МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет «Запорізька політехніка»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра програмних засобів\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(найменування кафедри)

**КУРСОВИЙ ПРОЕКТ**

з дисципліни «Об’єктно-орієнтоване програмування»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(назва дисципліни)

на тему:\_\_\_\_ «Генерація двомірного рівня для гри за параметрами»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студенту 1 курсу групи \_\_\_\_\_\_\_\_\_ КНТ-129сп\_\_\_\_\_\_\_\_

спеціальності Інженерія програмного забеспечення

напряму підготовки: Інженерія програмного забезпе -

чення\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Криви С. С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник Доцент к.н.т., Каплієнко Т. І.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала

Кількість балів: \_\_\_\_\_\_\_ Оцінка: ECTS

Члени комісії Каплієнко Т. І.

(підпис) (прізвище та ініціали)

Миронова Н. О.

(підпис) (прізвище та ініціали)

Табунщик Г. В.

(підпис) (прізвище та ініціали)

м. Запоріжжя

2020 рік

**ЗАВДАННЯ  
на курсовий проект студентів**

Криви Семен Сергійович

1. Тема проекту: Генерація двомірного рінвя для гри за параметрами

2. Термін здачі студентами закінченого проекту: 24 квітня 2020

3. Вихідні дані до проекту: Згенерувати ігровий рівень

дані: розмір рівня, кількість кімнат;

вихідні дані: згенерований рівень.

4. Зміст розрахунково-пояснювальної записки:

1. Аналіз предметної області;

2. Аналіз програмних засобів;

3. Основні рішення з реалізації компонентів системи;

4. Керівництво програміста;

5. Керівництво корисутвача;

Висновки

Додаток А. Текст програми

Додаток Б. Інтерфейс програми

Додаток В. Слайди презентації

5. Дата видачі завдання:

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів курсового  проекту (роботи) | Термін виконання етапів  проекту (роботи) | Примітка |
| 1. | Аналіз індивідуального завдання. | 1-2 тиждень |  |
| 2. | Аналіз програмних засобів, що будуть використовуватись в роботі. | 3-4 тиждень |  |
| 3. | Аналіз структур даних, що необхідно використати в курсової роботі. | 4-5 тиждень |  |
| 4. | Вивчення можливостей програмної реалізації структур даних та інтерфейсу користувача. | 5-9 тиждень |  |
| 5. | Аналіз вимог до апаратних засобів | 9 тиждень |  |
| 6. | Розробка програмного забезпечення | 9-15 тиждень |  |
| 7. | Оформлення, відповідних пунктів пояснювальної записки. | 10-16 тиждень | Розділи 1-5 ПЗ |
| 8. | Захист курсової роботи. | 17 тиждень |  |

Студент Крива С. С.

Керівник Каплієнко Т. І.

«\_\_\_\_\_\_\_» 2019 р.

реферат

Мета роботи – створення програмного забезпечення для генерації ігрових рівнів.

Проведено аналіз предметної області, досліджено аналогічне програмне забезпечення, а також існуючі методи та програмні засоби для вирішення завдання.

Для реалізації програмного продукту використовувалася мова програмування С# та середа розробки Unity.

Здійснено опис прийнятих рішень, реалізованих класів, наведено опис полів та методів реалізованих класів. Також у роботі були використані масиви для збереження рівнів.

У даній роботі було проведено дослідження процесу процедурної генерації у реаліях сьогоденних відеоігор, розглянуто особливості мови програмування С# у середовищі розробки Unity 2019. За основу для розробки графічного інтерфейсу для користувача була узята мова розмітки сторінки XAML, що використовувалася на базі графічної системи WPF. Потреби у збереженні, виведенні, конвертації та роботі з даними реалізовані у роботі із встроеною в Unity функцією Resources.

C#, GAME, PROCEDURAL LEVEL GENERATION, RESOURCES, UNITY

зміст

[Календарний план 2](#_Toc27629364)

[реферат 3](#_Toc27629365)

[зміст 4](#_Toc27629366)

[перелік умовних позначень, символів, одиниць, скорочень і термінів 6](#_Toc27629367)

[вступ 7](#_Toc27629368)

[1 Аналіз предметної області 8](#_Toc27629369)

[1.1 Основні вимоги 8](#_Toc27629370)

[1.2 Огляд існуючих методів вирішення завдання 8](#_Toc27629371)

[1.3 Огляд існуючих програм та сервісів для пошуку найкоротшого шляху 8](#_Toc27629372)

[1.3.1 Передмова 8](#_Toc27629373)

[1.3.2 Система «Procedural Level Generaor» 9](#_Toc27629374)

[1.3.3 Система «Level Generator 2D» 10](#_Toc27629375)

[1.3.4 Система «Tiled Level Genearator» 11](#_Toc27629376)

[2 Аналіз програмних засобів 12](#_Toc27629377)

[2.1 Огляд особливостей мови програмування 12](#_Toc27629378)

[2.2 Огляд особливостей обраного компілятору 14](#_Toc27629379)

[2.3 Висновки з розділу 17](#_Toc27629381)

[3 основні рішення з реалізації компонентів системи 18](#_Toc27629382)

[3.1 Основні рішення щодо розроблених класів 18](#_Toc27629383)

[3.1.1 Клас LevelGenerator 19](#_Toc27629384)

[3.1.2 Клас InputCheck 20](#_Toc27629385)

[3.1.3 Клас CameraControl 20](#_Toc27629386)

[3.1.4 Клас SaveLoad\_Level 21](#_Toc27629387)

[3.2 Основні розроблені алгоритми 22](#_Toc27629401)

[3.2.1 Аналіз головного алгоритму 22](#_Toc27629402)

[3.3 Основні рішення щодо розробки інтерфейсу 22](#_Toc27629408)

[3.3.1 Вибір платформи. 22](#_Toc27629409)

[3.3.2 Основні рішення з впорядкування елементів інтерфейсу 23](#_Toc27629410)

[3.3.3 Основні рішення з комунікації з користувачем 23](#_Toc27629411)

[4 керівництво програміста 24](#_Toc27629417)

[4.1 Призначення та умови застосування програми 24](#_Toc27629418)

[4.1.1 Призначення програми 24](#_Toc27629419)

[4.1.2 Функції програми 24](#_Toc27629420)

[4.1.3 Умови застосування програми 24](#_Toc27629421)

[4.2 Характеристика програми 25](#_Toc27629422)

[4.2.1 Структура програми 25](#_Toc27629423)

[4.2.2 Алгоритм роботи програми 25](#_Toc27629424)

[4.3 Звертання до програми 30](#_Toc27629426)

[4.4 Вхідні та вихідні дані 30](#_Toc27629427)

[4.4.1 Вхідні дані 30](#_Toc27629428)

[4.4.2 Вихідні дані 30](#_Toc27629429)

[5 керівництво користувача 32](#_Toc27629431)

[5.1 Призначення програми 32](#_Toc27629432)

[5.2 Умови виконання програми 32](#_Toc27629433)

[5.2.1 Апаратні вимоги програми 32](#_Toc27629434)

[5.3 Виконання програми 32](#_Toc27629436)

[5.3.1 Запуск програми 32](#_Toc27629437)

[5.3.2 Виконання роботи з програмою 33](#_Toc27629438)

[Висновки 36](#_Toc27629441)

[ПЕРЕЛІК ПОСИЛАНЬ 37](#_Toc27629442)

[Додаток А Діаграма класів 38](#_Toc27629443)

[Додаток Б Код Програми 39](#_Toc27629444)

перелік умовних позначень, символів, одиниць, скорочень і термінів

С# – об’єктно-орієнтована мова програмування.

Resources – дозволяє шукати та мати доступ до об'єктів та ассетів.

Unity – багатоплатформовий інструмент для розробки [дво-](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0) та [тривимірних](https://uk.wikipedia.org/wiki/3D_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0) додатків та ігор.

Масив – впорядкований набір фіксованої кількості однотипних елементів, що зберігаються в послідовно розташованих комірках оперативної пам'яті, мають порядковий номер і спільне ім'я, що надає користувач.

ООП - об’єктно-орієнтоване програмування.

вступ

Процедурна генерація рівнів – базова особливість багатьох ігор.

Отже, актуальність даної курсової роботи полягає у створенні програмного забезпечення, що має на меті генерацію рівня по параметрам.

