МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	Кафедра
«Вычислительная техника»	«Системы автоматизированного проектирования»
(наименование)	(наименование)
Направление подготовки	02.03.03 Математическое обеспечение и АИС
	(код и наименование)
Профиль	Администрирование информационных систем
	(наименование)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему:

Верификация игровых уровней с помощью алгоритма Ли

Студент	_____	Ильинская Е.С.
	(подпись, дата)	(фамилия, инициалы)
Руководитель	_____	Эпп В.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	проф. каф. САПР	_____ Бождай А.С.
	(должность, место работы)	(подпись, дата) (фамилия, инициалы)

Работа допущена к защите

Заведующий кафедрой _____ _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от_____№____)

Секретарь ГЭК _____ _____
подпись) (фамилия, инициалы)

Пенза 2023

“Утверждаю”

_____/А.М. Бершадский/

Заведующий кафедрой САПР

“____” _____ 2023 г

.

З А Д А Н И Е

на выполнение бакалаврской работы

1. Студент гр. 19ВА1_____факультета __ВТ___ специальности __Математическое обеспечение и администрирование информационных систем_____
Ильинская Екатерина Сергеевна _____ _____
(фамилия, имя, отчество)

2. Руководитель проекта ____к.т.н., доцент кафедры САПР Эпп Виталина Викторовна

3. Время дипломного проектирования с _____ 2023 г. по _____ 2023 г.

4. Место преддипломной практики _____кафедра САПР ФВТ ПГУ_____ _____

5. Тема проекта: «Верификация игровых уровней с помощью алгоритма Ли»
Тема утверждена приказом ПГУ № _____ от “_____” _____ _____ г.

6. Техническое задание на бакалаврскую работу (назначение устройства, условия применения, внешние взаимодействия, специальные требования т.п.)
6.1. Назначение разработки: Разрабатываемая программа предназначена для генерации и верификации игровых уровней
6.2. Требования к функциональности системы: 1) игровой уровень должен создаваться случайным образом 2) Должна быть осуществлена проверка проходимости уровня с помощью алгоритма Ли
6.3. Требования к ПО: Поддержка Unity 2019.4.12f1
6.4 Требования к аппаратному обеспечению: Персональный компьютер с операционной системой Windows 7 SP1+, 8, 10, только 64-разрядные версии; макОС 10.12+. (Серверные версии Windows и OS X не тестировались.)

7. Объем и содержание основной части проекта

7.1. Содержание пояснительной записки (перечень вопросов, подлежащих разработке, обоснований, описаний)

7.1.1. Анализ предметной области и технического задания

7.1.2. Проектирование алгоритма

7.1.3. Разработка алгоритма

7.1.4. Тестирование алгоритма

8. Календарный график работ по выполнению проекта.

Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя, консультанта
1. Анализ ТЗ	10 %	7.2-25.2	
2. Проектирование	30 %	25.2-20.3	
3. Разработка ПО	35%	20.3-10.5	
4. Тестирование программы	15%	10.5-15.5	
5. Оформление ПЗ	10 %	15.5-30.5	
6. Нормоконтроль		31.05	

Дата выдачи задания “____” _____2023 г.

Руководитель дипломного проекта _ к.т.н., доцент кафедры САПР Эпп Виталина Викторовна _____/_____

Задание к исполнению принял “____” _____2023 г.

Студент Ильинская Екатерина Сергеевна____/_____

Бакалаврскую работу к защите допустить

Декан ФВТ _____/ Л.Р. Фионова .

Реферат

Пояснительная записка содержит страниц, рисунка, 7 таблиц, 10 использованных источников и 1 приложение.

ПРОГРАММА ГЕНЕРАЦИИ И ВЕРИФИКАЦИИ ИГРОВЫХ УРОВНЕЙ, АЛГОРТИМ ЛИ, UNITY, C#, 2D, КОМПЬЮТЕРНЫЕ ИГРЫ, ЛАБИРИНТ.

Объект разработки – программа верификации игровых уровней с помощью алгоритма Ли.

Цель разработки: разработать программу генерации и верификации игровых уровней.

Результаты разработки: случайно генерируемые игровые уровни, пройденные проверку алгоритмом и являющиеся проходимыми.

Средства разработки: VisualStudio 2022, Unity 2019.4.12f1 (64-bit)

Содержание

Введение 6

1. Анализ предметной области и технического задания 9

1.1 Сравнительный анализ аналогичных разработок 9

1.2 Анализ технического задания 13

1.3 Концепция алгоритма Ли 14

2. Проектирование алгоритма 18

3. Разработка 23

3.1 Создание проекта 23

3.2 Разработка алгоритма 31

4. Тестирование алгоритма 39

Заключение 49

Список используемых источников 50

Приложение 51

Введение

В настоящее время игры занимают важное место в жизни многих людей, и разработчики игр постоянно стараются улучшить качество их продуктов. Одним из ключевых аспектов игрового процесса является создание уровней, которые должны быть интересными и вызывающими у игроков желание продолжать играть. Однако, создание уровней — это сложный процесс, который требует много времени и труда. В этом контексте возникает необходимость верификации уровней, чтобы убедиться, что они соответствуют заданным требованиям и не содержат ошибок. В данной работе рассматривается использование алгоритма Ли для верификации игровых уровней, что позволяет автоматизировать процесс и повысить качество создаваемых уровней.

Целью данной работы является разработка программы для генерации и верификации игровых уровней с помощью алгоритма Ли.

Тема верификации игровых уровней с помощью алгоритма Ли является актуальной в свете растущей популярности игр, особенно в жанре платформеров и головоломок. Алгоритм Ли позволяет автоматически проверять уровни на наличие пути от начала до конца, а также на наличие недостижимых областей и других ошибок в дизайне уровня. Это значительно упрощает процесс верификации и повышает качество игрового контента. Кроме того, использование алгоритма Ли может помочь ускорить процесс создания уровней, так как автоматическая проверка может выявлять ошибки и предупреждать разработчиков заранее.

Задачей разработки является создание программы для случайной генерации игровых уровней, которые будут проверяться на проходимость, с помощью алгоритма Ли.

Алгоритм Ли — это алгоритм поиска пути в лабиринте, который использует волновую функцию. Он был разработан Ли Харви в 1961 году и с тех пор нашел широкое применение в робототехнике, компьютерных играх и других областях [2].

Также стоит отметить, что использование алгоритма Ли для верификации игровых уровней может существенно сократить время, затрачиваемое на создание уровней. Вместо того, чтобы ручным образом проверять каждый уровень на

наличие ошибок, мы можем автоматизировать этот процесс и получить результаты гораздо быстрее и точнее.

Существует несколько способов верификации игровых уровней: Автоматическая верификация — это процесс, при котором система автоматически проверяет игровой уровень на наличие ошибок и недочетов. Этот способ используется в большинстве игр и позволяет быстро и эффективно проверять уровни.

Ручная верификация — это процесс, при котором опытный игрок или модератор вручную проверяет игровой уровень на наличие ошибок и недочетов. Этот способ используется в случае, если автоматическая верификация не справилась с задачей или если уровень слишком сложный для автоматической проверки.

Групповая верификация — это процесс, при котором несколько опытных игроков или модераторов совместно проверяют игровой уровень на наличие ошибок и недочетов. Этот способ используется в случае, если уровень очень сложный или содержит множество элементов, которые нужно проверить.

Верификация через видео — это процесс, при котором игрок записывает видеоигру и отправляет его модераторам для проверки. Этот способ используется в случае, если уровень очень сложный или содержит элементы, которые нельзя проверить автоматически.

Каждый из этих способов имеет свои преимущества и недостатки, и выбор конкретного способа зависит от типа игры, уровня сложности и других факторов.

Верификация будет осуществляться на уровне типа «лабиринт» или «подземелье». В данной работе эти понятия предлагаю считать эквивалентными.

Алгоритм может быть использован для поиска пути в лабиринтах любой сложности и формы, и не требует больших вычислительных мощностей. Кроме того, алгоритм Ли может быть легко модифицирован для учета препятствий и других условий.

Создание игр на Unity является удобным и популярным выбором для многих разработчиков. Во-первых, Unity предоставляет широкий спектр инструментов и

ресурсов для создания игр, включая графический движок, физический движок, аудиосистему, инструменты моделирования и анимации, а также множество готовых компонентов и библиотек.

Во-вторых, Unity обладает простым и интуитивно понятным интерфейсом, что делает процесс создания игры более доступным для начинающих разработчиков.

В-третьих, Unity поддерживает множество платформ, включая PC, мобильные устройства, консоли и виртуальную реальность, что позволяет создавать игры для широкой аудитории.

Кроме того, Unity имеет большое сообщество разработчиков, которые делятся своими знаниями и опытом, а также предоставляют готовые решения и инструменты для ускорения процесса разработки.

В целом, использование Unity для создания игр является удобным и эффективным выбором, который позволяет разработчикам быстро и качественно создавать игровой контент для различных платформ.

1 Анализ предметной области и технического задания

Анализ предметной области показал, что создание игровых уровней является важным этапом разработки игр. Ошибки при создании уровней могут привести к снижению интереса игроков к игре и ухудшению ее продаж.

В качестве инструмента для верификации игровых уровней был выбран алгоритм Ли. Этот алгоритм является одним из наиболее эффективных алгоритмов для поиска кратчайшего пути в графе. Он может быть использован для проверки, что каждая точка на карте достижима из начальной точки, что все связи между точками корректны и что кратчайший путь между двумя точками на карте вычисляется правильно.

Создание игр на Unity является удобным и популярным выбором для многих разработчиков. Во-первых, Unity предоставляет широкий спектр инструментов и ресурсов для создания игр, включая графический движок, физический движок, аудиосистему, инструменты моделирования и анимации, а также множество готовых компонентов и библиотек.

Во-вторых, Unity обладает простым и интуитивно понятным интерфейсом, что делает процесс создания игры более доступным для начинающих разработчиков.

В-третьих, Unity поддерживает множество платформ, включая PC, мобильные устройства, консоли и виртуальную реальность, что позволяет создавать игры для широкой аудитории.

Кроме того, Unity имеет большое сообщество разработчиков, которые делятся своими знаниями и опытом, а также предоставляют готовые решения и инструменты для ускорения процесса разработки.

В целом, использование Unity для создания игр является удобным и эффективным выбором, который позволяет разработчикам быстро и качественно создавать игровой контент для различных платформ.

1.1 Сравнительный анализ аналогичных разработок

В ходе анализа ТЗ мной были рассмотрены несколько алгоритмов поиска пути в лабиринте.

1. Алгоритм Дейкстры [3] — это один из самых простых и популярных алгоритмов поиска пути в лабиринте. Он основан на принципе поиска кратчайшего пути от начальной точки до конечной. Алгоритм Дейкстры работает с положительными весами ребер и гарантирует нахождение кратчайшего пути.

Алгоритм Дейкстры также имеет несколько минусов:

– Требует большого объема памяти. Это может привести к проблемам производительности при работе с большими лабиринтами или в ограниченных условиях памяти.

– Неэффективен при работе с динамическими изменениями лабиринта. При динамических изменениях, алгоритм Дейкстры может потребовать пересчета всего пути, что может занять много времени.

– В лабиринтах требуется много времени, чтобы найти путь до конечной точки.

2. Алгоритм A* [4] — это эвристический алгоритм, который использует оценку расстояния от текущей точки до конечной, чтобы выбирать следующую точку для поиска. Он работает быстрее, чем алгоритм Дейкстры, и может находить оптимальный путь даже в случаях, когда есть препятствия на пути.

Несмотря на то, что алгоритм A* является одним из наиболее популярных и эффективных алгоритмов поиска пути, он также имеет несколько минусов:

– Требует большого объема памяти. Это может привести к проблемам производительности при работе с большими лабиринтами или в ограниченных условиях памяти.

– Неэффективен при работе с динамическими изменениями лабиринта. При динамических изменениях, то алгоритм A* может потребовать пересчета всего пути, что может занять много времени.

– Не подходит для работы с непрерывными пространствами. Алгоритм A* оптимизирован для работы с сетками узлов или сетками с ограниченным

числом возможных путей, что делает его неэффективным для непрерывных пространств, таких как трехмерные сцены или графы.

3. Лучевой алгоритм поиска пути в лабиринте [5], также известный как алгоритм луча, используется для нахождения кратчайшего пути от заданной начальной точки до конечной точки в лабиринте. Алгоритм обеспечивает проведение ломаной линии, соединяющую источник и приемник и обходящую препятствия.

Однако, у лучевого алгоритма так же есть ряд минусов:

– Не гарантирует нахождение оптимального пути в графах с препятствиями. Если в графе есть препятствия, то лучевой алгоритм может найти неоптимальный путь.

– Не учитывает динамические изменения графа. Если препятствия или другие элементы графа изменяются, то лучевой алгоритм может потребовать пересчета всего пути, что может занять много времени.

– Требует большого объема памяти для хранения информации о карте и пути. Это может привести к проблемам производительности при работе с большими картами или в ограниченных условиях памяти.

– Не подходит для работы с графами с циклами. Если в графе есть циклы, то лучевой алгоритм может зациклиться и не дать правильный результат.

4. Алгоритм Ли [2] — это алгоритм, который использует волновой алгоритм для поиска пути от начальной точки до конечной. Он гарантирует нахождение кратчайшего пути и может быть быстрее на подземельях с большими комнатами.

Из вышеперечисленных алгоритмов наиболее эффективными являются алгоритмы A* и Ли, так как они гарантируют нахождение кратчайшего пути и работают быстрее других алгоритмов. Однако выбор конкретного алгоритма зависит от конкретной задачи и требований к скорости и точности поиска пути. Ниже приведена таблица сравнения вышеописанных имитаторов по основным характеристикам и функционалу.

Сравнение будет проведено по следующим критериям:

1. Скорость работы - время, затрачиваемое на поиск пути в лабиринте.

2. Точность - способность находить оптимальный путь в лабиринте.

3. Ресурсоемкость - количество памяти и вычислительных ресурсов, необходимых для выполнения алгоритма.

4. Гибкость - способность алгоритма работать с различными типами лабиринтов и условиями.

5. Устойчивость - способность алгоритма работать при наличии ошибок в данных или препятствий в лабиринте.

Таблица 1–Аналоги алгоритма Ли поиска пути в лабиринте.

Критерии	**Алгоритм Ли**	Алгоритм Дейкстры	Алгоритм А*	Лучевой алгоритм
Скорость работы	-	-	+	+
Точность	+	+	+	-
Ресурсоемкость	-	+	-	+
Гибкость	+	-	+	-
Устойчивость	+	-	-	-

Как видно из таблицы, алгоритм Ли работает медленнее, чем А* или Лучевой алгоритм, зато он может быть более эффективным в поиске оптимального пути. Алгоритм Ли может быть настроен для работы с различными типами лабиринтов и условиями.

Алгоритм Ли имеет высокую точность, поскольку он учитывает все граничные условия и позволяет получить точный путь между начальной и конечной точками в лабиринте. Он может находить кратчайший путь, если такой существует, и в противном случае сообщать об этом

Алгоритм Ли достаточно гибкий и может быть адаптирован для различных типов лабиринтов и задач поиска пути. Он может работать с лабиринтами любой формы и размера, и может быть настроен для различных критериев поиска пути, таких как кратчайший путь или минимальное количество поворотов.

Алгоритм Ли устойчив к препятствиям и другим помехам в лабиринте, поскольку он учитывает все клетки лабиринта и не зависит от отдельных препятствий. Он может обходить препятствия, находить путь вокруг них и продолжать поиск пути. Однако, если в лабиринте слишком много препятствий, это может замедлить работу алгоритма.

