            return null;
    }

    public XmlNode getRoot() {
            return root;
    }

}
```

Для файлу XmlNode.java

```
package org.minemapcreator.minemapcreator.xmlparser;

import java.util.ArrayList;
import java.util.HashMap;
import org.minemapcreator.minemapcreator.mapgenerator.Progress;
import org.minemapcreator.minemapcreator.mapgenerator.ProgressWithComment;

public class XmlNode {

    private enum State {
        UNDEFINED, READ, READ_PARAMS, READ_NAME, BEGIN, END, FIELD_END, END_START
    }

    private ArrayList<XmlNode> children;
    private HashMap<String, String> params;

    private String name = "";
    private String content = "";

    public XmlNode() {
        children = new ArrayList<XmlNode>();
        params = new HashMap<String, String>();
    }

    public int parseFromText(StringBuilder text, Progress lambda) throws XmlParseException {
        countTest = 0;
        return parseFromText(text, lambda, text.length());
    }

    private static int countTest = 0;

    private static void countInc(int allLength, Progress lambda){
        countTest++;
        lambda.set((int) ((float) countTest / (float) allLength * 100.0));
    }

    private int parseFromText(StringBuilder text, Progress lambda, int allLength) throws XmlParseException {

        State state = State.UNDEFINED;
```

```
for (int i = 0; i < text.length(); i++, countInc(allLength, lambda)) {

   if (state == State.UNDEFINED) {
         if (text.charAt(i) == ' ' || text.charAt(i) == '\n' || text.charAt(i) == '\t') {
            continue;
         } else {
            state = State.BEGIN;
         }
   }

   if (state == State.BEGIN) {
         if (text.charAt(i) != ' ' && text.charAt(i) != '\n' && text.charAt(i) != '\t') {
            if (text.charAt(i) == '<') {
                  state = State.READ_NAME;
                  continue;
            } else {
                  throw new XmlParseException();
            }
         }
         continue;
   }

   if (state == State.READ_NAME) {
         if (text.charAt(i) == ' ' || text.charAt(i) == '\n' || text.charAt(i) == '\t') {
            state = State.READ_PARAMS;
            continue;
         }
         if (text.charAt(i) == '>') {
            state = State.READ_PARAMS;
         } else {
            if (text.charAt(i) == '/') {
                  state = State.READ_PARAMS;
            } else {
                  name += text.charAt(i);
                  continue;
            }
         }
   }

   if (state == State.READ_PARAMS) {
         if (text.charAt(i) == ' ' || text.charAt(i) == '\n'
                  || text.charAt(i) == '\t' || text.charAt(i) == '"') {
            continue;
         }
         if (text.charAt(i) == '>') {
            state = State.READ;
            continue;
         }
         if (text.charAt(i) == '/') {
            state = State.END;
            i++;
            countInc(allLength, lambda);
            continue;
```

```
                }
                int j = i;

                for (boolean f = false;; i++, countInc(allLength, lambda)) {
                  if (text.charAt(i) == '"' && !f) {
                        f = true;
                        continue;
                  }
                  if (text.charAt(i) == '"' && f) {
                        break;
                  }
                }

                String keyAndValue = text.substring(j, i);
                String splited[] = keyAndValue.split("=\"");
                if (splited.length != 1) {
                  params.put(splited[0], splited[1]);
                } else {
                  params.put(splited[0], "");
                }
          }

          if (state == State.READ) {
                if (text.charAt(i) == ' ' || text.charAt(i) == '\n'
                        || text.charAt(i) == '\t') {
                  continue;
                }
                for (; i < text.length(); i++, countInc(allLength, lambda)) {
                  if (text.charAt(i) == '<') {
                        if (text.charAt(i + 1) == '/') {
                          state = State.END_START;
                        } else {
                          XmlNode child = new XmlNode();
                          int increase = child.parseFromText(new StringBuilder(text.substring(i,
text.length())), lambda, allLength);
                          i += increase;

                          children.add(child);
                        }
                        break;
                  } else {
                        content += text.charAt(i);
                  }
                }
          }

          if (state == State.END_START) {
                for (;; i++, countInc(allLength, lambda)) {
                  if (text.charAt(i) == '>') {
                        state = State.END;
                        break;
                  }
                }
                continue;
```

```
            }

            if (state == State.END) {
                return i;
            }

        }

        return text.length();
    }

    public ArrayList<XmlNode> getChildren(String name) {
        ArrayList array = new ArrayList<XmlNode>();
        for (XmlNode child : children) {
            if (child.getName().equals(name)) {
                array.add(child);
            }
        }
        return array;
    }

    public ArrayList<XmlNode> getAllChildren() {
        return children;
    }

    public HashMap<String, String> getParams() {
        return params;
    }

    public String getParam(String key) {
        return params.get(key);
    }

    public String getName() {
        return name;
    }

    public String getContent() {
        return content;
    }


}
```

Для файлу XmlParseException.java

```
package org.minemapcreator.minemapcreator.xmlparser;

public class XmlParseException extends Exception{

}
```