Область використання такого застосунку включає в себе різні ігрові жанри, такі як: rogue-like, action, shooter та інші.

В якості характерних прикладів використання процедурної генерації називають: генерацію без участі людини підземіль в [пригодницьких](https://ru.wikipedia.org/wiki/Action-adventure) іграх (таких як [The Legend of Zelda](https://ru.wikipedia.org/wiki/The_Legend_of_Zelda)), отриманні нового світу при кожному запуску; системі, що створює нові типи озброєння у шутері; генерація повної, играбельної та [сбалансованої](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81) настільної гри; внутрішня процедура ігрового рушія, швидко заповнює ігровий світ рослинами; інструмент, який надає людині можливість створювати карти для стратегічної гри, та при запитах та заданих змінах перегенеровувати карту для її покращення, а також пропонує варіанти, які дозволяють зробити карту більш збалансованою та інтересною. У той самий час, простий редактор карт, [штучний інтеллект](https://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B8%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82) для настільної гри або інструмент інтеграції створенного контенту не відносяться до процедурної генерації [1].

1 Аналіз предметної області

В даному розділі розглянуті існуючі методи цвирішення завдання генерації ігрових рівнів та інструменти, що реалізують дану потребу.

1.1 Основні вимоги

Основні вимоги до генератора рівнів наступні:

* створення оточуючих стін;
* створення підлоги;
* створення кімнат.

1.2 Огляд існуючих методів вирішення завдання

Є ряд методів, які виконують генерацію рівня, який відповідає початковим умовам. Один з таких методів був створений у результаті виконання завдання.

Методи генерації рівнів характеризується величезною різноманітністю. Найбільш розповсюдженими є [2]:

* BSP дерева;
* алгоритм тунелювання;
* клітинні автомат

**1.3 Огляд існуючих програм та сервісів для пошуку найкоротшого шляху**

1.3.1 Передмова

Були розглянуті наступні інструменти для генерації рівнів:

* «Procedural Level Generator»;
* «Level Generator 2D»;
* «Tiled Level Generator».

В ході написання роботи був проведений аналіз інструментів для генерації рівнів, який проводився за настуними критеріями:

* виконання завдань генерації рівнів;
* необхідність закупівлі додаткових програм;
* вартість ліцензії;
* платформа використання;
* зручність і простота інтерфейсу.

1.3.2 Система «Procedural Level Generator»

Компанія-видавець: Unity Technologies.

Посилання:

<https://assetstore.unity.com/packages/tools/ai/procedural-level-generator-136626>

Короткий опис: бесплатний інструмент у Unity Asset Store, який дозволяє створювати рівні.

Переваги:

* дозволяє працювати з префабами;
* генеруєма секція може мати декілька тегів та може генерувати декілька видів секцій одночасно.

Недоліком є те, що дана програма не дозволяє детально налаштовувати декор секцій, крім випадків коли секція вже готовий префаб.

Робота програми продемонстрована на рисунку 1.1.

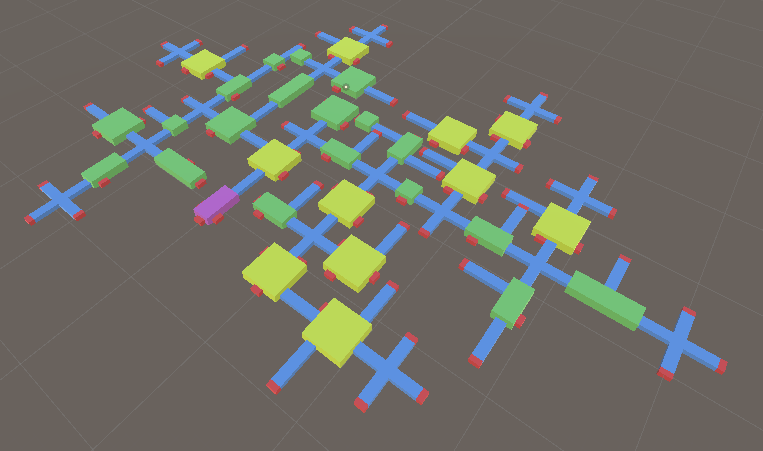


Рисунок 1.1 – Робота програми «Procedural Level Generator»

**1.3.3 Система** «Level Generator 2D»

Компанія-видавець: Unity Technologies.

Посилання: <https://assetstore.unity.com/packages/tools/level-design/level-generator-2d-68279#description>

Короткий опис: ще один інструмент з Unity Asset Store для творення двомірних рівнів.

Переваги:

* дозволяє детально налаштовувати встановлені двомірні секції;
* може генерувати рівні не залишаючи порожнього місця між приміщеннями.

Недоліком є те, що програма не має шаблонів для створення секцій.

Робота програми продемонстрована на рисунку 1.2.

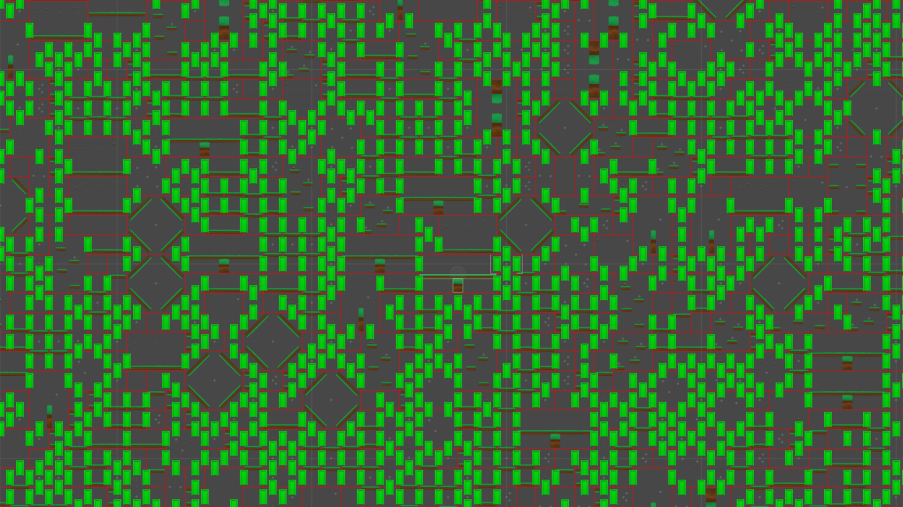


Рисунок 1.2 – Робота програми «Level Generator 2D»

1.3.4 Система «Tiled Level Generator»

Компанія-видавець: Epic Games.

Посилання:<https://www.unrealengine.com/marketplace/en-US/product/tiled-level-generator>.

Короткий опис: інструмент з Unreal Engine Marketplace для генерації рівнів.

Переваги:

* працює з двомірними та тривимірними рівнями;
* повністю сумісний з Blueprint ситемою у Unreal Engine.

Робота програми продемонстрована на рисунку 1.3.

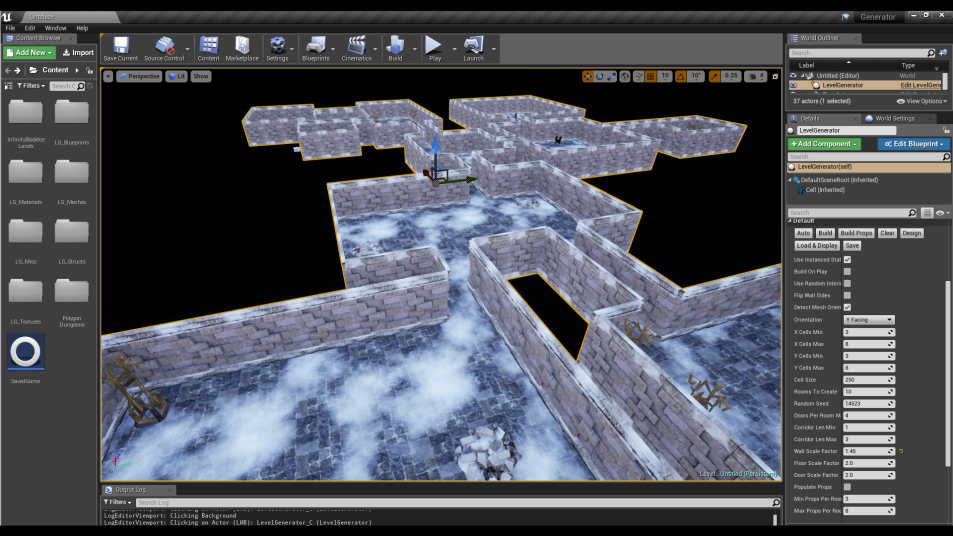


Рисунок 1.3 – Робота з програмою «Tiled Level Generator»

2 Аналіз програмних засобів

В даному розділі розглянуті основні особливості програмних засобів, за допомогою яких реалізовано курсовий проект.