В целом, алгоритм Ли является хорошим выбором для поиска пути в лабиринтах, особенно на небольших и средних размерах.

1.2 Анализ технического задания

Для реализации проекта была выбрана платформа Unity, которая позволяет создавать игры для различных платформ. Unity – межплатформенная среда разработки компьютерных игр [6]. Большинство необходимых средств в Unity бесплатна, а это является одним из главных плюсов.

Преимущества Unity:

- Unity бесплатна. Существуют и платные версии Unity, однако для студента или начинающего разработчика будет достаточно бесплатной версии этого программного продукта.
- Кроссплатформенная поддержка. Это означает, что с помощью Unity можно создать проект для любой платформы, в том числе Windows, Linux, iOS, macOS и т.д.
- Магазин ассетов. При необходимости, в Asset Store можно приобрести необходимые скрипты, модели, звуки и многое другое для создаваемых игр.
- Визуальный редактор. Визуальный стиль разработки позволяет экономить время, требуемое на разработку проекта.

Исходя из данных преимуществ, для реализации проекта была выбрана платформа Unity. Разработка проекта будет вестись на языке

программирования C#. Техническое задание на проект включает в себя следующие этапы:

1. Разработка алгоритма Ли для поиска кратчайшего пути в лабиринте.
2. Создание игровых уровней в Unity и представление их в виде подземелья.
3. Реализация верификации игровых уровней с помощью алгоритма Ли.
4. Создание пользовательского интерфейса для управления процессом генерации и верификации.
5. Тестирование и отладка проекта.

В результате выполнения проекта ожидается получить инструмент для автоматической верификации игровых уровней, что позволит сократить время, затрачиваемое на создание и тестирование уровней, и повысить качество создаваемых игр.

Анализ технического задания в данном дипломном проекте направлен на определение требований к создаваемому программному инструменту для верификации игровых уровней на платформе Unity с использованием Алгоритма Ли.

Основными элементами анализа технического задания являются функциональные и нефункциональные требования, описание возможностей и ограничений, а также определение технических характеристик создаваемого инструмента.

Задачей разработки является создание программы для случайной генерации игровых уровней, которые будут проверяться на проходимость, с помощью алгоритма Ли.

Описание возможностей и ограничений позволяет определить, какие функции и возможности могут быть реализованы в рамках создаваемого инструмента, а также какие ограничения могут быть наложены на его работу.

1.3 Концепция алгоритма Ли

Алгоритм Ли, также известный как алгоритм распространения волны, используется для поиска кратчайшего пути в лабиринте. Он начинает работу с заданной начальной точки и распространяет волну от нее до тех пор, пока не достигнет конечной точки. В процессе распространения волны, каждая клетка лабиринта помечается расстоянием от начальной точки, что позволяет найти кратчайший путь.

Алгоритм Ли работает следующим образом:

1. Задается начальная и конечная точки в лабиринте.

2. Начальная точка помечается единицей, а все остальные клетки лабиринта помечаются нулем или NULL.

3. На первой итерации алгоритма, клетки, соседствующие с начальной точкой, помечаются двойкой.

4. На следующей итерации, клетки, соседствующие с уже помеченными клетками единицей, помечаются тройкой.

5. Процесс продолжается до тех пор, пока волна не достигнет конечной точки. Каждая клетка помечается расстоянием от начальной точки.

6. Когда волна достигнет конечной точки, алгоритм останавливается и кратчайший путь находится обратным ходом от конечной точки к начальной, следуя по клеткам с наименьшим расстоянием.

На выходе получаем значение равное длине пути, а также маршрут от источника к приемнику.

Для устранения неопределенности проведения трассы в случае, если несколько соседних площадок имеют одинаковый вес, используются путевые координаты. Эти координаты указывают предпочтительное направление трассы [7].

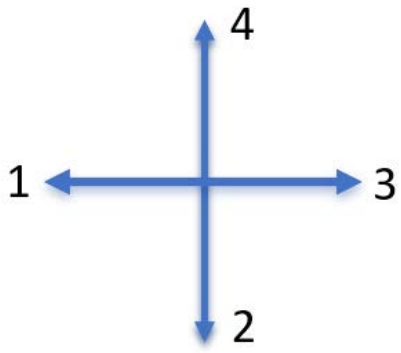

Рисунок 1 - Путевые координаты

Пример работы алгоритма Ли (рис. 2-4):

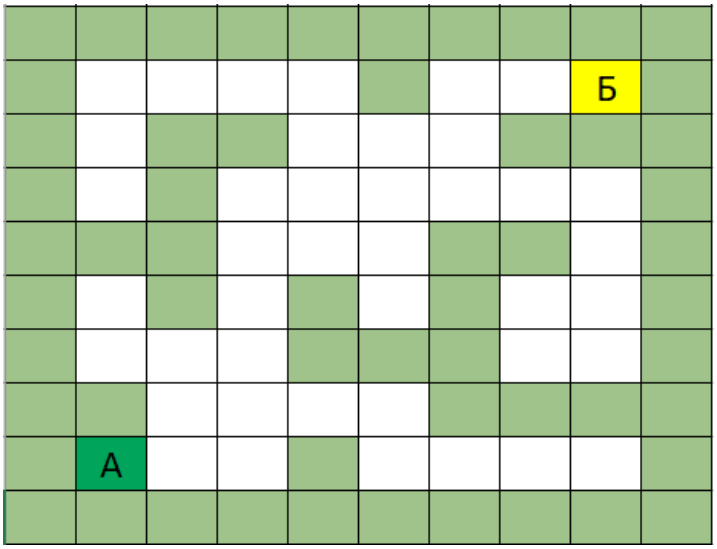

Рисунок 2 – Лабиринт

На рисунке 2 представлено поле, которое подаётся на вход. Белым обозначены проходимые(свободные) клетки, зеленым – непроходимые(занятые).

Точка А является источником, т.е. начальной точкой, от которой будет распространяться волна. Точка Б – приёмник, т.е. конечная точка, к которой следует проложить маршрут.

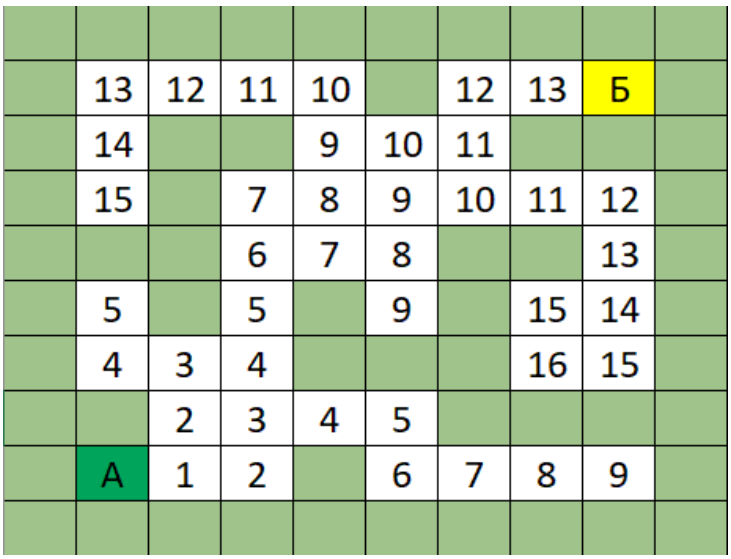

Рисунок 3 - Распространение волны

На рисунке 3 показан результат распространения волны по полю.

Рисунок 4 - Поиск пути

Когда волна достигнет конечной точки, алгоритм останавливается и кратчайший путь находится обратным ходом от конечной точки к начальной, следуя по клеткам с наименьшим расстоянием.

2 Проектирование алгоритма

В данной работе алгоритм Ли будет использоваться для верификации созданных игровых уровней. С помощью данного алгоритма будет осуществлена проверка является ли уровень проходимым или нет

Для выполнения алгоритма нам необходимо знать:

- Размер карты

- Какие клетки являются проходимыми

- Какие клетки желательно избегать

После того как определены все необходимые данные, можно начинать выполнение алгоритма.

В первую очередь необходимо создать само игровое поле.

На данном этапе создается несколько комнат случайного размера. Далее между комнатами формируются проходы, соединяющие их между собой.

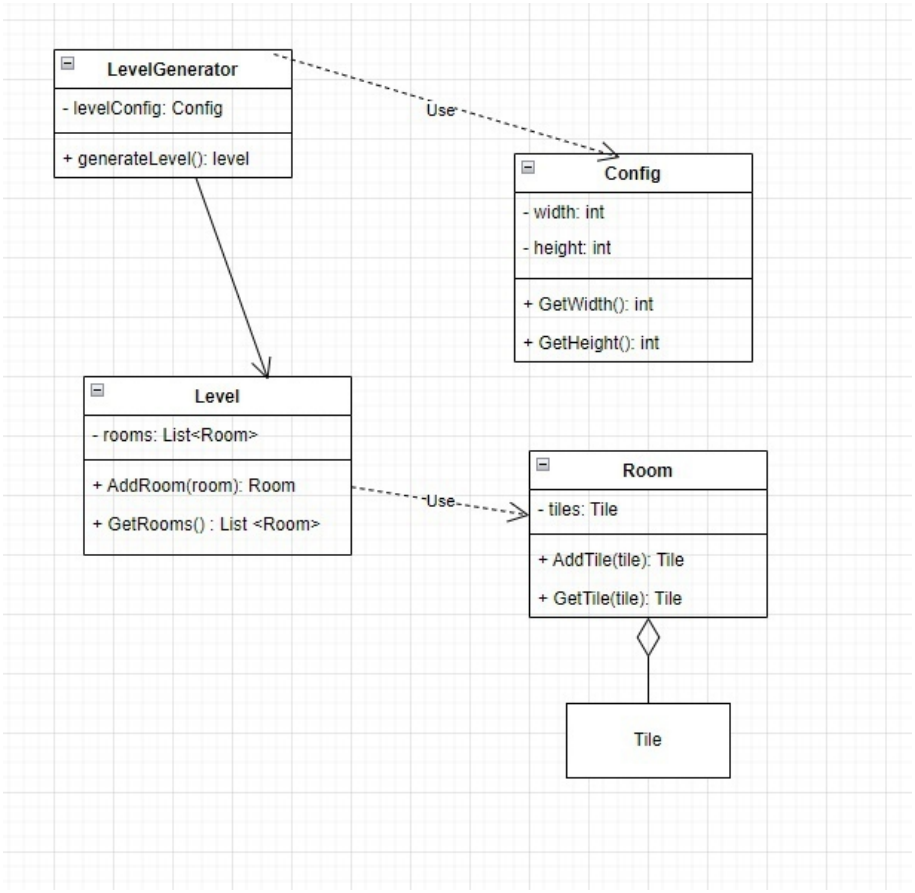

Рисунок 5 – Условная генерация игрового уровня

Далее на получившемся игровом поле создаются дополнительные объекты-препятствия, которые игроку желательно избегать.

На рисунке 6 показан алгоритм создания игровых уровней.

Рисунок 6 – Алгоритм создания уровней

В процессе формирования пути допустимы два сценария:

1. Сформирован безопасный маршрут.
2. Сформирован маршрут, при котором игрок столкнется с некоторым количеством препятствий, вследствие которого здоровье игрока может уменьшится.

Оба этих варианта допустимы, однако, при втором сценарии следует сообщить о том, что уровень необходимо доработать таким образом, чтобы была возможность сформировать безопасный маршрут.

Для тестирования генерации 2D подземелий с комнатами и препятствиями будет использоваться алгоритм Ли. Этот алгоритм позволяет найти кратчайший путь от начальной точки до конечной точки в лабиринте.

Для тестирования генерации подземелий с комнатами и препятствиями будет создано игровое поле. Затем нужно выбрать начальную и конечную точки в лабиринте и запустить волновой алгоритм для поиска кратчайшего пути между ними. Алгоритм Ли должен находить кратчайший путь между начальной и конечной точками без прохождения через стены или препятствия. Если же волновой алгоритм не может найти кратчайший путь, то это может указывать на ошибки в генерации лабиринта.

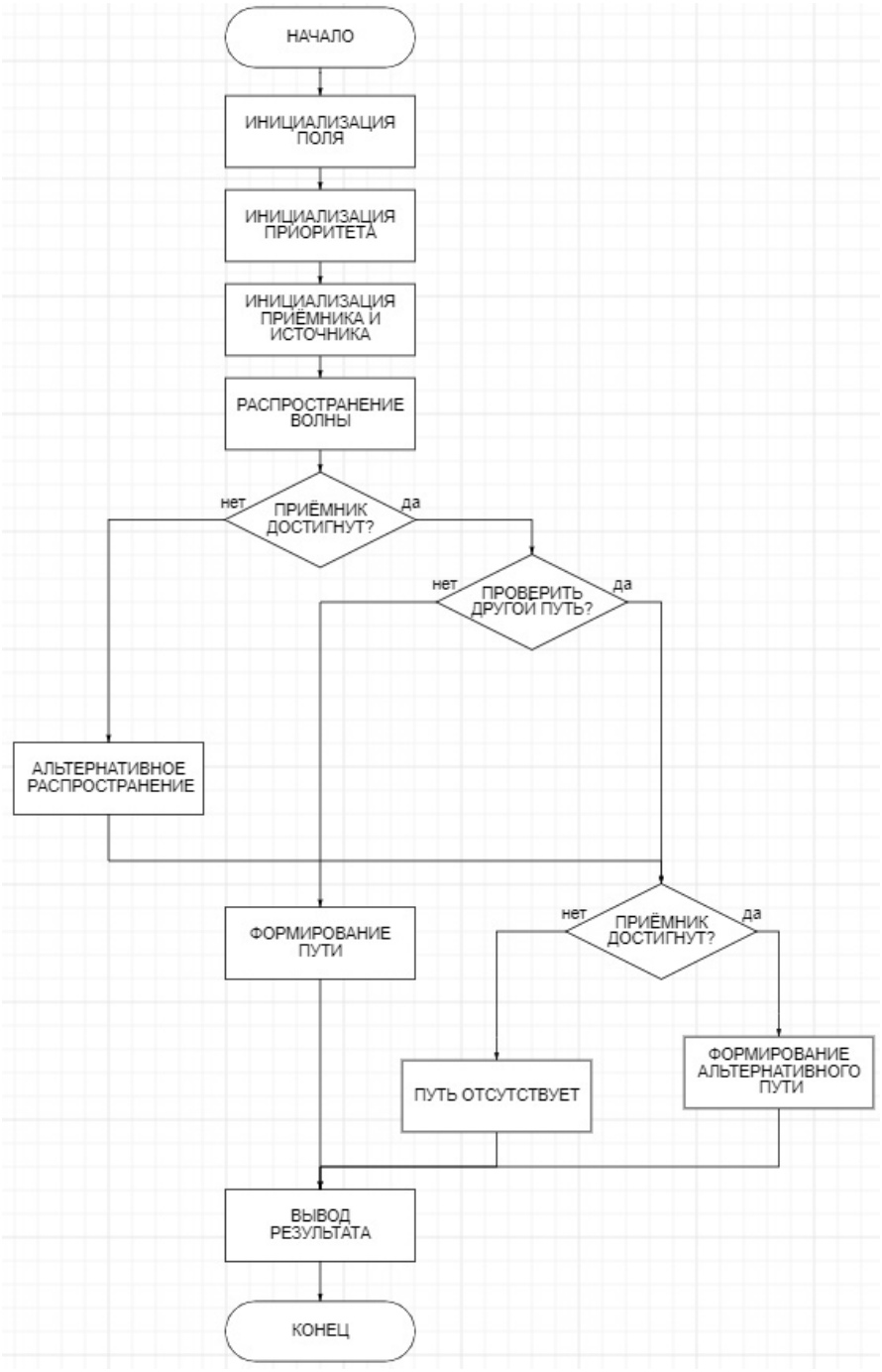

Рисунок 7 – Волновой алгоритм

Таким образом, использование волнового алгоритма позволяет протестировать генерацию 2D подземелий с комнатами и препятствиями и убедиться в правильной работе генератора подземелий.