2.1 Огляд особливостей мови програмування

Для написання програми "Генерація двомірного рівня для гри за параметрами" обрано таку мову програмування, як С#.

C# - це об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft. Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато що від своїх попередників – мов С++, Delphi, Модула і Smalltalk – С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем: так, C# не підтримує множинне спадкування класів (на відміну від C++) або виведення типів (на відміну Haskell).

Головною перевагою мови C# є можливість повторного використання створених компонентів. C# створювався паралельно з каркасом Framework Net і повною мірою враховує всі його можливості - як FCL, так й CLR; C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами; C# є потужною об'єктною мовою з можливостями спадкування й універсалізації; C# є спадкоємцем мов C/C++, зберігаючи кращі риси цих популярних мов програмування; завдяки каркасу Framework .Net, що стали надбудовою над операційною системою, програмісти C# одержують ті ж переваги роботи з віртуальною машиною, що й програмісти Java. Ефективність коду навіть підвищується, оскільки виконавче середовище CLR являє собою компілятор проміжної мови, у той час як віртуальна Java-машина є інтерпретатором байта-коду; потужна бібліотека каркасів підтримує зручність побудови різних типів додатків на C#, дозволяючи легко будувати Web-служби, інші види компонентів, досить просто зберігати й одержувати інформацію з бази даних й інших сховищ даних; реалізація, що сполучає побудову надійного й ефективного коду, є немаловажним чинником, що сприяє успіху C# .

Разом з тим C# передусім розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. Проте ця закономірність була порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET. CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування [3].

Отже, мова програмування С#, у якій поєднуються потужність і гнучкість універсальних мов програмування з високою ефективністю виконавчого коду й можливістю безпосереднього доступу до апаратних ресурсів комп’ютера – одна з найкращих мов програмування. Тому глибоке знання й практичне оволодіння інструментальними засобами мови С# є обов’язковим для фахівців із інформаційних технологій, систем автоматизованого керування й проектування, комп’ютерної інженерії, а також для всіх, хто бажає опанувати програмування.

2.2 Огляд особливостей обраного компілятору

Unity — багатоплатформовий інструмент для розробки [дво-](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0) та [тривимірних](https://uk.wikipedia.org/wiki/3D_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0) додатків та ігор, що працює на операційних системах [Windows](https://uk.wikipedia.org/wiki/Microsoft_Windows) і [OS X](https://uk.wikipedia.org/wiki/Mac_OS_X). Створені за допомогою Unity [застосунки](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) працюють під системами Windows, OS X, [Android](https://uk.wikipedia.org/wiki/Android), [Apple iOS](https://uk.wikipedia.org/wiki/Apple_iOS), [Linux](https://uk.wikipedia.org/wiki/Linux), а також на гральних консолях [Wii](https://uk.wikipedia.org/wiki/Wii), [PlayStation 3](https://uk.wikipedia.org/wiki/PlayStation_3) і [XBox 360](https://uk.wikipedia.org/w/index.php?title=XBox_360&action=edit&redlink=1).

Візуальний робочий процес виділяє даний інструмент з-посеред інших середовищ розробки ігор. Частина інструментів розробки ігор часто являють собою сукупність розрізнених частин, які потрібно контролювати, чи бібліотеку, для роботи з якою потрібно налаштовувати власне інтегроване середовище розробки, процес збирання проекту, тощо. Робочий процес в Unity прив'язаний до ретельно продуманого візуального редактору. Редактор надає можливість компонувати сцени майбутньої гри, пов'язуючи ігрові ресурси і код в інтерактивні об'єкти. Він дозволяє швидко і раціонально створювати професійні гри, забезпечуючи високу продуктивність праці розробників і надаючи в їх розпорядження вичерпний перелік сучасних технологій в області відеоігор. Редактор дозволяє редагувати об'єкти в редакторі і рухати елементи в сцені при запущеній грі. Unity надає можливість налаштовувати редактор за допомогою сценаріїв, що додають нові функціональні можливості і елементи меню до інтерфейсу.

Unity підтримує наступні платформи: Windows, OSX, Linux, Android, WebGL, iOS, Xbox One, Xbox 360, PS3, PS4, Windows Store, Lumin, Facebook, WebGL.

Швидкість виконання гри вимірюється в кількості кадрів за секунду (FPS – frame per second). На рисунку 2.1 наведено структуру кадру гри, реалізованої на Unity3D [4].

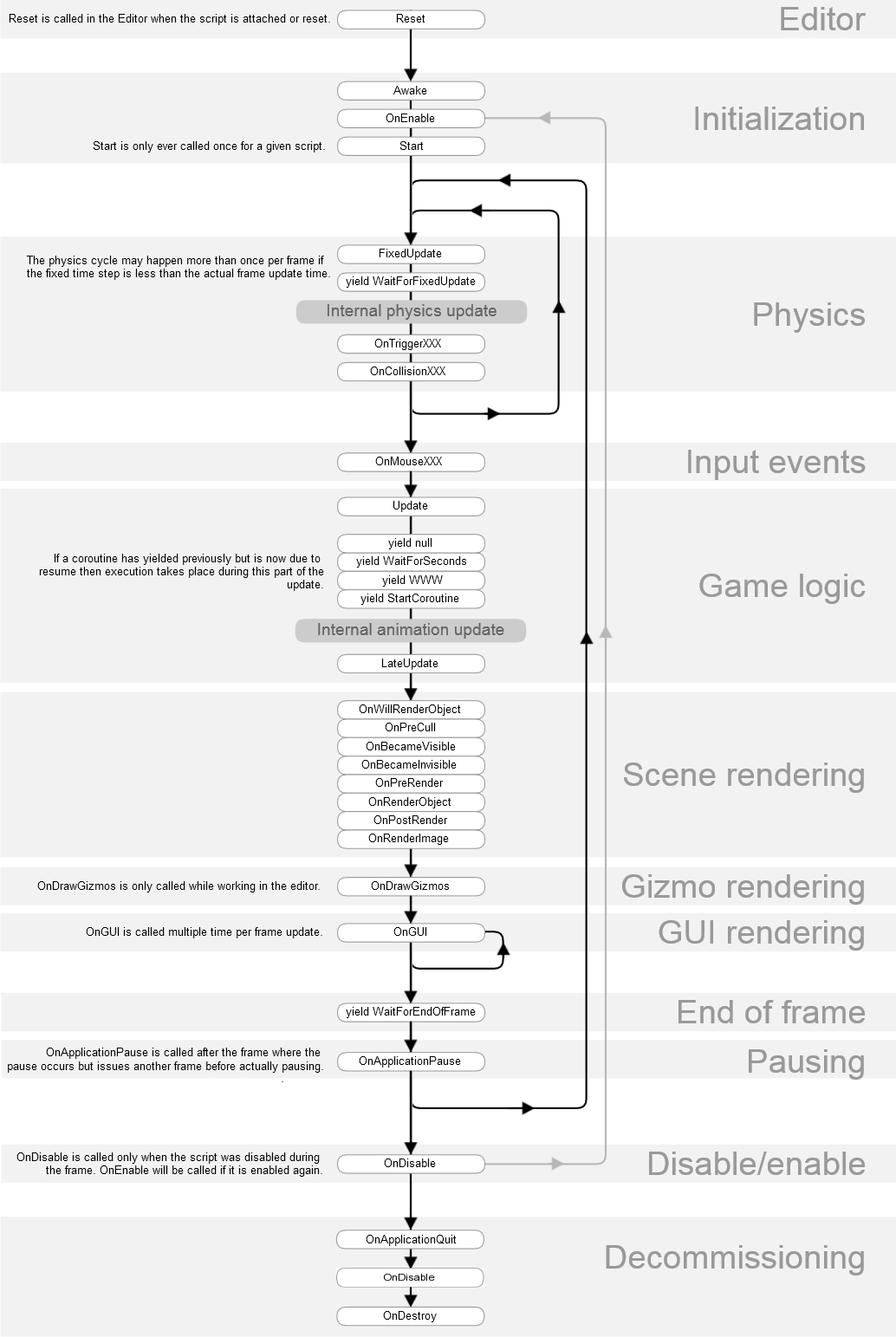


Рисунок 2.1 – Порядок подій [5]

У Unity3D існує цілий ряд подій, які виконуються в певному порядку.

Наступні функції викликаються при запуску сцени (один раз для кожного об'єкта в сцені):

1. Awake: функція завжди викликається перед будь-якими функціями Start, а також тільки після того, як створено екземпляр prefab (якщо GameObject неактивний під час запуску, функція не викликається, поки об’єкт не стане активним).

2. OnEnable: функція викликається тільки після того, як об'єкт активується.

3. OnLevelWasLoaded: подія настає при завантаженні сцени.

Перед першим викликом Update настає подія Start лише якщо об’єкт активовано.

OnApplicationPause настає між кадрами, параметр функції вказує на те, чи перебуває додаток в стані паузи.

Для побудови основної логіки гри використовуються функції Update:

1. FixedUpdate. Функція викликається фіксовану кількість раз на кадр незалежно від FPS (частіше чи рідше). Після FixedUpdate відбувається розрахунок фізики гри.

2. Update. Функція викликається одразу ж після обробки вхідних даних з частотою відповідною до FPS.

3. LateUpdate. Подія настає після завершення всіх Update. Зручно використовувати для зміни позиції та повороту камери від 3-го лиця.

Для прорисовки GUI викликається функція OnGUI (викликається декілька разів на кадр).

Функції yield використовуються для обробки підпрограм, оскільки додаток Unity3D виконується в одному потоці.

OnDisable настає при деактивації об’єкта.

OnApplicationQuit викликається при завершенні роботи додатку.

Гра в Unity складається з декількох сцен, кожна з який містить об’єкти типу GameObject. Об‘єкти на сцені складають ієрархічну структуру. GameObject – основний клас для всіх об’єктів на сцені. Кожний GameObject має декілька компонентів (Component), що приєднанні до нього. Обов’язковий компонентом є transform. Transform містить дані про абсолютну позицію об’єкта на сцені, позицію відносно батька, розмір об’єкта, абсолютний поворот об’єкта в сцені та поворот відносно батька. Компоненти реалізують логіку гри. Для створення власного сценарію необхідно унаслідуватися від класу MonoBehaviour, що є компонентом та може бути приєднаний до ігрового об’єкта на сцені.

Ігровий двигун Unity3D є універсальним двигуном для розробки ігрових додатків, міжплатформенним та легким у вивченні та використанні [5].

Також у Unity Asset Store існує огромна бібліотека ассетів та плагінів за допомогою яких можно значно прискорити процес розробки гри. Їх можно імпортувати та експортувати, додавати у гру цілі заготовки – рівні, ворогів, патерни поведінки AI і так далі. Багато ассетів доступні безплатно, інші пропонуються за невелику сумму.

2.3 Висновки з розділу

У даному розділі було описано мову програмування та її особливості. Також було описано обрану програму для розробки.

3 основні рішення з реалізації компонентів системи

В даному розділі розглянуті основні рішення з розробки класів, основні розроблені алгоритми, рішення щодо розробки інтерфейсу користувача, рішення щодо збереження даних та використання бази даних.

3.1 Основні рішення щодо розроблених класів

Структура програми складається з 4 класів, усі з яких – користувацькі. Діаграми користувацьких класів знаходяться у додатку А.

Класову структуру програми зображено на рисунку 3.1.

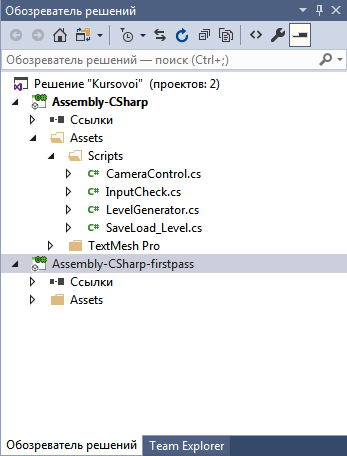


Рисунок 3.1 – Класова структура програми

3.1.1 Клас LevelGenerator

Клас LevelGenearator відповідає за генерацію рівня.

Таблиця 3.1 – Опис полів та методів класу LevelGenerator

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public void Generate() | Метод, який генерує рівень. |
| public void LevelSizeX\_ValueChange(string value) | Метод, що змінює значення розміру рівня X. |
| public void LevelSizeY\_ValueChange(string value) | Метод, що змінює значення розміру рівня Y. |
| public void NumberOfRooms\_ ValueChange(string value) | Метод, що змінює кількість кімнат які будуть згенеровані. |
| public void LoadLevel(string[] lines) | Метод, який обробляє дані при завантажені готового рівня. |
| saveData | Змінна, до якої заносяться данні про згенерований або завантажений. |
| ***private:*** | |
| cameraControl | Змінна, яка звертається до скрипту який керує камерою. |
| FloorTile | Змінна, яка зберігає тайл підлоги. |
| WallTile | Змінна, яка зберігає тайл стін. |
| FloorTiles | Змінна, яка зв'язана з тайлмепом подлоги. |
| WallTiles | Змінна, яка зв'язана з тайлмепом стін. |
| LevelSize | Змінна, яка зберігає у двомірному векторі розмір рівня. |

3.1.2 Клас InputCheck

Клас InputCheck перевіряє введення данних.

Таблиця 3.2 – Опис полів та методів класу InputCheck

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public void Generate() | Метод, який генерує рівень. |
| public void LevelSizeX\_InputCheck(string value) | Метод, що змінює значення розміру рівня X. |
| public void LevelSizeY\_ InputCheck (string value) | Метод, що змінює значення розміру рівня Y. |
| public void NumberOfRooms\_ ValueChange(string value) | Метод, що змінює кількість кімнат які будуть згенеровані. |
| ***private:*** | |
| cameraControl | Змінна, яка звертається до скрипту який керує камерою. |
| FloorTile | Змінна, яка зберігає тайл підлоги. |
| WallTile | Змінна, яка зберігає тайл стін. |
| FloorTiles | Змінна, яка зв'язана з тайлмепом подлоги. |
| WallTiles | Змінна, яка зв'язана з тайлмепом стін. |
| LevelSize | Змінна, яка зберігає у двомірному векторі розмір рівня. |

3.1.3 Клас CameraControl

Клас CameraControl керує камерою.

Таблиця 3.3 – Опис полів та методів класу CameraControl

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public void SetCamera(int sizeX, int sizeY) | Встановлює камеру у положення де вона може бачити весь рівень. |
| ***private:*** | |
| levelGenerator | Змінна, яка звертається до скрипту який генерує рівень для обробки завантаженних данних. |

3.1.4 Клас SaveLoad\_Level

Клас SaveLoad\_Level виконує збереження або завантаження рівня.

Таблиця 3.4 – Опис полів та методів класу SaveLoad\_Level

|  |  |
| --- | --- |
| Поля та методи класу | Опис |
| **1** | **2** |
| ***public:*** | |
| public void SaveLevel() | Метод, який зберігає рівень. |
| public void LoadLevel() | Метод, який завантажує рівень. |
| ***private:*** | |
| levelGenerator | Змінна, яка звертається о скрипту генерації рівня для обробки завантаженних даних. |

3.2 Основні розроблені алгоритми

3.2.1 Аналіз головного алгоритму

В ході написання курсової роботи був створений власний алгоритм генерації рівня.

Розроблений алгоритм видає згенерований рівень, але не гарантує, що він є найкращим серед можливих. Це позначає найголовнішу особливість алгоритму, а саме подальшу оптимізацію та можливе розширення виконуваних завдань.

Вхідними даними для алгоритму є:

* розмір рівня по координаті X;
* розмір рівня по координаті Y;
* кількість кімнат.

Усі значення мають бути коректними (за це відповдає перевірка вводу).

Увесь алгоритм можна поділити на 6 етапів:

1. Створення підлоги;
2. Створення оточуючих стін;
3. Визначення розміру кімнат та їх кількості у ряді.
4. Підраховування кількості рядків.
5. Створення кімнат.
6. Занесення данних про рівень до змінної для подальшого збереження.

Після виконання алгоритму маємо згенерований ріень.

3.3 Основні рішення щодо розробки інтерфейсу

3.3.1 Вибір платформи.