На диаграмме прецедентов (рисунок 8) указаны кнопки, которые будут находиться в разделе «Меню» и функции, которые к ним привязаны.

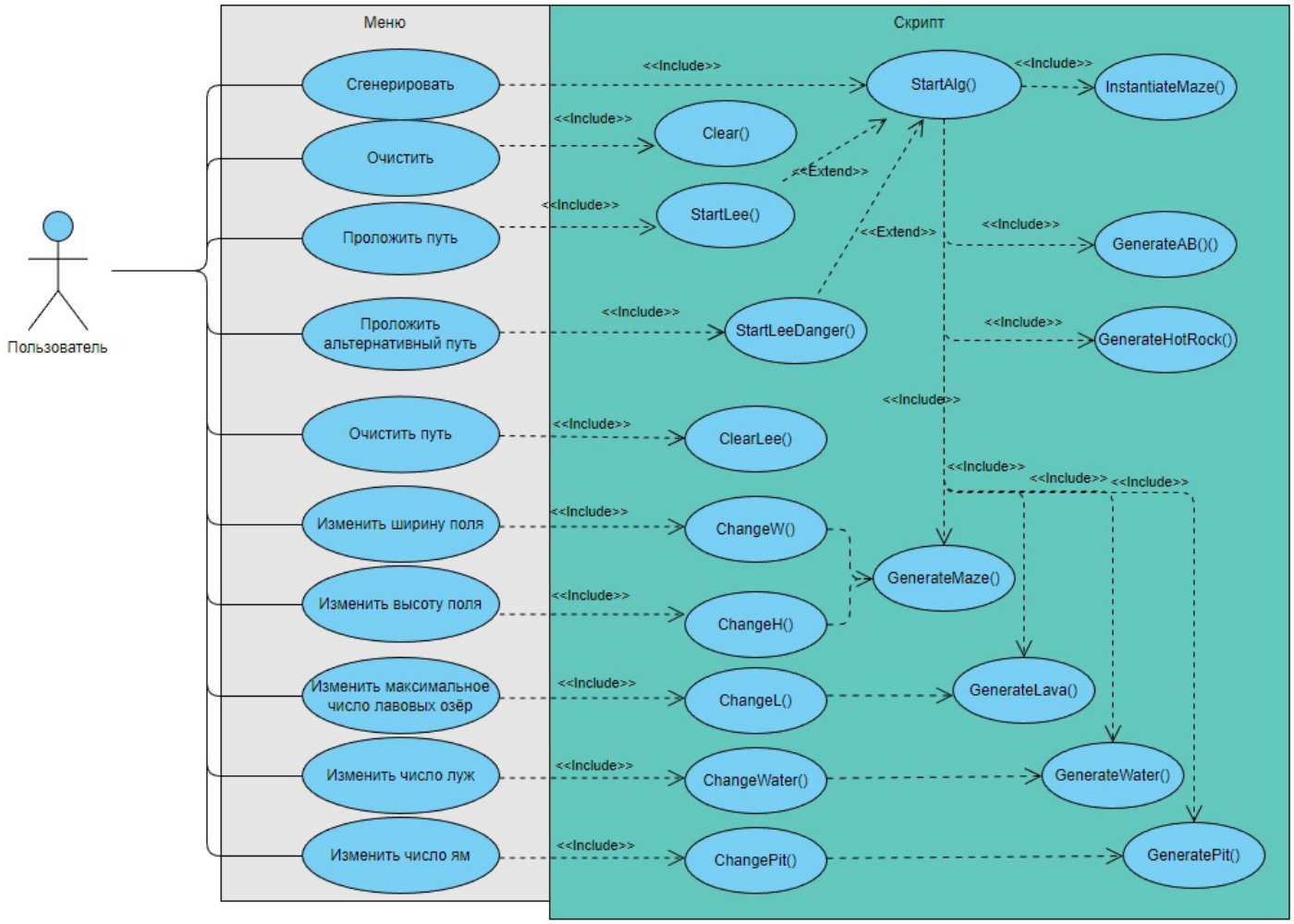

Рисунок 8 – Диаграмма прецедентов

Кнопка «Сгенерировать» позволяет сгенерировать игровой уровень случайным образом.

Кнопка «Очистить» нужна для того, чтобы удалить сгенерированный до этого игровой уровень.

Кнопка «Проложить путь» отвечает за визуальное представление маршрута от точки А, к точке Б.

Кнопка «Проложить альтернативный путь» отвечает за визуальное представление маршрута от точки А, к точке Б. Маршрут будет проложен так же по желательным для избегания клеткам. Под альтернативным маршрутом понимается тот маршрут, который подразумевает прохождение по клеткам «HotRock» (горячий пол).

Кнопка «Очистить путь» позволяет удалить все маршруты на карте.

Поле для ввода «Изменить ширину поля» меняет ширину поля. Изменения вступят в силу со следующей генерации.

Поле для ввода «Изменить высоту поля» меняет высоту поля. Изменения вступят в силу со следующей генерации.

Поле для ввода «Максимальное количество лавовых озер» позволяет изменить количество лавовых озер до указанного значения.

Поле для ввода «Количество луж» позволяет изменить количество луж согласно введённому значению.

Поле для ввода «Количество ям» позволяет изменить количество ям согласно введённому значению.

3 Разработка

В данной главе рассмотрен процесс разработки игровых уровней. Было описано создание некоторых элементов проекта. Также были описаны скрипты генерации и верификации игрового уровня.

В заключительной части главы описаны результаты разработки, приведены примеры созданных игровых уровней и процесс поиска пути.

3.1 Создание проекта

Для начала создадим пустой проект в Unity. Для этого откроем Unity и в появившемся окне нажмем на кнопку New, рисунок 9.

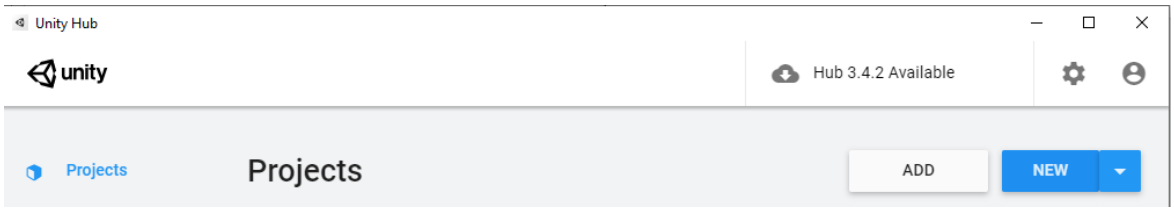

Рисунок 9 – Создание проекта в Unity

В открывшемся окне необходимо выбрать имя для нового проекта, путь, где будет храниться проект, указать организацию и выбрать шаблон для нового проекта, по умолчанию стоит 2D. Затем необходимо нажать на кнопку CREATE, рисунок 10.

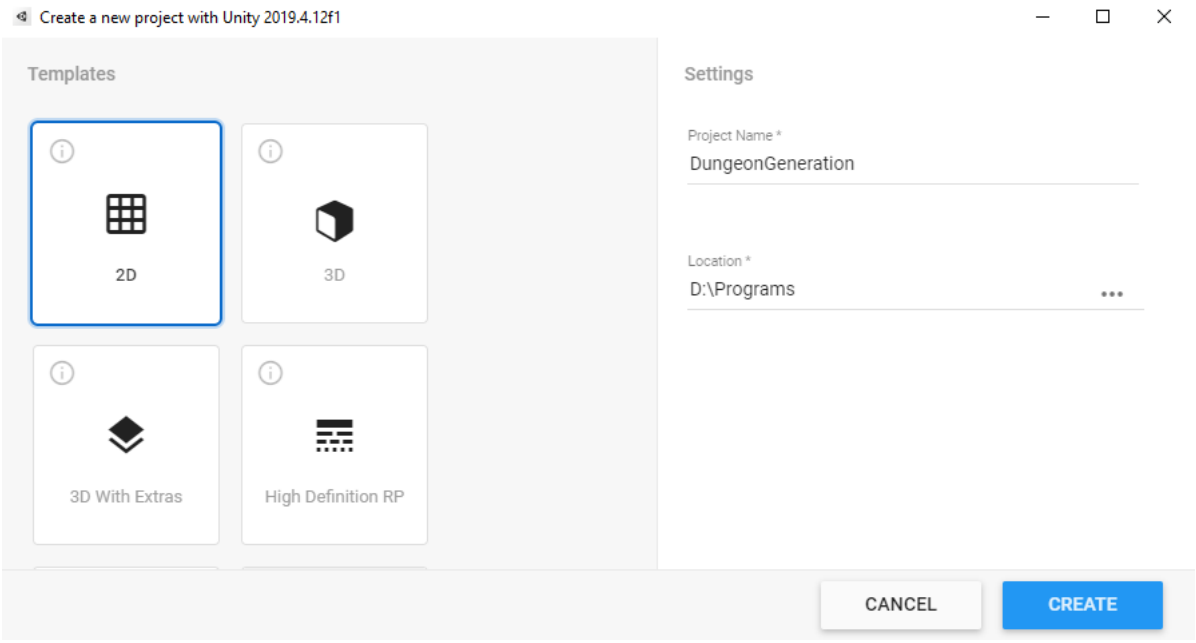

Рисунок 10 – Добавление информации о проекте.

После загрузки проекта, откроется главное окно в Unity. Основные разделы в Unity это: иерархия, проект, сцена, инспектор и игра.

Раздел иерархия содержит список всех игровых объектов в текущей сцене. Некоторые из них являются прямыми экземплярами файлов активов (например, 3D-модели), а другие-экземплярами сборных панелей, которые являются пользовательскими объектами, составляющими большую часть игры. По мере добавления и удаления объектов в сцене они также будут появляться и исчезать из Иерархии. Раздел иерархия изображен на рисунке 11.

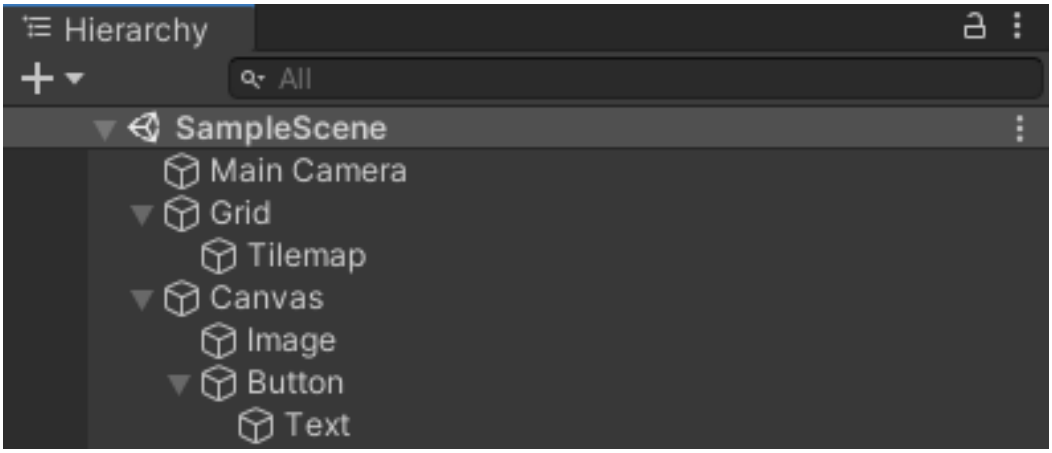

Рисунок 11 – Раздел иерархия

Раздел Проект содержит все файлы, используемые в проекте, такие как текстуры, аудио, скрипты. С помощью этого раздела выполняется весь основной менеджмент, например, создание, удаление и редактирование файлов. Раздел проект изображен на рисунке 12.

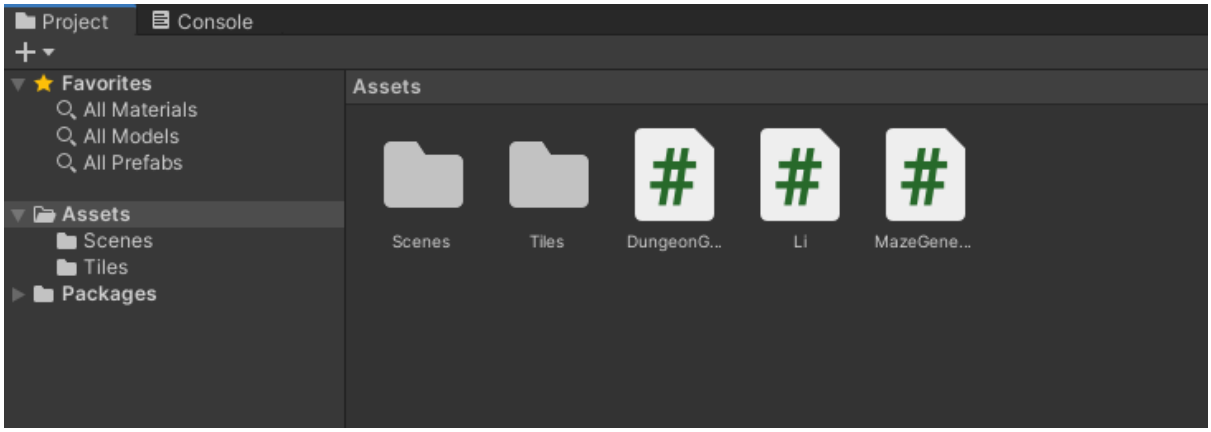

Рисунок 12 – Раздел проект

Раздел сцена используются для интерактивного взаимодействия с игровыми объектами, их размещение и позиционирование происходит именно в этом разделе.

Раздел игра является финальным видом игры, иначе говоря, раздел игра — это то, что будет видеть игрок.

Раздел инспектор содержит в себе всю информацию об объектах. В разделе инспектор можно изменять объекты, например, их позицию, текстуры, настраивать входные значения для прикрепленных к объекту скриптов. Раздел инспектор изображен на рисунке 13.