Рішенням щодо розробки інтрерфейсу було обрати встроєні можливості Unity.

3.3.2 Основні рішення з впорядкування елементів інтерфейсу

Під час розробки інтерфейсу користувача було використано наступні елементі графічного інтерфейсу:

* кнопки;
* текстові поля;
* поля вводу.

3.3.3 Основні рішення з комунікації з користувачем

Для комунікації з користувачем було використано наступне діалогове вікно: вікно для вводу параметрів рівня.

4 керівництво програміста

В даному розділі розглянуті призначення, умови застосування, характеристика програми, звертання до програми, початкові та вихідні дані та представлені повідомлення.

4.1 Призначення та умови застосування програми

4.1.1 Призначення програми

Програма “Генерація двомірного рівня для гри за параметрами” призначена для створення рівня на основі даних користувача.

4.1.2 Функції програми

Функції, що виконує програма:

* генерує рівень по заданим параметрам;
* завантажує рівень із txt файлу;
* зберігає рівень до txt файлу.

4.1.3 Умови застосування програми

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* операційна система Windows 7, 8, 10;
* наявність дисплею;
* наявність мищі та клавіатури;
* до 600 МБ оперативної пам’яті за умови використання функції генерації рівня.

4.2 Характеристика програми

Програма виконана за допомогою мови програмування високого рівня С# в середовищі розробки Unity 2019.

4.2.1 Структура програми

Програма складається з 4 класів. Ієрархія представлена на рисунку 4.1.

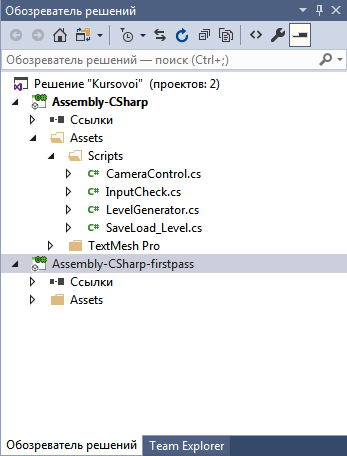


Рисунок 4.1 – Ієрархія класів проекту

4.2.2 Алгоритм роботи програми

Алгоритм починає з того, що бере розмір кімнати та заповнює тайлмеп підлоги відповідно до її розміру та тайлмеп стін за кімнатою товщиною в один тайл. Потім він створює змінну розміру кімнати, призначає їй максимальний можливий розмір та зменшує розмір до тих пір поки не отримає мінімальний можливий розмір при якому: помістяться усі кімнати, для теоретичного гравця залишиться коридор як мінімум в один тайл товщиною, яким він сможе дійти він нижньої до верхньої частини рівня. Потім він розставляє кімнати починаючи з правого нижнього кінця кімнати, чередуючи сторони з яких кімнати будуть створювати, поки усі кімнати не будуть створені.

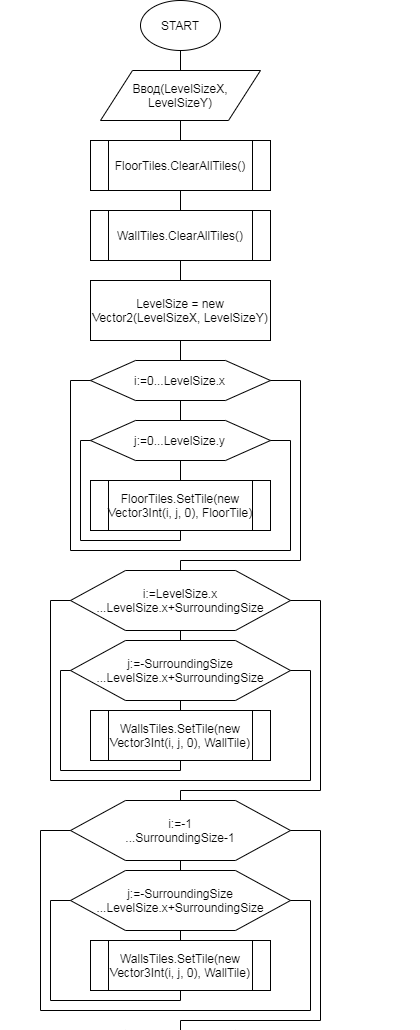


Рисунок 4.2 – Блок-схема алгоритму

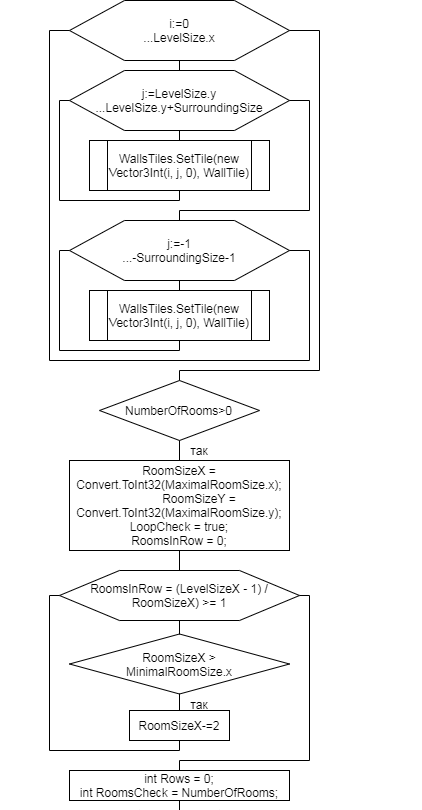


Рисунок 4.3 – Продовження блок-схеми алгоритму

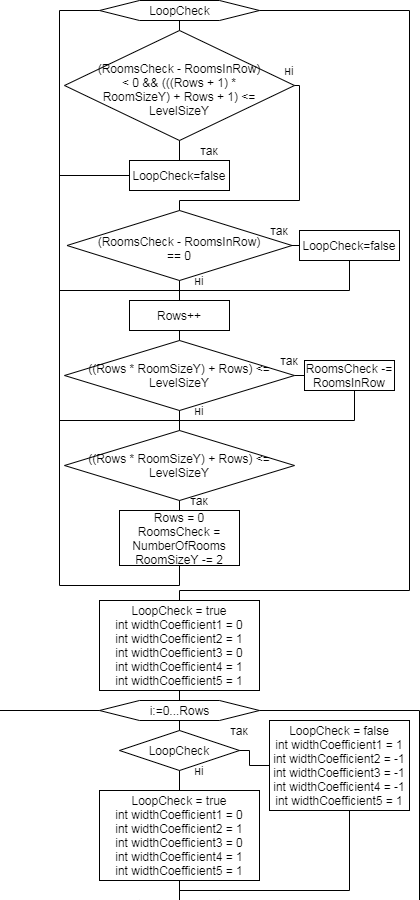


Рисунок 4.4 – Продовження блок-схеми алгоритму

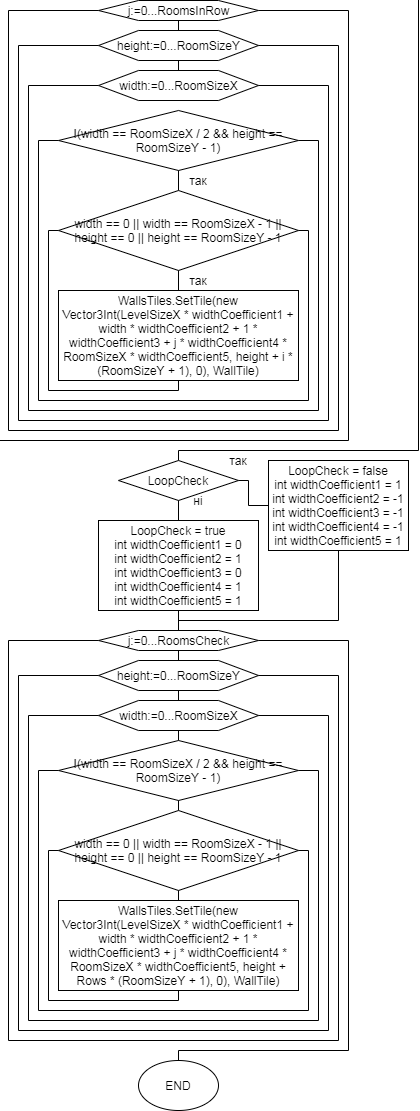


Рисунок 4.5 – Продовження блок-схеми алгоритму

4.3 Звертання до програми

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми. Потрібно упевнитися у наявності всіх файлів програми.