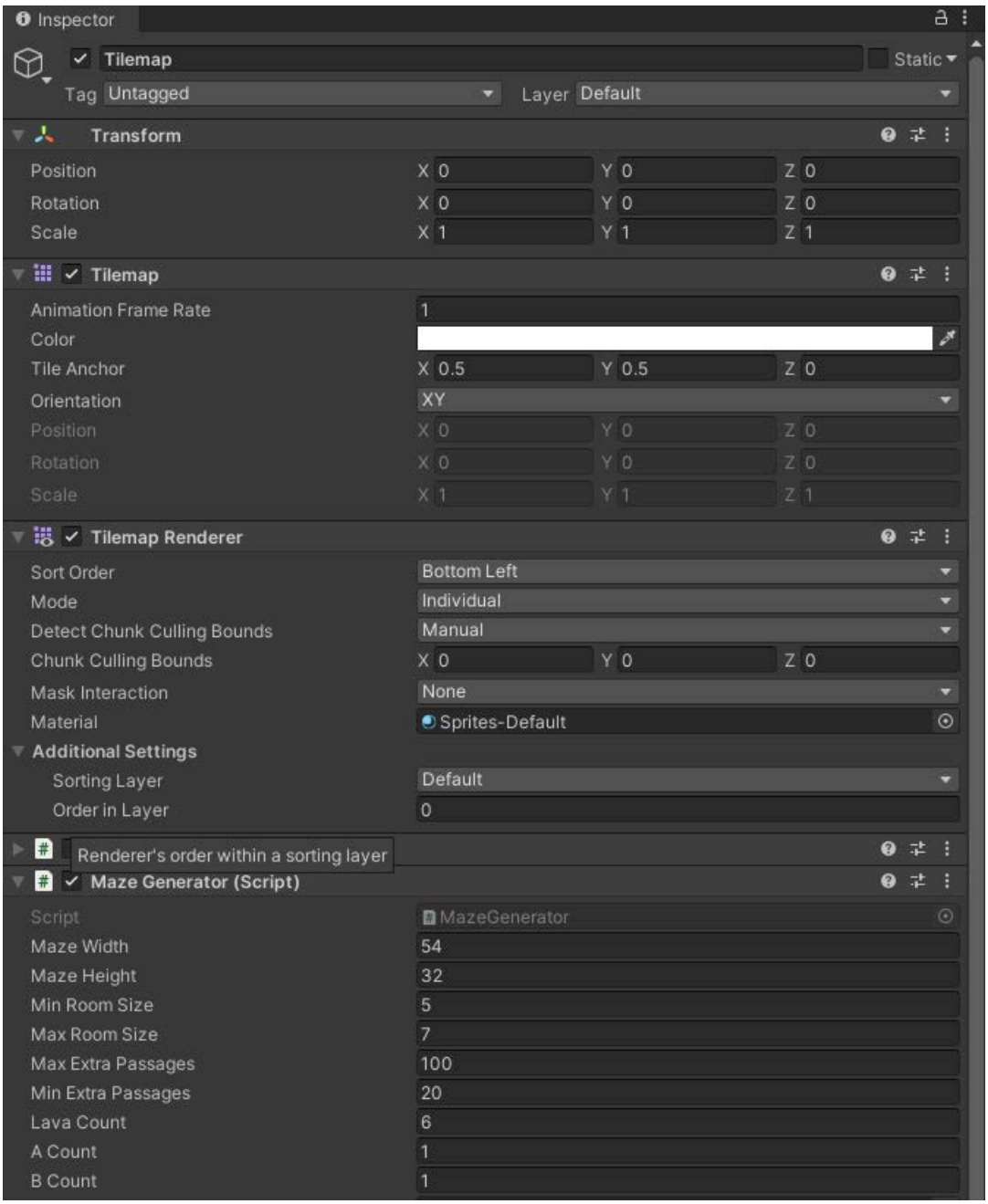

Рисунок 13 – Раздел инспектор

Для того, чтобы создать новый скрипт необходимо в разделе проект в свободном месте кликнуть правой кнопкой мыши, выбрать Create и в появившейся вкладке нажать на C# Script. Таким образом был создан скрипт. Превью скрипта можно увидеть в окне инспектора, кликнув по скрипту левой кнопкой мыши, рисунок 14.

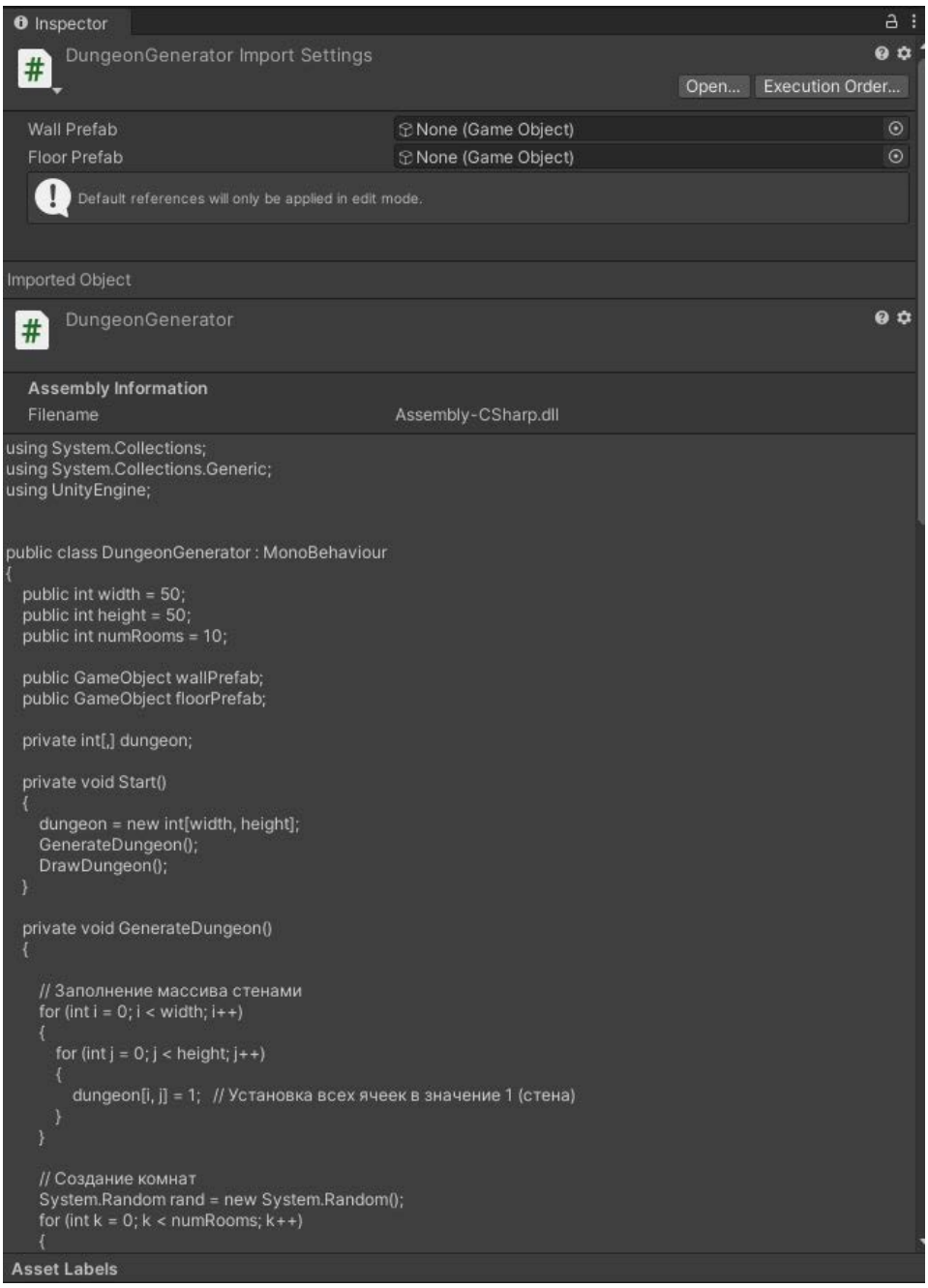

Рисунок 14 – Превью созданного скрипта

Можно заметить, что новый скрипт создался с двумя пустыми методами. Структура Unity такова, что при инициализации игры идет выполнение метода Start(), а потом при покадровом обновлении вызывается метод Update(). Благодаря такому подходу очень удобно создавать скрипты, которые будут выполняться по ходу всей игры.

Для вызова меню настроек генерации уровня была создана кнопка, по которой вызывается данная панель (рисунок 15).

Рисунок 15 – Создание кнопки

На кнопку был привязан скрипт (рисунок 16), который позволяет при нажатии на неё открывать меню генерации.

OpenPanel

Assembly Information

Filename Assembly-CSharp.dll

```
using UnityEngine;

public class OpenPanel : MonoBehaviour
{
    public GameObject panel;

    private bool isPanelVisible;

    private void Start()
    {
        isPanelVisible = false;
        panel.SetActive(false);
    }

    public void TogglePanel()
    {
        isPanelVisible = !isPanelVisible;
        panel.SetActive(isPanelVisible);

    }
}
```

Рисунок 16 – Превью скрипта для кнопки «Меню настроек генерации»

В Canvas было создано всё необходимое для функционала меню настроек генерации уровня.

В Canvas было использовано три типа UI элементов:

- Панель
- Поле ввода

– Кнопка

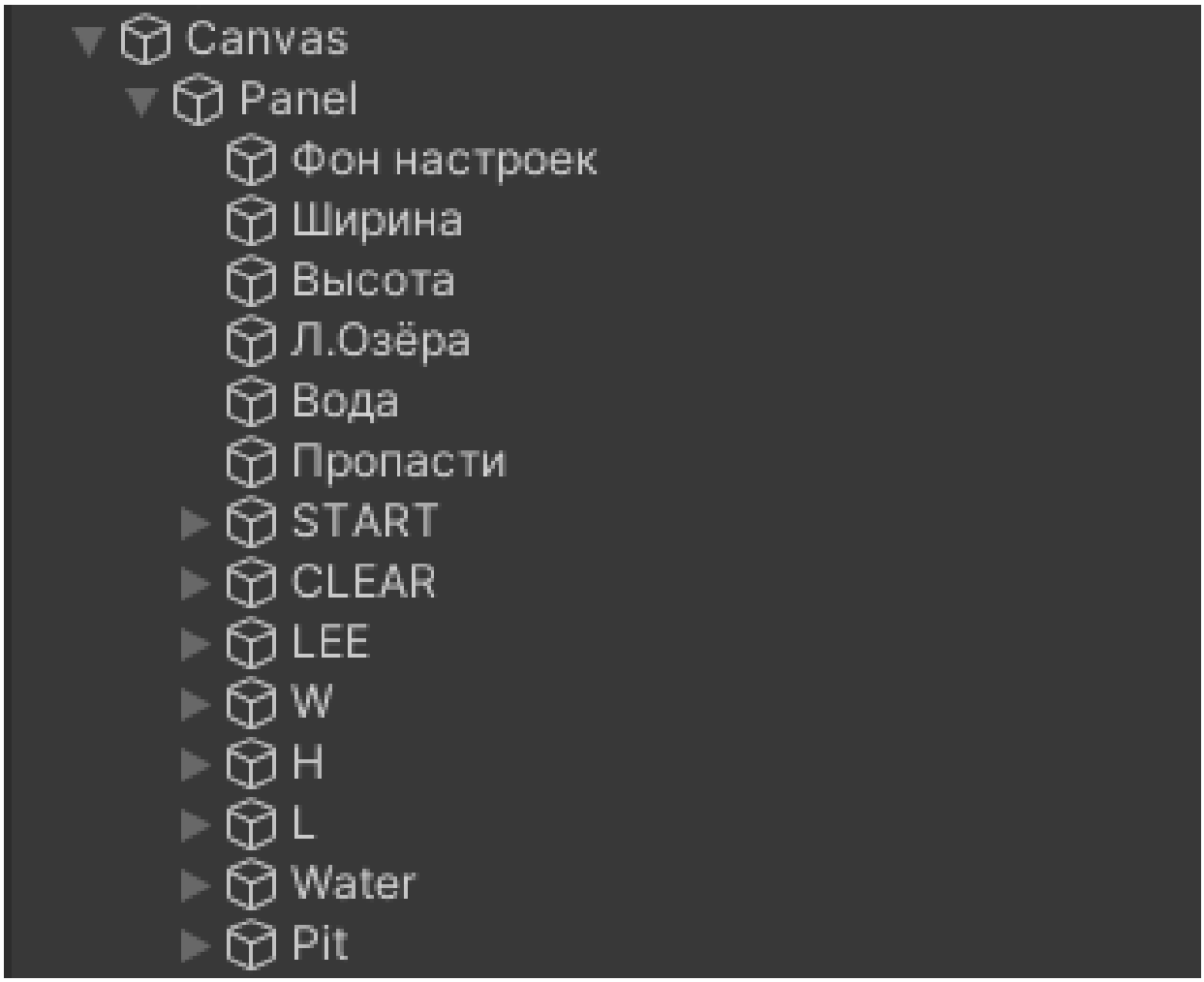

Рисунок 17 – Элементы Canvas

Для генерации уровня, была создана кнопка «Сгенерировать». Были созданы поля «Ширина поля», «Высота поля», «Максимальное количество лавовых озёр», «Количество луж», «Количество ям» (рисунок 18). При запуске приложения для этих параметров выставлены настройки по умолчанию, которые можно менять по усмотрению пользователя. Однако, для всех параметров установлена проверка на допустимые значения.

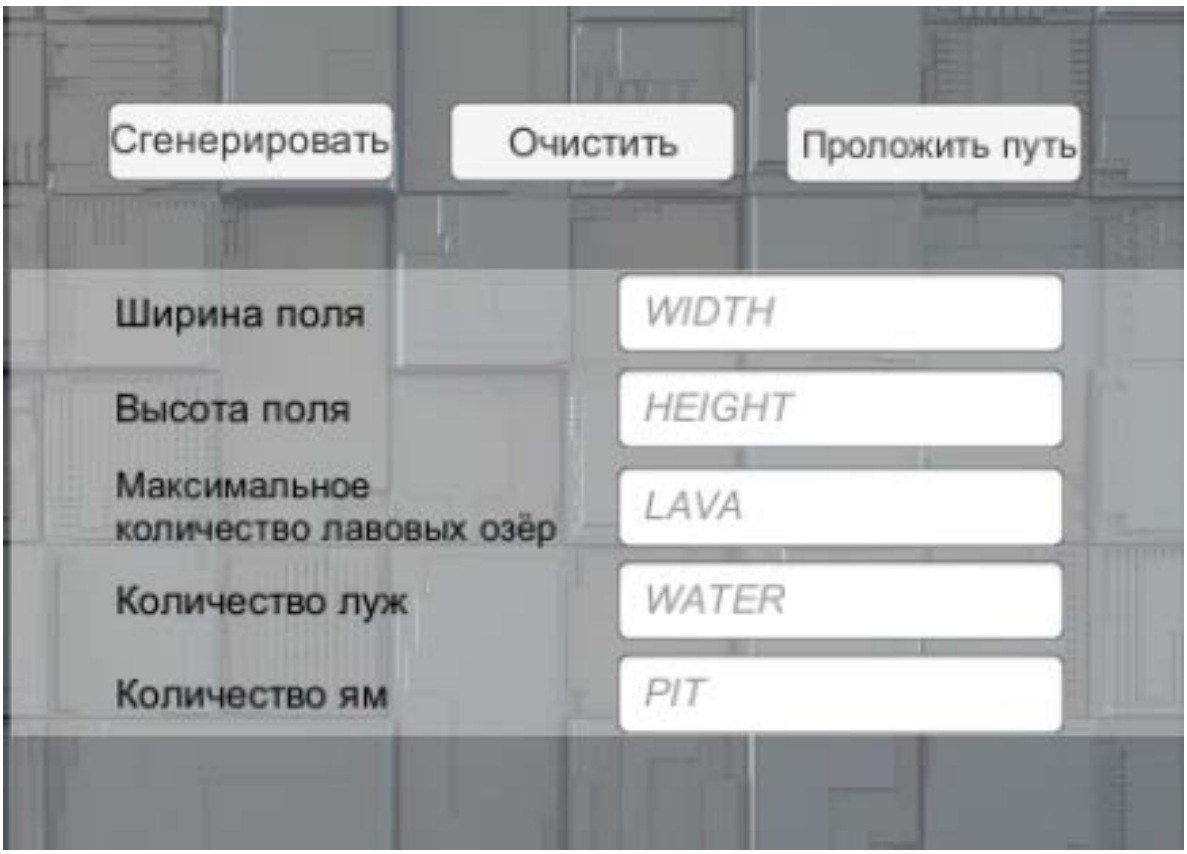

Рисунок 18 – Меню генерации уровня

Далее, для визуализации алгоритма Ли, так же была создана кнопка «Проложить путь». Если безопасный путь не найден, добавлена возможность поиска альтернативного пути. Под альтернативным маршрутом понимается тот маршрут, который подразумевает прохождение про клеткам «HotRok» (горячий пол). Однако, если прохождение уровня невозможно безопасным маршрутом, то рекомендуется перегенерировать уровень, таким образом, чтобы не затрагивать, желательные для избегания, блоки. Для поиска альтернативного пути создана кнопка «Альтернативный путь»

Добавлена возможность очищать сгенерированную карту от визуализированного маршрута поиска пути, для этого была создана кнопка «Очистить путь» (рисунок 19).

Рисунок 19 – Меню генерации уровня

После генерации уровня с большими параметрами высоты и ширины поля, может возникнуть такая ситуация, что части карты становятся не видны пользователю, так как они выходят за границы камеры (рисунок 20).

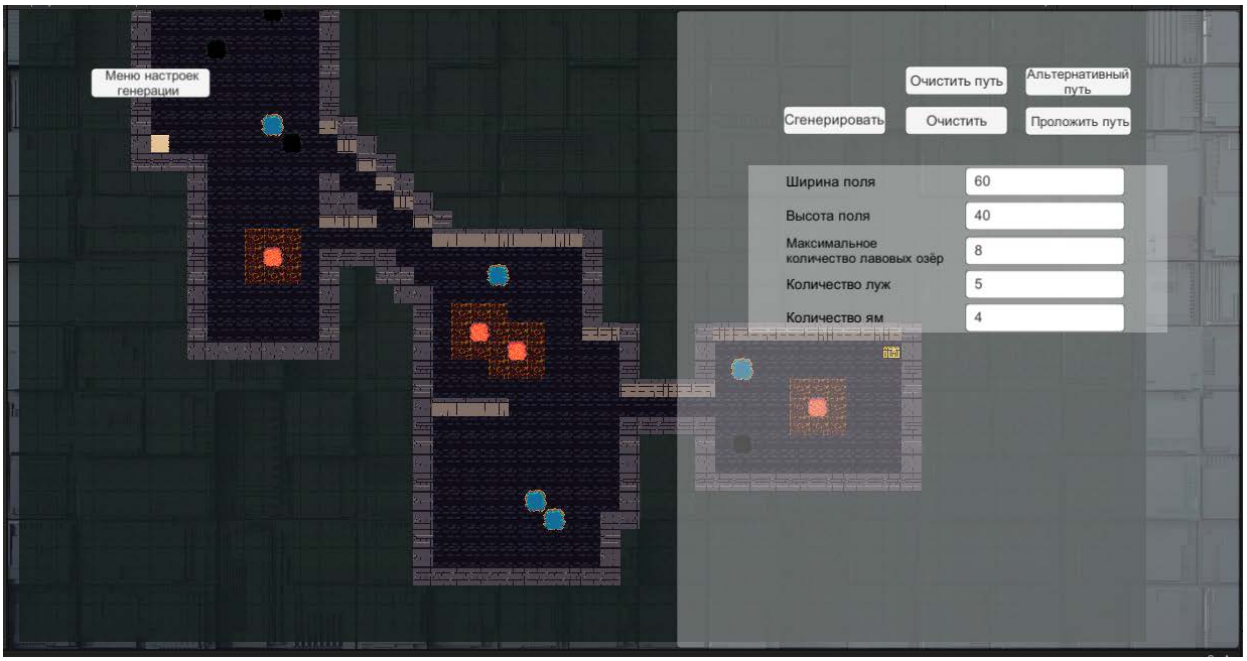

Рисунок 20 – Некорректное отображение сгенерированного уровня.

Для исправления данного недочета, для камеры был написан соответствующий скрипт (рисунок 21).

```
using UnityEngine;

public class CameraZoom : MonoBehaviour
{
    public float zoomSpeed = 20f;
    public float minZoom = 2f;
    public float maxZoom = 1000f;
    public float moveSpeed = 100f;

    private Camera mainCamera;

    private void Awake()
    {
        mainCamera = Camera.main;
    }

    private void Update()
    {
        // Получаем значение оси ввода для зума (по умолчанию используется "Mouse ScrollWheel")
        float zoomInput = Input.GetAxis("Mouse ScrollWheel");

        // Вычисляем новое значение зума
        float newZoom = mainCamera.orthographicSize - zoomInput * zoomSpeed;

        // Ограничиваем зум в пределах minZoom и maxZoom
        newZoom = Mathf.Clamp(newZoom, minZoom, maxZoom);

        // Применяем новое значение зума к камере
        mainCamera.orthographicSize = newZoom;

        // Получаем значение осей ввода для перемещения (по умолчанию используются "Horizontal" и "Vertical")
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        // Вычисляем смещение на основе ввода и скорости перемещения
        Vector3 moveOffset = new Vector3(horizontalInput, verticalInput, 0f) * moveSpeed * Time.deltaTime;

        // Применяем смещение к позиции камеры
        transform.position += moveOffset;
    }
}
```

Рисунок 21 – Скрипт для управления камерой.

С помощью данного скрипта с помощью колесика мыши можно управлять масштабом изображения, а с помощью стрелок на клавиатуре двигать камеру по четырем направлениям.

3.2 Разработка алгоритма

Стартовой точкой для создаваемого алгоритма станет DungeonGenerator. Класс будет иметь 5 глобальных переменных, с помощью которых пользователь сможет настраивать параметры генерации уровней из меню.

Список переменных:

– mazeWidth

– mazeHeight
– lavaCount
– WaterCount
– PitCount

В методе StartAlg() запускаются все необходимые методы для генерации уровня.

GenerateMaze() предназначен для генерации самого подземелья, которое состоит из комнат и проходами между ними. За генерацию комнат отвечает метод GenerateRooms(), а за генерацию коридоров метод GenerateRoomPaths.

Метод GenerateLava() предназначен для генерации лавовых озер. Логика расположения лавовых озёр такова, что они не могут генерироваться в проходах, чтобы не блокировать пути для игрока. Озера располагаются только в комнатах подземелья.

Метод GenerateHotRock() предназначен для генерации горячего пола. Горячий пол генерируется только вокруг лавовых озер и, в отличие от них, является проходимой, но желательной к избеганию, клеткой.

Метод GenerateWater() предназначен для генерации луж, которые в свою очередь, являются непроходимой клеткой. Лужи не генерируются рядом (на соседней клетке) с горячим полом.

Метод GeneratePit() предназначен для генерации пропастей, которые в свою очередь, являются непроходимой клеткой. Пропасти не генерируются рядом (на соседней клетке) с горячим полом или лужей.

Методы GenerateUp() и GenerateDown() предназначены для генерации стен вокруг игрового поля. Стены, сгенерированные этими методами, служат лишь для приятной глазу визуализации игрового уровня и носят только эстетический характер.

Метод GenerateAB() отвечает за создание точки А (точка спавна игрока) и точки Б (на ней находится конечная цель прохождения уровня).

Метод InstantiateMaze() служит для создание основного массива (рисунок 22) из которого будет создан игровой уровень (рисунок 23).

В таблице 2 указаны условные обозначения для создаваемого двумерного массива.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 12 11 11 11 11 11 11 12 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 12 11 11 11 11 11 11 11 11 11 11 11 11 11 11 12 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1
12 11 11 11 11 0 0 0 0 0 0 0 0 0 3 3 3 0 0 0 0 0 0 0 0 0 11 11 11 11 11 11 11 11 12 1 1 1 1 1
12 0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 12 1 1 1 1 1
12 0 0 0 0 0 3 3 3 0 0 0 0 0 3 3 3 0 0 11 0 0 0 0 0 0 12 0 0 0 0 3 2 3 12 1 1 1 1 1
12 0 4 0 0 0 3 2 3 0 0 0 0 3 3 3 0 0 0 0 0 0 0 0 0 0 12 0 0 0 0 3 3 3 12 1 1 1 1 1
12 0 0 0 0 0 3 3 3 0 0 0 0 3 2 3 0 0 0 12 12 12 12 12 12 12 12 0 0 0 0 0 0 9 12 1 1 1 1 1
12 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 0 0 0 12 1 1 1 1 1 1 12 0 0 0 0 0 0 0 12 1 1 1 1 1
12 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 4 0 12 1 1 1 1 1 1 12 0 0 0 0 0 0 0 12 1 1 1 1 1
12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 12 12 12 12 12 12 12 12 12 1 1 1 1 1
12 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
12 12 12 12 12 12 12 12 12 12 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 12 12 12 12 11 11 11 11 0 0 11 11 11 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 3 3 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 2 3 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 3 3 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 4 0 4 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 12 12 12 12 12 12 12 12 12 12 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Рисунок 22 – Двумерный массив для создания игрового уровня

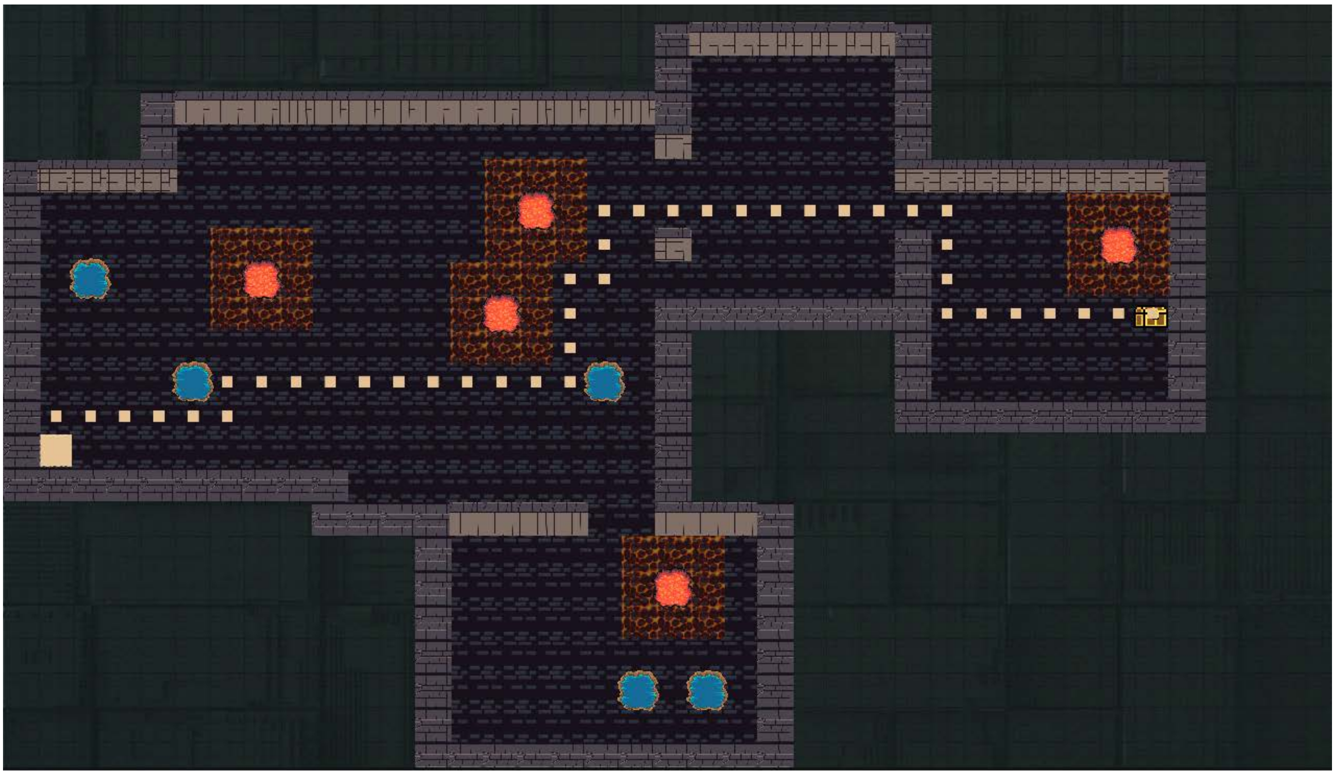

Рисунок 23 – Игровой уровень, созданный из массива

Таблица 2 – Условные обозначения для двумерного массива на этапе генерации

Значение	Обозначение	Свойства в игровом процессе
0	Пол	Проходимая клетка
1	Пределы карты	Непроходимая клетка
2	Лавовое озеро	Непроходимая клетка
3	Горячий пол	Проходимая клетка, желательная к избеганию
4	Лужа	Непроходимая клетка
5	Яма	Непроходимая клетка
11	Стена	Непроходимая клетка
12	Стена	Непроходимая клетка
8	Точка А	Точка спавна
9	Точка Б	Конечная цель

Методы ChangeW(), ChangeH(), ChangeL(), ChangeWater() и ChangePit() предназначены для работы меню. Эти методы привязаны к полям ввода, в которые пользователь вносит значения для настройки генерации уровней. С помощью меню осуществляется настройка генерации уровней. Пользователь может указать размеры создаваемого поля, количество лавовых озер, луж и пропастей.

Метод StartLee() запускает алгоритм Ли по созданному массиву. Приоритет распространения (рисунок 24) определен таким образом, так как точка спавна в большинстве случаев находится в левом нижнем углу, а целевая точка в правом верхнем углу сгенерированного уровня.

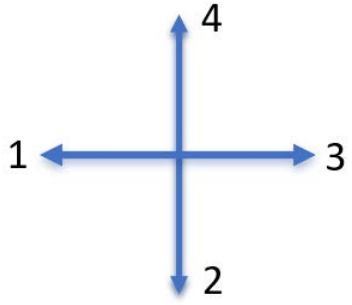

Рисунок 24 – Приоритет распространения в данном алгоритме

Во время работы алгоритма, на основном массиве (рисунок 25) происходит распространение волны от источника (точка А) к приёмнику (точка Б).

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 11 11 11 11 11 11 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 4 0 0 11 11 11 11 11 11 11 11 11 12 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 3 3 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 2 3 0 0 0 0 0 4 0 12 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 3 3 3 0 0 0 0 0 0 0 11 12 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 12 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 11 12 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 0 0 11 11 11 11 11 11 11 12 1 1
1 1 1 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 12 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 9 12 1 1
1 1 1 1 12 11 11 11 11 11 11 11 11 12 12 12 12 12 12 12 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 1 1
1 1 1 1 12 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 12 0 0 0 0 3 3 3 0 0 0 0 0 0 0 0 0 12 1 1
1 1 1 1 12 3 3 3 0 0 0 0 0 11 11 11 11 11 11 11 11 0 0 0 0 3 2 3 0 0 0 0 0 0 0 0 0 12 1 1
1 1 1 1 12 3 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 0 0 0 0 0 0 0 0 0 12 1 1
1 1 1 1 12 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 12 1 1
1 1 1 1 12 0 0 0 0 0 0 0 0 12 12 12 12 12 12 12 12 12 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 12 1 1
1 1 1 1 12 0 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 12 0 0 4 0 0 0 12 12 12 12 12 12 12 12 12 1 1
1 1 1 1 12 8 0 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 12 0 0 0 0 0 0 12 1 1 1 1 1 1 1 1 1 1
1 1 1 1 12 12 12 12 12 12 12 12 12 12 1 1 1 1 1 1 1 1 12 12 12 12 12 12 12 12 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Рисунок 25 – Двумерный массив для создания игрового уровня

Волна распространяется на все проходимые клетки, а непроходимые помечаются в массиве нулём (рисунок 26).