Звертання до програми передбачене одним способом: через виконуваний файл Kursovoi.exe. Звертання до програми через командний рядок не передбачене, адже програма розроблена саме для відображення рівня.

4.4 Вхідні та вихідні дані

4.4.1 Вхідні дані

При завантаженні рівня вхідні дані уявляють собою текстові файли, з розширенням .txt. У них за шаблоном повинні міститися дані рівня. При генерації рівня вхідні дані уявляють собою параметри, введені на головному екрані. Приклад такого файлу представлений на рисунку 4.2.

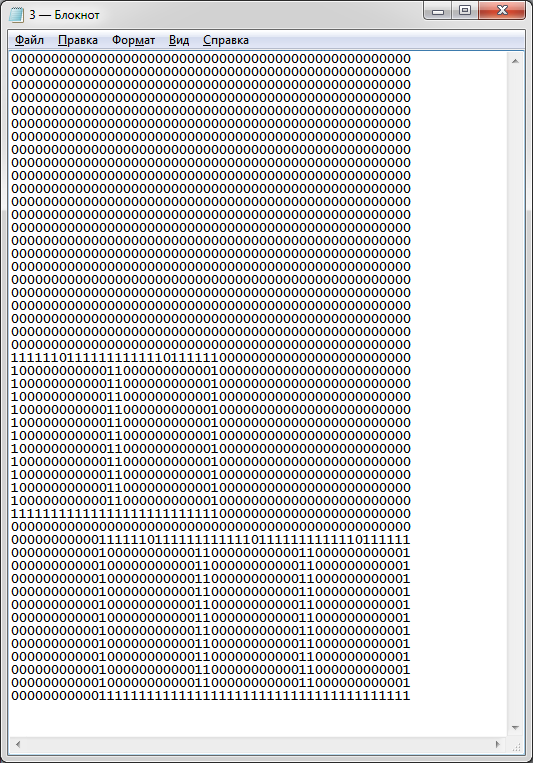


Рисунок 4.2 – Файл з рівнем

4.4.2 Вихідні дані

При збереженні рівня вихідні данні являють собою текстовий файл з розширенням .txt. У них за шаблоном повинні міститися дані рівня. При генерації рівня вихідні данні являють собою рівень підготовлений для збереження. Приклад такого файлу представлений на рисунку 4.2.

5 керівництво користувача

В даному розділі розглянуто призначення програми, умови її виконання, процес виконання програми та повідомлення для користувача.

5.1 Призначення програми

Програма “Kursovoi” призначена для генерації рівнів за параметрами, їх просмотру, збереження та завантаження.

5.2 Умови виконання програми

5.2.1 Апаратні вимоги програми

До комп’ютеру, на якому виконується програма, висуваються наступні вимоги:

* наявність дисплею;
* наявність клавіатури та миші;
* до 300 МБ оперативної пам’яті за умови використання функції створення розкладу.

5.3 Виконання програми

5.3.1 Запуск програми

Для запуску програми потрібно попередньо упевнитися в відповідності характеристик комп’ютера, на який встановлена програма, до системних вимог програми. Другим кроком потрібно упевнитися в наявності всіх бібліотек для роботи програми.

Звертання до програми передбачене одним способом: через виконуваний файл. Звертання до програми через командний рядок не передбачене, адже програма розроблена саме для відображення згенерованого рівня.

Після запуску виконуваного файлу Kursovoi.exe з’являється головна форма програми, що свідчить про початок роботи з програмою. Головна форма програми зображена на рисунку 5.1.

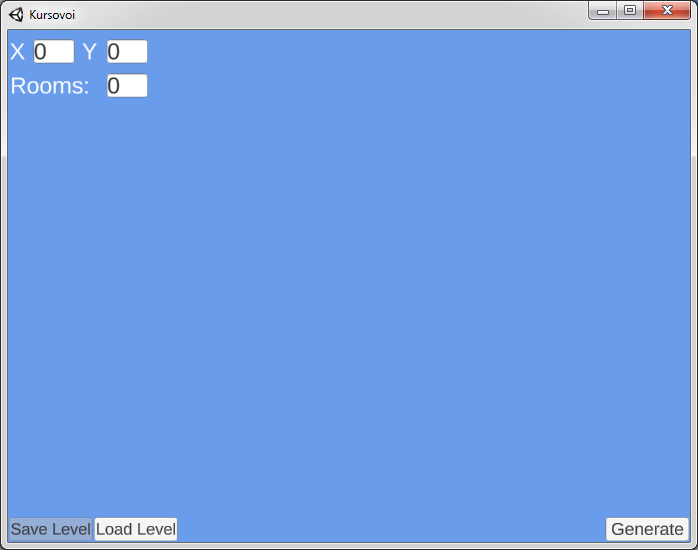


Рисунок 5.1 – Головна форма програми

5.3.2 Виконання роботи з програмою

Для генерації рівня користувач має ввести параметри та натиснути на кнопку Generate. Приклад згенерованого рівня зображено на рисунку 5.2.

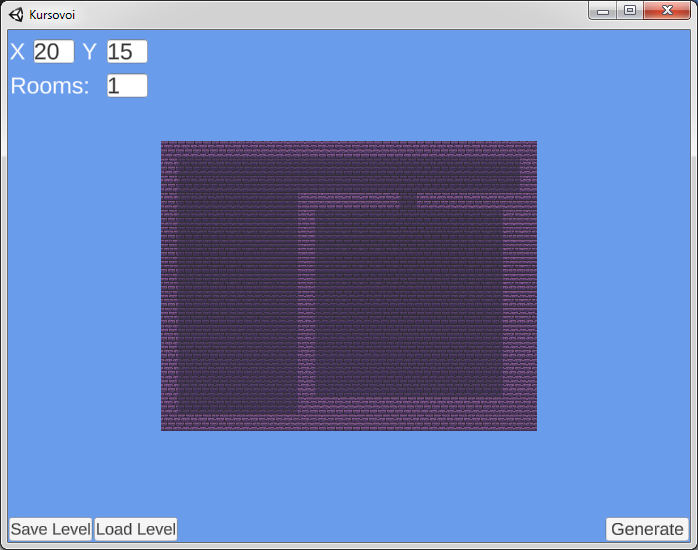


Рисунок 5.2 – Згенерований рівень

Для завантаження рівня користувач має натиснути кнопку LoadLevel та обрати потрібний файл. Приклад збереження рівня зображено на рисунку 5.3.

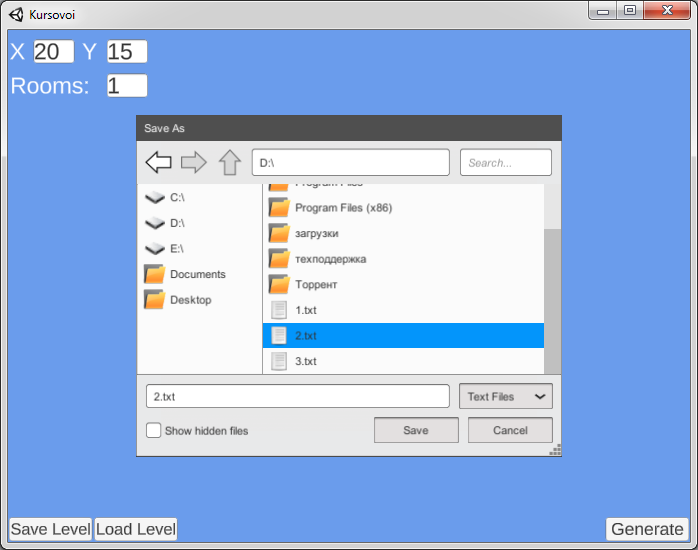


Рисунок 5.3 – Збереження рівня

Для збереження рівня користувач має натиснути SaveLevel та обрати шлях для збереження файлу. Приклад завантаження рівня зображено на рисунках 5.4 та 5.5.