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 49 48 47 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 49 0 47 46 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 49 48 47 46 45 44 43 42 41 42 43 44 45 46 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 49 48 47 46 45 0 0 0 41 40 41 42 43 44 45 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 48 47 46 45 44 0 0 0 40 39 40 41 42 0 44 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 47 46 45 44 43 0 0 0 39 38 39 40 41 42 43 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 46 45 44 43 42 41 40 39 38 37 38 39 40 41 42 43 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 45 44 43 0 41 40 39 38 37 36 37 38 39 40 41 42 43 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 44 43 42 41 40 39 38 37 36 35 36 37 38 39 40 0 42 43 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 45 44 43 42 41 40 0 0 35 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 0 0 0
0 0 0 0 0 22 21 20 19 20 21 22 23 0 0 0 0 0 0 0 0 32 33 34 35 0 0 0 39 40 41 42 43 44 45 46 47 0 0 0
0 0 0 0 0 0 0 0 18 19 20 21 22 0 0 0 0 0 0 0 0 31 32 33 34 0 0 0 38 39 40 41 42 43 44 45 46 0 0 0
0 0 0 0 0 0 0 0 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 0 0 0 37 38 39 40 41 42 43 44 45 0 0 0
0 0 0 0 0 0 0 0 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 0 45 46 0 0 0
0 0 0 0 0 12 13 14 15 16 17 18 19 0 0 0 0 0 0 0 0 0 0 32 33 34 35 36 37 38 39 40 41 42 43 44 45 0 0 0
0 0 0 0 0 11 12 13 14 15 16 17 18 0 0 0 0 0 0 0 0 0 0 33 34 0 36 37 38 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 10 11 12 13 14 15 16 17 0 0 0 0 0 0 0 0 0 0 34 35 36 37 38 39 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Рисунок 26 – Распространение волны в массиве

После этого, в обратном направлении, строится кратчайший путь, который в данном случае равен 39 клеткам (рисунок 27).

Рисунок 27 – Построение пути

На рисунке 28 изображена визуализация маршрута от точки А к точке Б на сгенерированном игровом уровне.

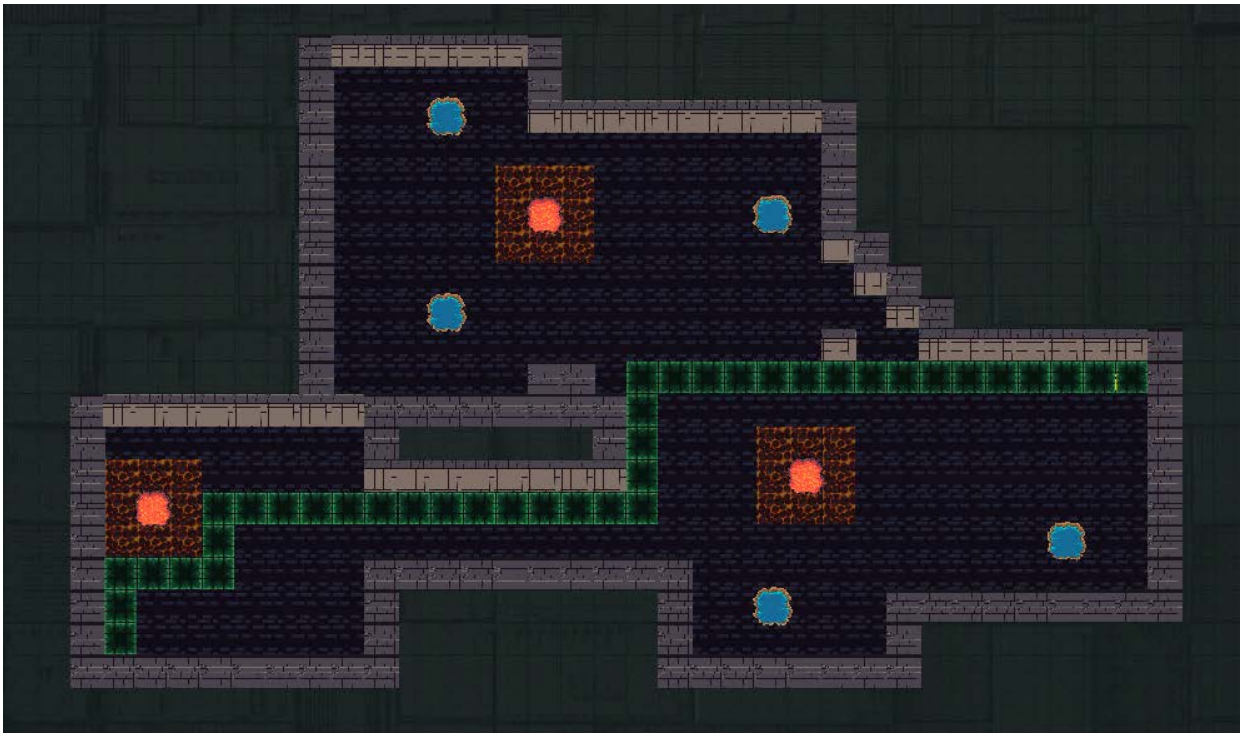

Рисунок 28 – Построение пути на игровом уровне

Если путь не может быть построен безопасным способом, то функция StartLeeDanger() позволяет проложить альтернативный маршрут (рисунок 29).

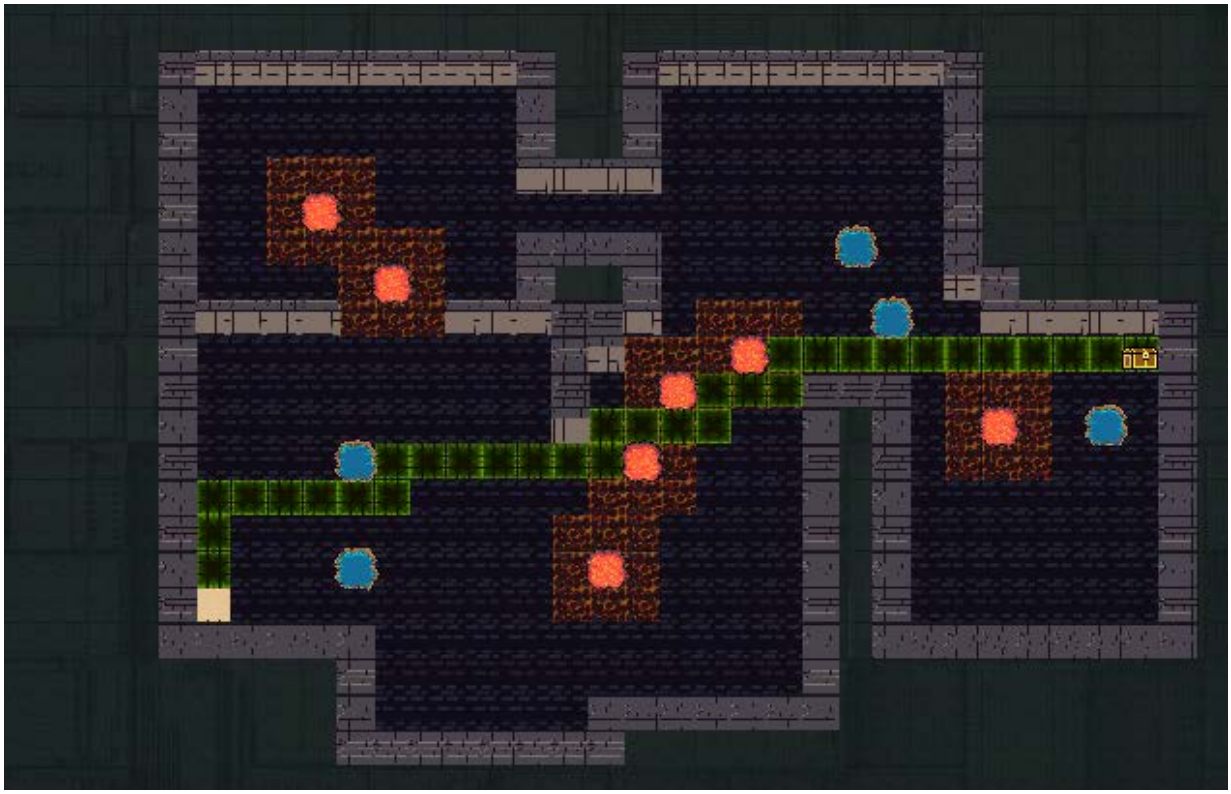

Рисунок 29 – Построение альтернативного пути на игровом уровне

Под альтернативным маршрутом понимается тот маршрут, который подразумевает прохождение про клеткам «HotRok» (горячий пол). Однако, если прохождение уровня невозможно безопасным маршрутом, то рекомендуется перегенерировать уровень, таким образом, чтобы не затрагивать, желательные для избегания, блоки.

Шаги генерации в методах StartLee() и StartLeeDanger():

- Создается новый двумерный массив wave, который заполняется нулями
- Стартовой точке A (источнику), присваивается значение 1
- По всему массиву проходит распространение волны, параллельно с распространением происходит проверка на то, не достигнута ли конечная точка.

– В массиве path, в обратном направлении, от точки Б (приёмника) к точке А (к источнику), стоится сам маршрут. Именно на данном этапе внутри программы происходит определение приоритета.

– Каждое числовое значение массива path визуализируется с помощью GameObject. Все GameObject добавляются в список, для возможности очистки игрового поля от визуализированного маршрута.

Таким образом был разработан алгоритм генерации и верификации игровых уровней с помощью алгоритма Ли. Весь исполняемый код представлен в приложении А.

3 Тестирование алгоритма

Тестирование программы является важным этапом в процессе ее разработки. Это позволяет выявить ошибки и недочеты в программе, которые могут привести к неправильной работе или даже краху системы. Тестирование также позволяет убедиться, что программа соответствует требованиям заказчика и выполняет все необходимые функции.

В текущем разделе представлено тестирование программы при разных входных данных и разных вариантов генерации игрового уровня.

Проверка работоспособности разработанного алгоритма, при корректных входных данных на этапе генерации (1)

Шаги:

1. Открыть меню проекта.
2. Заполнить входные переменные согласно таблице 3.

Таблица 3 – Входные данные для проверки работоспособности разработанного алгоритма, при корректных входных данных на этапе генерации

Параметры	**Значение**
Ширина поля	40
Высота поля	25
Максимальное количество лавовых озёр	8
Количество луж	5
Количество ям	4

Ожидаемый результат:

– Ошибок нет, на карте появились сгенерированные комнаты, между комнатами, если таковых больше одной, сгенерировались коридоры

корректным образом, размер сгенерированной карты соответствует заданным параметрам.

– Количество лавовых озёр не больше 8, количество луж равно 5, количество ям равно 4

Фактический результат:

– Ошибок нет, фактический результат полностью совпал с ожидаемым. Алгоритм выполнил свою работу корректно, результат выполнения работы алгоритма изображен на рисунке 30.

–

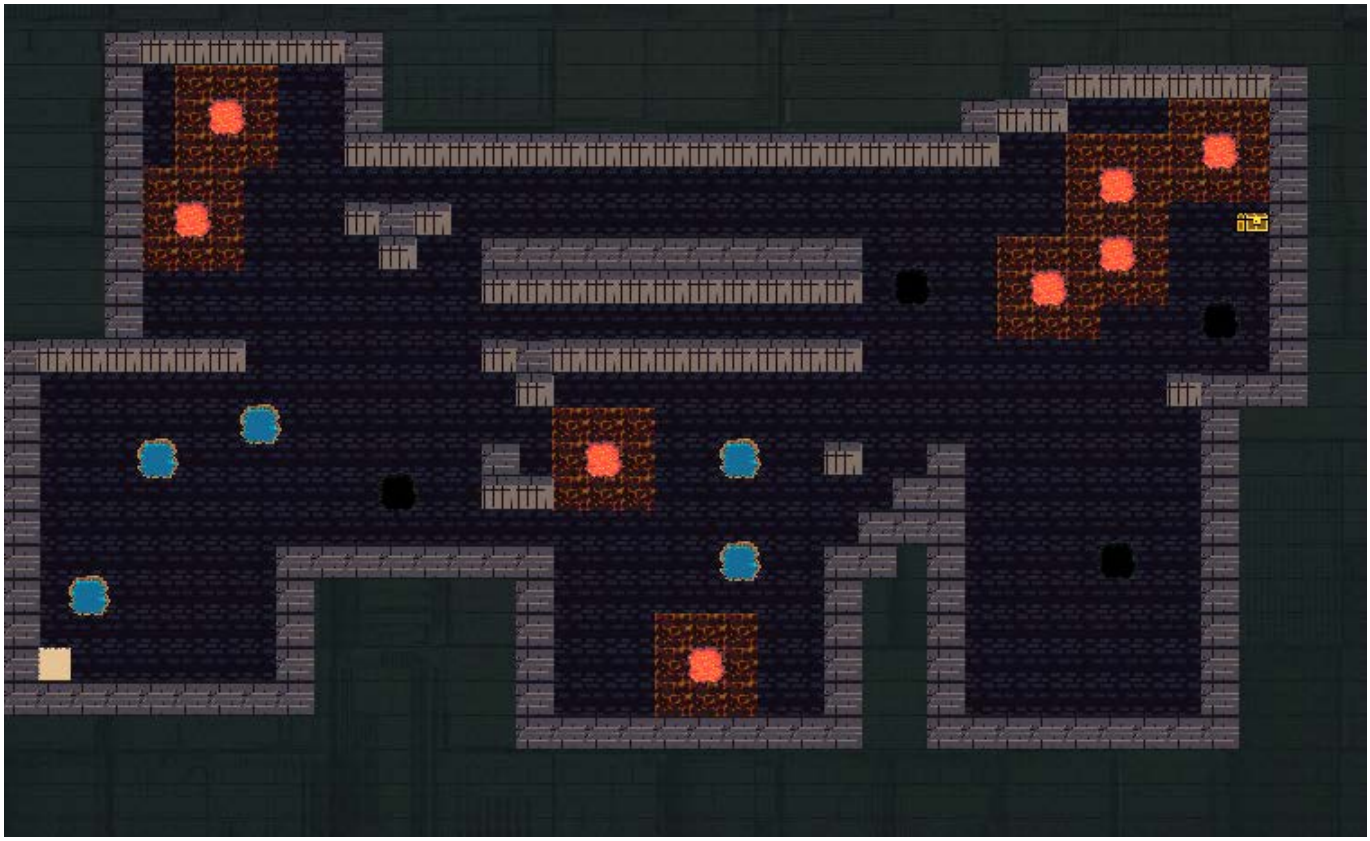

Рисунок 30 – Результат работы алгоритма

Проверка работоспособности разработанного алгоритма, при некорректных входных данных (2)

Шаги:

1. Открыть меню проекта.
2. Заполнить входные переменные согласно таблице 4.

Таблица 4 – Входные данные для проверки работоспособности разработанного алгоритма, при некорректных входных данных на этапе генерации

Параметры	**Значение**
Ширина поля	40
Высота поля	25
Максимальное количество лавовых озёр	8
Количество луж	99
Количество ям	99

Ожидаемый результат:

- Ошибок нет, на карте появились сгенерированные комнаты, между комнатами,
- если таковых больше одной, сгенерировались коридоры корректным образом, размер сгенерированной карты соответствует заданным параметрам.
- Защита от некорректных значений сократила количество некоторых генерируемых объектов до оптимального значения.

Фактический результат:

- Ошибок нет, фактический результат полностью совпал с ожидаемым. Алгоритм выполнил свою работу корректно, результат выполнения работы алгоритма изображен на рисунке 31.
- Количество лавовых озёр не больше 8, количество лужи и ямы сгенерированы, но в меньшем количестве.

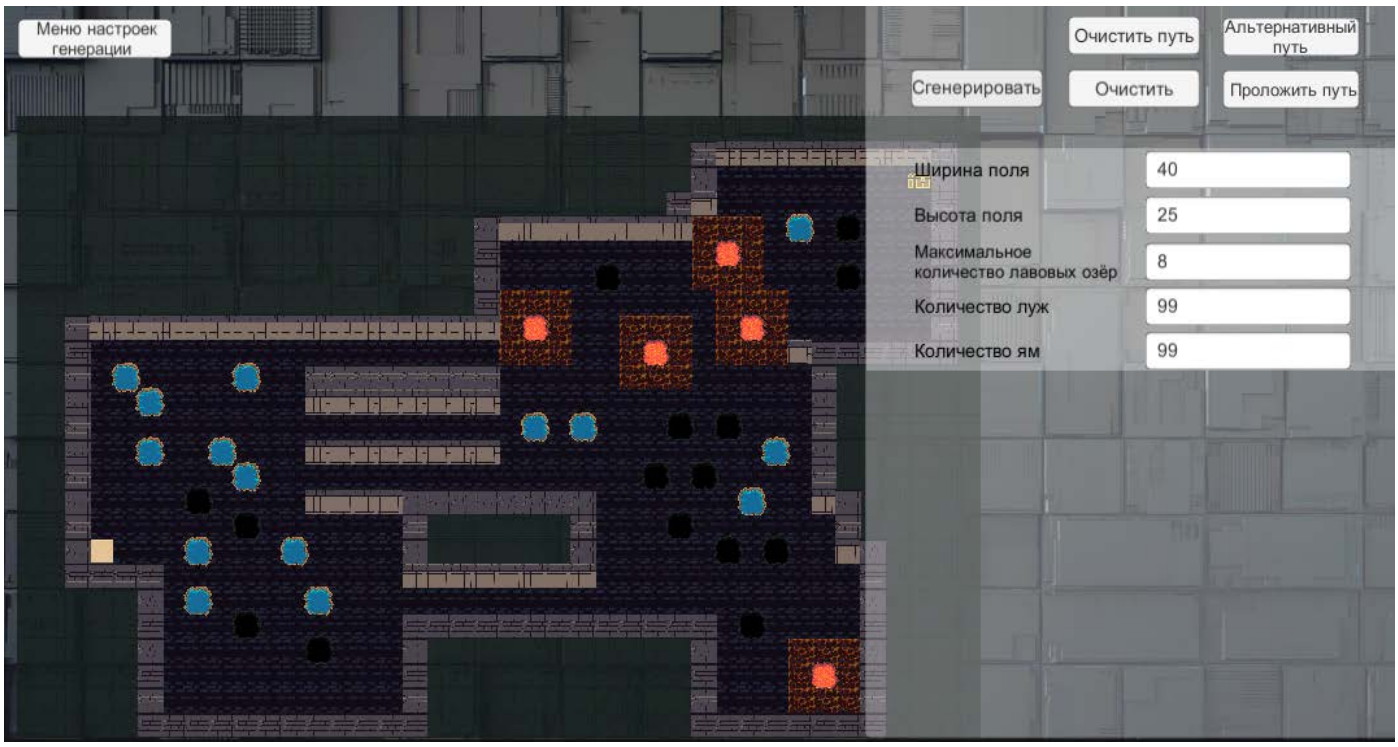

Рисунок 31 – Результат работы алгоритма

Проверка работоспособности разработанного алгоритма, при корректных входных данных на этапе поиска пути с возможностью проложить безопасный путь. (3)

Шаги:

1. Открыть меню проекта.
2. Нажать кнопку «Сгенерировать».
3. Нажать кнопку «Проложить путь».

Ожидаемый результат:

- Ошибок нет, на карте появились сгенерированные комнаты, между комнатами, если таковых больше одной, сгенерировались коридоры корректным образом, размер сгенерированной карты соответствует заданным параметрам.
- Количество объектов соответствуют заданным параметрам.
- Маршрут от точки А (от источника) к точке Б (к приёмнику) успешно проложен.

Фактический результат отображен в таблице 5:

Таблица 5 – Результат работы программы при корректных входных данных на этапе поиска пути с возможностью проложить безопасный путь

Параметр	**Значение**
Ширина поля	40
Высота поля	25
Фактическое количество лавовых озёр	4
Количество луж	5
Количество ям	4
Длина безопасного пути	41

Результат теста:

Ошибок нет, фактический результат полностью совпал с ожидаемым. Алгоритм выполнил свою работу корректно, результат выполнения работы алгоритма изображен на рисунке 32.

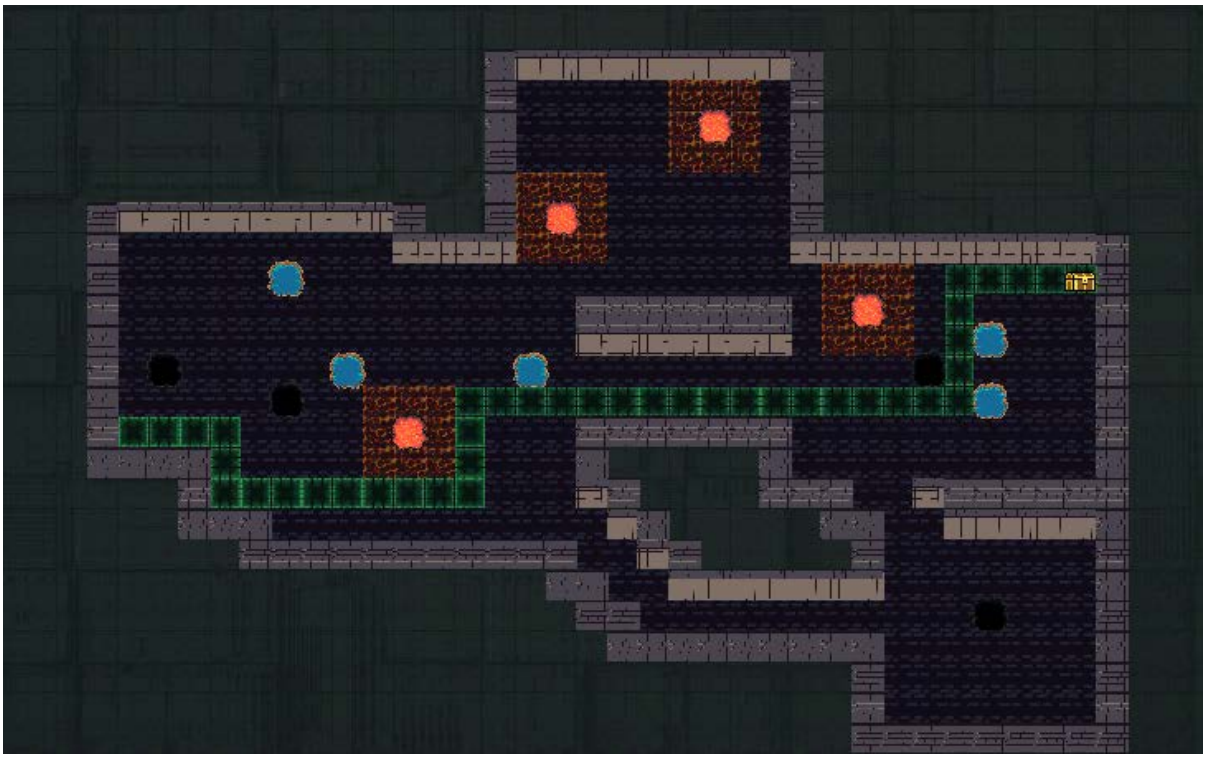

Рисунок 32 – Результат работы алгоритма

Проверка работоспособности разработанного алгоритма, при корректных входных данных на этапе поиска пути, если проложить безопасный путь невозможно. (4)

Шаги:

1. Открыть меню проекта.
2. Нажать кнопку «Сгенерировать».
3. Нажать кнопку «Проложить путь».
4. Нажать на кнопку «Альтернативный путь».

Ожидаемый результат:

- Ошибок нет, на карте появились сгенерированные комнаты, между комнатами, если таковых больше одной, сгенерировались коридоры корректным образом, размер сгенерированной карты соответствует заданным параметрам. Количество объектов соответствуют заданным параметрам.
- При нажатии на кнопку «Проложить путь» маршрут от точки А (от источника) к точке Б (к приёмнику) проложить не удалось (рисунок 33).
- При нажатии на кнопку «Альтернативный путь» маршрут от точки А (от источника) к точке Б (к приёмнику) успешно проложен.

Фактический результат отображен в таблице 6:

Таблица 6 – Результат работы программы при корректных входных данных на этапе поиска пути, если проложить безопасный путь невозможно

Параметр	**Значение**
Ширина поля	40
Высота поля	25
Фактическое количество лавовых озёр	5
Количество луж	5
Количество ям	4

Продолжение таблицы 6

Длина безопасного пути	Не существует
Длина альтернативного пути	39

Результат теста:

Ошибок нет, фактический результат полностью совпал с ожидаемым. Алгоритм выполнил свою работу корректно, результат выполнения работы алгоритма изображен на рисунке 34.

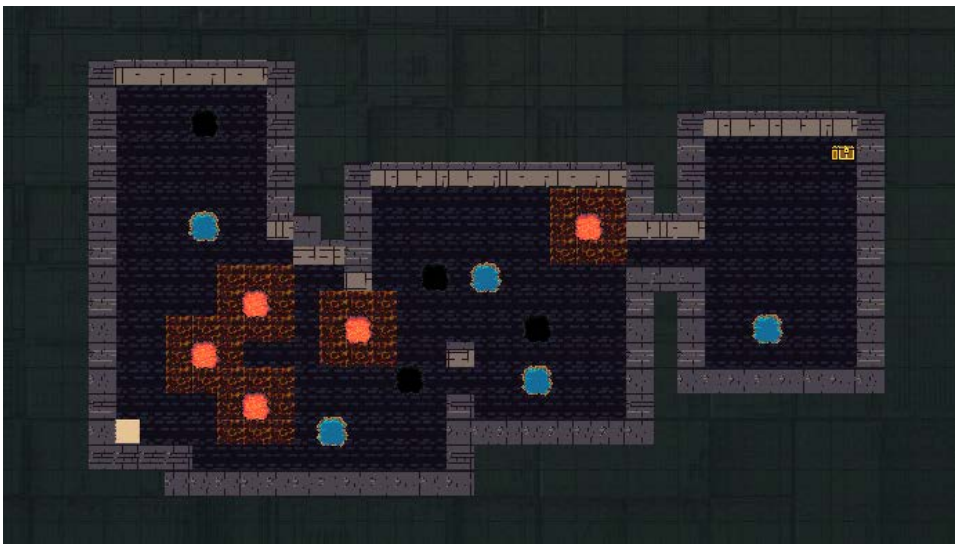

Рисунок 33 – Не удалось проложить безопасный маршрут

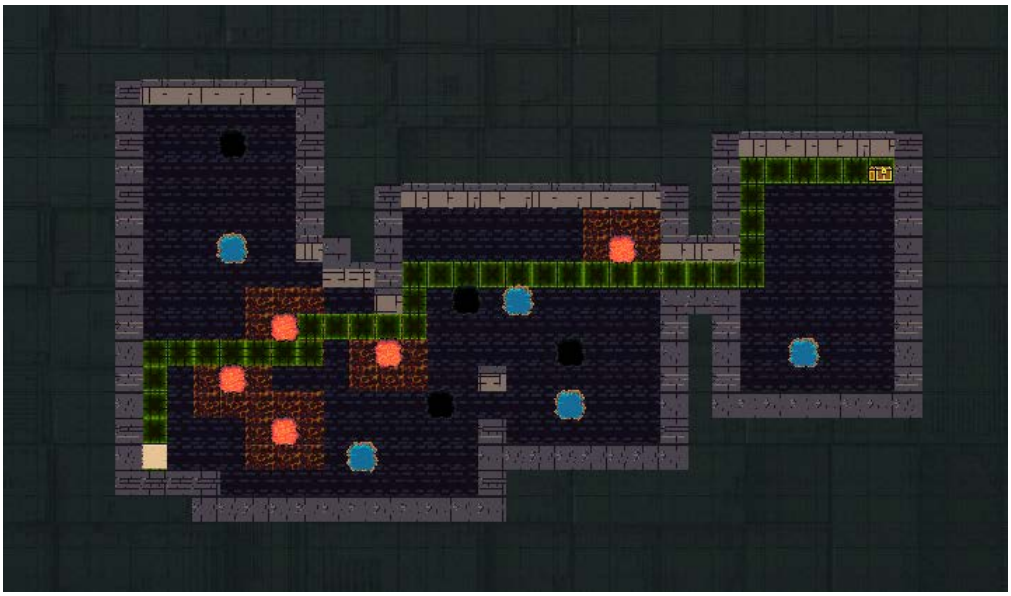

Рисунок 34 – Альтернативный маршрут

Проверка работоспособности разработанного алгоритма, при корректных входных данных на этапе поиска пути. Попытка проложить безопасный и альтернативные маршруты вместе (5)

Шаги:

5. Открыть меню проекта.
6. Нажать кнопку «Сгенерировать».
7. Нажать кнопку «Проложить путь».
8. Нажать на кнопку «Альтернативный путь».

Ожидаемый результат:

- Ошибок нет, на карте появились сгенерированные комнаты, между комнатами, если таковых больше одной, сгенерировались коридоры корректным образом, размер сгенерированной карты соответствует заданным параметрам. Количество объектов соответствуют заданным параметрам.
- При нажатии на кнопку «Проложить путь» маршрут от точки А (от источника) к точке Б (к приёмнику) успешно проложен (рисунок 32).
- При нажатии на кнопку «Альтернативный путь» маршрут от точки А (от источника) к точке Б (к приёмнику) успешно проложен.
- На сгенерированном уровне отображены сразу два маршрута. Ожидается, что альтернативный путь, будет такой же или быстрее, чем безопасный маршрут

Фактический результат:

Таблица 7 – Результат работы программы при попытке проложить безопасный маршрут и альтернативный маршрут

Параметр	**Значение**
Ширина поля	40
Высота поля	25
Фактическое количество лавовых озёр	7

Продолжение таблицы 7

Количество луж	5
Количество ям	4
Длина безопасного пути	49
Длина альтернативного пути	47

Результат теста:

– Ошибок нет, фактический результат полностью совпал с ожидаемым.
– Длина безопасного пути составила 49 клеток. Длина альтернативного пути составила 47 клеток. Как и ожидалось, альтернативный маршрут оказался быстрее, чем безопасный.
– Альтернативный путь в данном тесте оказался короче безопасного пути на 2 клетки.

Алгоритм выполнил свою работу корректно, результат выполнения работы алгоритма изображен на рисунке 35.