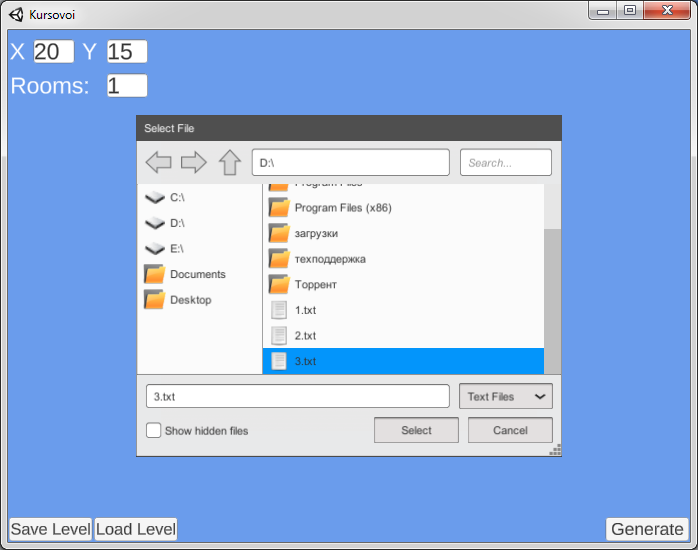


Рисунок 5.4 – Вибір файлу для завантаження

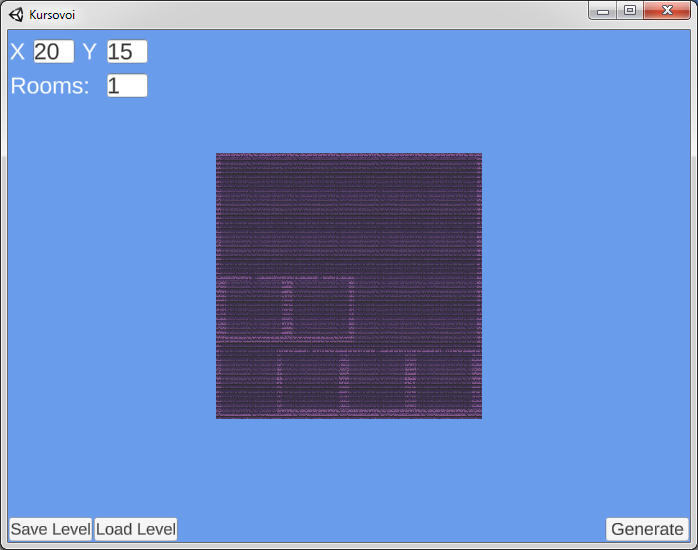


Рисунок 5.4 – Завантаженний рівень

Висновки

Під час виконання курсового проекту було розроблено програмне забезпечення для генерації двомірного рінвя за параметрами.

Було виконано огляд існуючих методів вирішення завдання та аналогів, проаналізовано переваги та недоліки і розроблено функціонал власного програмного продукту.

Для програми було реалізовано інструментарій для роботи з файловим вводом та вивидом інформації у Unity, що є звичним для цільової аудиторії користувачів та значно облегшує роботу з продуктом.

Було розроблено алгоритм генерації рівня, основними задачами якого є:

* створення оточуючих стін;
* створення підлоги;
* створення кімнат.

Було розроблено максимально простий та зрозумілий інтерфейс, у якому не має зайвих функцій.

Одною з ключових переваг розробленого продукту було закладення засад для подальшого розвитку та модернізації.

ПЕРЕЛІК ПОСИЛАНЬ

1) «Процедурна генерація» [Електрон. ресурс]. – <https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F>

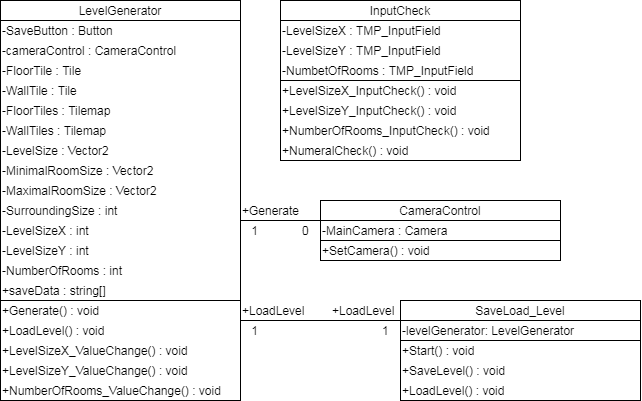
2) «Процедурна генерація підземіль» [Електрон. ресурс]. – <https://habr.com/ru/post/354826/>

3) «Переваги та недоліки мови програмування C#» [Електрон. ресурс]. – http://www.rusnauka.com/21\_NPN\_2017/Informatica/2\_226669.doc.htm

4) Хокинг Дж. Unity в действии. Мультиплатформенная разработка на C# / Пер. с англ. И. Рузмайкиной. — СПб.: Питер, 2016. — 336 с.: ил.

5) «Порядок виконання подій» [Електрон. ресурс]. – <https://docs.unity3d.com/Manual/ExecutionOrder.html>

Додаток А Діаграма класів



Додаток Б Код Програми

**Класс LevelGeneration:**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.Tilemaps;

using UnityEngine.UI;

public class LevelGenerator : MonoBehaviour

{

//4x2 Camera view = 1.15x

[SerializeField]

Button SaveButton;

[SerializeField]

CameraControl cameraControl;

[Header("Tiles")]

[SerializeField]

Tile FloorTile;

[SerializeField]

Tile WallTile;

[SerializeField]

Tilemap FloorTiles;

[SerializeField]

Tilemap WallsTiles;

Vector2 LevelSize = Vector2.zero;

Vector2 MinimalRoomSize = new Vector2(5, 5);

Vector2 MaximalRoomSize = new Vector2(13, 13);

const int SurroundingSize = 1;

public void Generate()

{

#region Clean

FloorTiles.ClearAllTiles();

WallsTiles.ClearAllTiles();

#endregion Clean

LevelSize = new Vector2(LevelSizeX, LevelSizeY);

char[,] save = new char[LevelSizeY, LevelSizeX];

for (int i = 0; i < LevelSizeY; i++)

for (int j = 0; j < LevelSizeX; j++)

save[i, j]= '0';

#region CreatingFloor

for (int i = 0; i < LevelSize.x; i++)

for (int j = 0; j < LevelSize.y; j++)

FloorTiles.SetTile(new Vector3Int(i, j, 0), FloorTile);

#endregion CreatingFloor

#region CreatingSurroundingWalls

for (int i = Convert.ToInt32(LevelSize.x); i < LevelSize.x + SurroundingSize; i++)

for (int j = -SurroundingSize; j < LevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int i = -1; i > -SurroundingSize - 1; i--)

for (int j = -SurroundingSize; j < LevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int i = 0; i < LevelSize.x; i++)

{

for (int j = Convert.ToInt32(LevelSize.y); j < LevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int j = -1; j > -SurroundingSize - 1; j--)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

}

#endregion CreatingSurroundingWalls

#region CreateRooms

if (NumberOfRooms > 0)

{

int RoomSizeX = Convert.ToInt32(MaximalRoomSize.x);

int RoomSizeY = Convert.ToInt32(MaximalRoomSize.y);

bool LoopCheck = true;

int RoomsInRow = 0;

do

{

if ((RoomsInRow = (LevelSizeX - 1) / RoomSizeX) >= 1)

LoopCheck = false;

else

{

if (RoomSizeX > MinimalRoomSize.x)

RoomSizeX -= 2;

else return;

}

}

while (LoopCheck);

int Rows = 0;

int RoomsCheck = NumberOfRooms;

LoopCheck = true;

do

{

if ((RoomsCheck - RoomsInRow) < 0 && (((Rows + 1) \* RoomSizeY) + Rows + 1) <= LevelSizeY)

{

LoopCheck = false;

}

else if ((RoomsCheck - RoomsInRow) == 0)

{

LoopCheck = false;

}

else

{

Rows++;

if (((Rows \* RoomSizeY) + Rows) <= LevelSizeY)

{

RoomsCheck -= RoomsInRow;

}

else if (RoomSizeY > MinimalRoomSize.y)

{

Rows = 0;

RoomsCheck = NumberOfRooms;

RoomSizeY -= 2;

}

else return;

}

}

while (LoopCheck);

LoopCheck = true;

int widthCoefficient1 = 0;

int widthCoefficient2 = 1;

int widthCoefficient3 = 0;

int widthCoefficient4 = 1;

int widthCoefficient5 = 1;

for (int i = 0; i < Rows; i++)

{

if (LoopCheck)

{

widthCoefficient1 = 1;

widthCoefficient2 = -1;

widthCoefficient3 = -1;

widthCoefficient4 = -1;

widthCoefficient5 = 1;

LoopCheck = false;

}

else

{

widthCoefficient1 = 0;

widthCoefficient2 = 1;

widthCoefficient3 = 0;

widthCoefficient4 = 1;

widthCoefficient5 = 1;

LoopCheck = true;

}

for (int j = 0; j < RoomsInRow; j++)

for (int height = 0; height < RoomSizeY; height++)

for (int width = 0; width < RoomSizeX; width++)

if (!(width == RoomSizeX / 2 && height == RoomSizeY - 1))

{

if (width == 0 || width == RoomSizeX - 1 || height == 0 || height == RoomSizeY - 1)

{

WallsTiles.SetTile(new Vector3Int(LevelSizeX \* widthCoefficient1 + width \* widthCoefficient2 + 1 \* widthCoefficient3 + j \* widthCoefficient4 \* RoomSizeX \* widthCoefficient5, height + i \* (RoomSizeY + 1), 0), WallTile);

save[height + i \* (RoomSizeY + 1) ,LevelSizeX \* widthCoefficient1 + width \* widthCoefficient2 + 1 \* widthCoefficient3 + j \* widthCoefficient4 \* RoomSizeX \* widthCoefficient5] = '1';

}

}

}

if (LoopCheck)

{

widthCoefficient1 = 1;

widthCoefficient2 = -1;

widthCoefficient3 = -1;

widthCoefficient4 = -1;

widthCoefficient5 = 1;

}

else

{

widthCoefficient1 = 0;

widthCoefficient2 = 1;

widthCoefficient3 = 0;

widthCoefficient4 = 1;

widthCoefficient5 = 1;

}

for (int j = 0; j < RoomsCheck; j++)

for (int height = 0; height < RoomSizeY; height++)

for (int width = 0; width < RoomSizeX; width++)

if (!(width == RoomSizeX / 2 && height == RoomSizeY - 1))

{

if (width == 0 || width == RoomSizeX - 1 || height == 0 || height == RoomSizeY - 1)

{

WallsTiles.SetTile(new Vector3Int(LevelSizeX \* widthCoefficient1 + width \* widthCoefficient2 + 1 \* widthCoefficient3 + j \* widthCoefficient4 \* RoomSizeX \* widthCoefficient5, height + Rows \* (RoomSizeY + 1), 0), WallTile);

save[height + Rows \* (RoomSizeY + 1), LevelSizeX \* widthCoefficient1 + width \* widthCoefficient2 + 1 \* widthCoefficient3 + j \* widthCoefficient4 \* RoomSizeX \* widthCoefficient5] = '1';

}

}

}

#endregion CreateRooms

saveData = new string[LevelSizeY];

for (int i = 0; i < LevelSizeY; i++)

{

string line = "";

for (int j = 0; j < LevelSizeX; j++)

line += save[i, j];

saveData[LevelSizeY - i-1] = line;

}

cameraControl.SetCamera(LevelSizeX, LevelSizeY);

SaveButton.interactable = true;

}

public void LoadLevel(string[] lines)

{

Vector2 LoadLevelSize = new Vector2(lines[0].Length, lines.Length);

saveData = lines;

#region Clean

FloorTiles.ClearAllTiles();

WallsTiles.ClearAllTiles();

#endregion Clean

#region Load

for (int i = lines.Length - 1; i >= 0; i--)

{

char[] line = lines[i].ToCharArray();

for (int j = 0; j < line.Length; j++)

{

if (line[j] == '0')

{

FloorTiles.SetTile(new Vector3Int(j, lines.Length - i-1, 0), FloorTile);

}

else if (line[j] == '1')

{

WallsTiles.SetTile(new Vector3Int(j, lines.Length - i-1, 0), WallTile);

}

}

}

#endregion Load

#region CreatingSurroundingWalls

for (int i = Convert.ToInt32(LoadLevelSize.x); i < LoadLevelSize.x + SurroundingSize; i++)

for (int j = -SurroundingSize; j < LoadLevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int i = -1; i > -SurroundingSize - 1; i--)

for (int j = -SurroundingSize; j < LoadLevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int i = 0; i < LoadLevelSize.x; i++)

{

for (int j = Convert.ToInt32(LoadLevelSize.y); j < LoadLevelSize.y + SurroundingSize; j++)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

for (int j = -1; j > -SurroundingSize - 1; j--)

WallsTiles.SetTile(new Vector3Int(i, j, 0), WallTile);

}

#endregion CreatingSurroundingWalls

cameraControl.SetCamera(Convert.ToInt32(LoadLevelSize.x), Convert.ToInt32(LoadLevelSize.y));

SaveButton.interactable = true;

}

#region TestInputs

int LevelSizeX = 1;

int LevelSizeY = 1;

int NumberOfRooms = 0;

public string[] saveData;

public void LevelSizeX\_ValueChange(string value)

{

if(value!="" && value!="0")

LevelSizeX = Convert.ToInt32(value);

}

public void LevelSizeY\_ValueChange(string value)

{

if (value != "" && value != "0")

LevelSizeY = Convert.ToInt32(value);

}

public void NumberOfRooms\_ValueChange(string value)

{

if (value != "" && value != "0")

NumberOfRooms = Convert.ToInt32(value);

}

#endregion TestInputs

}

**Класс InputCheck:**

using System.Collections;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using UnityEngine;

using TMPro;

public class InputCheck : MonoBehaviour

{

[SerializeField]

TMP\_InputField LevelSizeX;

[SerializeField]

TMP\_InputField LevelSizeY;

[SerializeField]

TMP\_InputField NumbetOfRooms;

public void LevelSizeX\_InputCheck(string value)

{

if (value.Length >= 4)

{

LevelSizeX.text = value.Remove(value.Length - 1, 1);

return;

}

NumeralCheck(LevelSizeX);

}

public void LevelSizeY\_InputCheck(string value)

{

if (value.Length >= 4)

{

LevelSizeY.text = value.Remove(value.Length - 1, 1);

return;

}

NumeralCheck(LevelSizeY);

}

public void NumberOfRooms\_InputCheck(string value)

{

if (value.Length >= 4)

{

NumbetOfRooms.text = value.Remove(value.Length - 1, 1);

return;

}

NumeralCheck(NumbetOfRooms);

}

public void NumeralCheck(TMP\_InputField CheckField)

{

if (CheckField.text.Length != 0 && !Regex.IsMatch(CheckField.text[CheckField.text.Length - 1].ToString(), @"^[0-9]+$"))

CheckField.text = CheckField.text.Remove(CheckField.text.Length - 1, 1);

}

}

**Класс CameraControl:**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraControl : MonoBehaviour

{

[SerializeField]

Camera MainCamera;

public void SetCamera(int SizeX, int SizeY)

{

float CameraFactor = (SizeX / 2f) > SizeY ? SizeX / 4f \* 2f: SizeY/2f \* 2f;

Vector3 position = new Vector3(SizeX/2f,SizeY/2f,-10);

MainCamera.transform.position = position;

MainCamera.orthographicSize = CameraFactor;

}

}

**Класс SaveLoad\_Level:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Text;

using UnityEngine;

using SimpleFileBrowser;

public class SaveLoad\_Level : MonoBehaviour

{

[SerializeField]

LevelGenerator levelGenerator;

private void Start()

{

FileBrowser.SetFilters(true, new FileBrowser.Filter("Text Files", ".txt"));

FileBrowser.SetDefaultFilter(".txt");

}

public void SaveLevel()

{

string[] saveData = levelGenerator.saveData;

FileBrowser.ShowSaveDialog(onSuccess=>

{

string fileName = onSuccess;

byte[] newline = Encoding.ASCII.GetBytes(Environment.NewLine);

using (FileStream fs = File.Create(fileName, 1024, FileOptions.WriteThrough))

{

for (int i = 0; i < saveData.Length; i++)

{

Byte[] title = new UTF8Encoding(true).GetBytes(saveData[i]);

fs.Write(title, 0, title.Length);

if(i!=saveData.Length-1)

fs.Write(newline, 0, newline.Length);

}

}

}, null, false, "C:\\", "Save As", "Save");

}

public void LoadLevel()

{

FileBrowser.ShowLoadDialog((path) =>

{

string[] lines = File.ReadAllLines(path);

levelGenerator.LoadLevel(lines);

},() => { Debug.Log( "Canceled" ); },false, null, "Select File", "Select" );

}

}