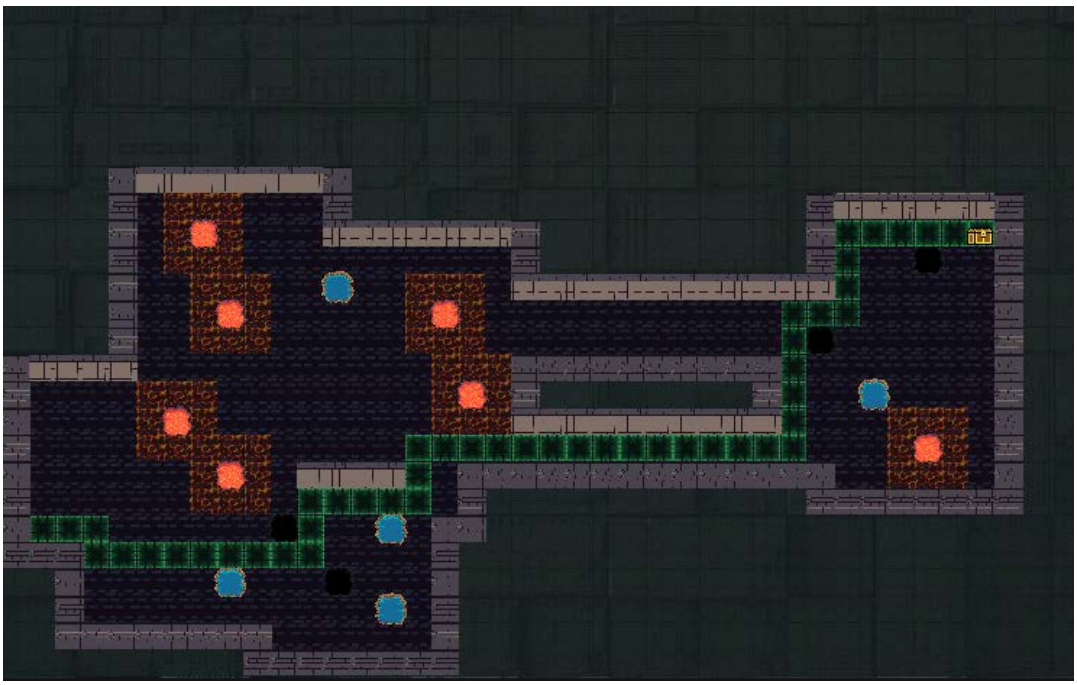

Рисунок 35 – Результат работы алгоритма (построен безопасный путь)

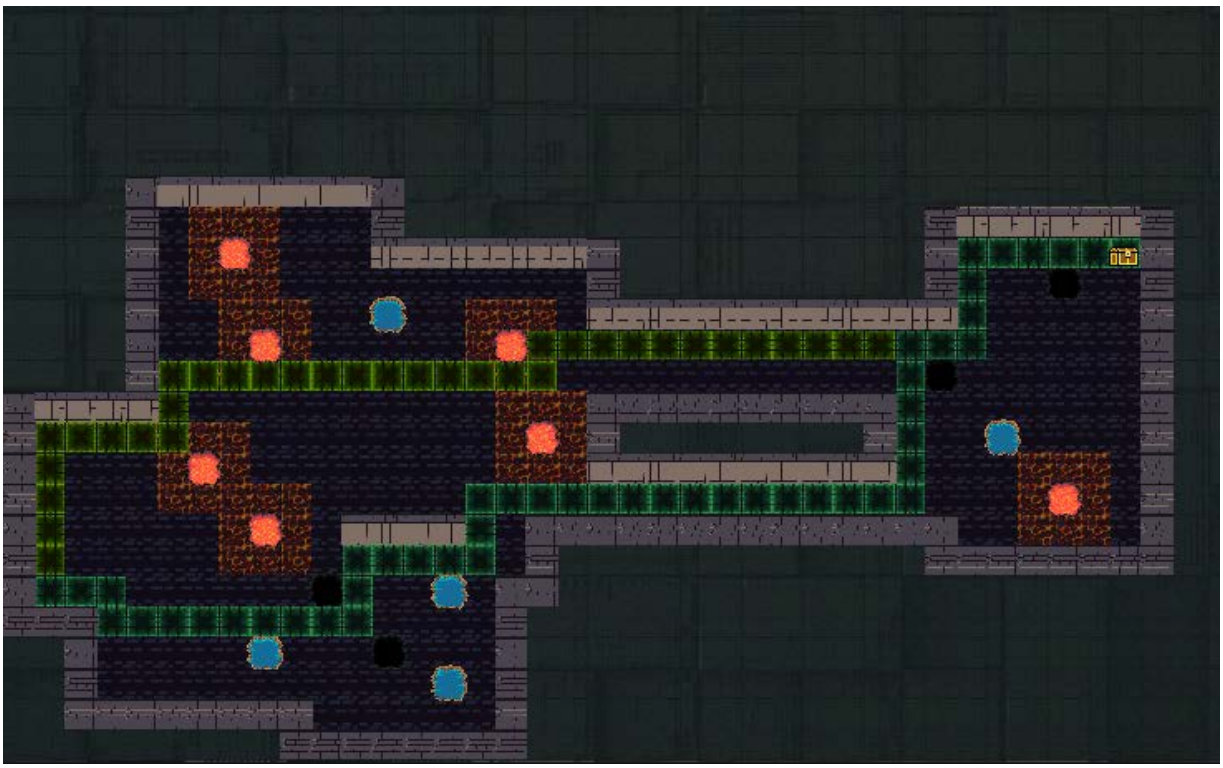

Рисунок 36 – Результат работы алгоритма (построен альтернативный путь)

Таким образом, в этой главе выполнено тестирование программы, которое доказало корректность разрабатываемого алгоритма.

В итоге было создано приложение, способное генерировать игровые уровни, с возможностью настройки параметров генерации, а также с возможностью верификации этого уровня по двум сценариям:

1. Безопасный маршрут существует.
2. Безопасного маршрута не существует. Сформирован маршрут, при котором игрок столкнется с некоторым количеством препятствий, вследствие которого здоровье игрока может уменьшится, но не будет достигать нуля.

Заключение

В выпускной квалификационной работе разработано приложение для генерации и верификации игровых уровней.

Рассмотрены другие алгоритмы поиска пути в лабиринте. Рассмотрены случаи, когда безопасный путь невозможно построить и варианты решения данной ситуации. В ходе проектирования была спроектирована программа, позволяющая производить процедурную генерацию игрового уровня.

В ходе разработки была разработана программа для генерации игровых уровней, внедрен алгоритм верификации игровых уровней. В программе реализован волновой алгоритм поиска пути в лабиринте. Выполнено тестирование программы, которое доказало корректность разрабатываемого алгоритма.

В итоге было создано приложение, способное генерировать игровые уровни, с возможностью настройки параметров генерации, а также с возможностью верификации этого уровня по двум сценариям.

Данная разработка помогает ускорить процесс создания уровней, так как с помощью автоматической проверки, уже на этапе создания уровня, выявляются ошибки генерации, о которых разработчик уведомлен заранее.

Таким образом требования технического задания было выполнено в полном объеме в соответствии с установленными сроками.

Список используемой литературы

1. Информационная система wikipedia [Электронный ресурс]: рабочая программа / URL: https://ru.wikipedia.org/wiki/Тайловая_графика (дата обращения 10.05.2023)
2. Информационная система wikipedia[Электронный ресурс]: рабочая программа / URL: https://ru.wikipedia.org/wiki/Алгоритм_Ли (дата обращения 5.05.2023)
3. Информационная система Habr [Электронный ресурс]: рабочая программа / URL: https://habr.com/Dijkstra (дата обращения 7.04.2023)
4. Информационная система Habr [Электронный ресурс]: рабочая программа / URL: https://habr.com/ASTAR (дата обращения 7.04.2023)
5. Информационная система Habr [Электронный ресурс]: рабочая программа / URL: https://habr.com/Лучевой_Алгоритм (дата обращения 7.04.2023)
6. Ларкович С.Н., Евдокимов П.В. C# для UNITY-разработчиков. Практическое руководство по созданию игр – Санкт-Петербург: Изд-во Наука и Техника, 2023. – 368 с.
7. Гниденко И.Г., Павлов Ф.Ф., Федоров Д.Ю. Технологии и методы программирования – Москва: Юрайт, 2022. – 236 с.
8. Эндрю Хант, Дэвид Томас. Программист-прагматик: 2-е юбилейное издание. Пер. с англ. – Санкт-Петербург: ООО «Диалектика», 2020. – 368 с.
9. Мартин Р. Чистый код: создание, анализ и рефакторинг. – Санкт-Петербург: Изд-во Питер, 2019. – 464 с.
10. Бонд Джереми Гибсон. Unity и C#. Геймдев от идеи до реализации. 2-е изд. – Санкт-Петербург: Изд-во Питер, 2019. – 928 с.

Приложение

Функция StartAlg, вызываемая при нажатии кнопки «Сгенерировать»:

```
public void StartAlg()
   {
      if(start == true)
      {
         maze = new int[mazeWidth, mazeHeight];
         roomCenters = new List<Vector2Int>();
         instantiatedObjects = new List<GameObject>();
         instantiatedPath = new List<GameObject>();
         instantiatedDanger = new List<GameObject>();

         GenerateMaze();
         GenerateLava();
         GenerateHotRock();
         while (WaterCount != 0)
         {
         GenerateWater();
         }
         while (PitCount != 0)
         {
            GeneratePit();
         }
         //стены
         GenerateUp();
         GenerateDown();

         GenerateAB();
         InstantiateMaze();
      }
```

```
        start = false;
    }
```

Функция Clear, вызываемая при нажатии кнопки «Очистить»:

```
    public void Clear()
    {
        start = true;
        if(instantiatedObjects.Count != 0)
        {
            foreach (GameObject obj in instantiatedObjects)
            {
                Destroy(obj);
            }
            // Очистка списка после удаления объектов
        }
        if (instantiatedPath.Count != 0)
        {
            foreach (GameObject obj in instantiatedPath)
            {
                Destroy(obj);
            }
            // Очистка списка после удаления объектов
        }
        if (instantiatedDanger.Count != 0)
        {
            foreach (GameObject obj in instantiatedDanger)
            {
                Destroy(obj);
            }
        }
```

```
        instantiatedObjects.Clear();
        instantiatedPath.Clear();
        instantiatedDanger.Clear();
        roomCenters.Clear();

        mazeWidth = 40;
        mazeHeight = 25;
        minRoomSize = 6;
        maxRoomSize = 10;

        lavaCount = 8;
        WaterCount = 5;
        PitCount = 4;
        ACount = 1;
        BCount = 1;

        inputFieldW.text = "" + mazeWidth;
        inputFieldH.text = "" + mazeHeight;
        inputFieldL.text = "" + lavaCount;
        inputFieldWater.text = "" + WaterCount;
        inputFieldPit.text = "" + PitCount;
    }
```

Функция ChangeW, вызываемая при вводе значение в поле ввода «Ширина поля». Аналогично выполняются и остальные функции, изменяющие базовые параметры генерации:

```
    public void ChangeW()
    {
        string enteredText = inputFieldW.text;
        Debug.Log(enteredText);
```

```
        int number;
        bool success = int.TryParse(enteredText, out number);
        if (success)
        {
            Debug.Log("Parsed number: " + number);
            mazeWidth = number;
        }
        else
        {
            Debug.Log("Failed to parse number.");
        }
    }
```

Функция GenerateLava, вызываемая в функции StartAlg. Аналогично выполняются и другие функции по генерации особых клеток на поле:

```
    void GenerateLava()
    {
        bool lava;
        for (int x = 0; x < mazeWidth; x++)
        {
            for (int y = 0; y < mazeHeight; y++)
            {
                if ((maze[x, y] == 0) && (lavaCount != 0) && (maze[x + 1, y] == 0) && (maze[x, y + 1] == 0) && (maze[x - 1, y] == 0) && (maze[x, y - 1] == 0) && (maze[x - 1, y - 1] == 0) && (maze[x + 1, y + 1] == 0))  // если пол
                {
                    lava = Random.value < 0.02f;
                    if (lava == true)
                    {
```

```
                maze[x, y] = 2;
                lavaCount--;
            }
        }
    }
}
```

Функция GenerateAB, вызываемая в функции StartAlg:

```
void GenerateAB()
{
    bool A;
    bool B;
    for (int x = 0; x < mazeWidth; x++)
    {
        for (int y = 0; y < mazeHeight; y++)
        {
            if ((maze[x, y] == 0) && (ACount != 0))  // если пол
            {
                maze[x, y] = 8;
                if (maze[x, y] == 8)
                {
                    A = true;
                    ACount--;
                }
            }
        }
    }
    for (int x = mazeWidth - 1; x >= 0; x--)
    {
        for (int y = mazeHeight - 1; y >= 0; y--)
```

```
        {
            if ((maze[x, y] == 0) && (BCount != 0))  // если пол
            {
                maze[x, y] = 9;
                if (maze[x, y] == 9)
                {
                    B = true;
                    BCount--;
                }
            }
        }
    }
}
```

Функция GenerateHotRock, вызываемая в функции StartAlg:

```
void GenerateHotRock()
{
    bool HOT = false;
    for (int x = 0; x < mazeWidth; x++)
    {
        for (int y = 0; y < mazeHeight; y++)
        {
            if (maze[x, y] == 2)  // если лава
            {
                maze[x + 1, y + 1] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }
                maze[x + 1, y] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }
                maze[x + 1, y - 1] = 3;
```

```
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }

                maze[x, y + 1] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }

                maze[x, y - 1] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }

                maze[x - 1, y + 1] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }

                maze[x - 1, y] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }

                maze[x - 1, y - 1] = 3;
                if (maze[x + 1, y + 1] == 3)
                { HOT = true; }
            }
        }
    }
}
```

Функция GenerateHotRock, вызываемая в функции StartAlg:

```
void GenerateMaze()
{
    for (int x = 0; x < mazeWidth; x++)
```

```
        {
            for (int y = 0; y < mazeHeight; y++)
            {
                maze[x, y] = 1; // Установка всех ячеек в значение true (стена)
            }
        }

        GenerateRooms();
        GenerateRoomPaths(); // Генерация путей между комнатами
    }
```

Функция GenerateRooms, вызываемая в функции GenerateMaze:

```
    void GenerateRooms()
    {
        int numRooms = Random.Range(6, 10);
        for (int i = 0; i < numRooms; i++)
        {
            int roomWidth = Random.Range(minRoomSize, maxRoomSize + 1);
            int roomHeight = Random.Range(minRoomSize, maxRoomSize + 1);

            int roomX = Random.Range(1, mazeWidth - roomWidth - 1);
            int roomY = Random.Range(1, mazeHeight - roomHeight - 1);
            for (int x1 = 1; x1 < mazeWidth - 1; x1++)
            {
                for (int y1 = 1; y1 < mazeHeight - 1; y1++)
                {
                    if ((maze[x1, y1] == 1) && (maze[x1, y1 + 1] == 1) && (maze[x1, y1 - 1]
== 1) && (maze[x1 + 1, y1] == 1) && (maze[x1 - 1, y1] == 1))
                    {
                        // Создание комнаты
```

```
                for (int x = roomX; x < roomX + roomWidth; x++)
                {
                    for (int y = roomY; y < roomY + roomHeight; y++)
                    {
                        maze[x, y] = 0;
                    }
                }
            }
        }
        // Определение центра комнаты
        int centerX = roomX + roomWidth / 2;
        int centerY = roomY + roomHeight / 2;
        roomCenters.Add(new Vector2Int(centerX, centerY));
    }
}
```

Функция GenerateRoomPaths, вызываемая в функции GenerateMaze:

```
void GenerateRoomPaths()
{
    // Генерация путей между комнатами
    for (int i = 0; i < roomCenters.Count - 1; i++)
    {
        Vector2Int startRoom = roomCenters[i];
        Vector2Int endRoom = roomCenters[i + 1];

        int startX = startRoom.x;
        int startY = startRoom.y;
        int endX = endRoom.x;
        int endY = endRoom.y;
```

```
        while (startX != endX || startY != endY)
        {
            if (startX < endX)
            {
                startX++;
            }
            else if (startX > endX)
            {
                startX--;
            }
            if (startY < endY)
            {
                startY++;
            }
            else if (startY > endY)
            {
                startY--;
            }
            maze[startX, startY] = 0; // Установка ячейки в значение false (проход)
            maze[startX + 1, startY] = 0;
        }
    }
}
```

Функция StartLee, вызываемая при нажатии кнопки «Проложить путь»:

```
public void StartLee()
{
    int startX = 0;
    int startY = 0;
    int finishX = 0;
```

```
int finishY = 0;
string stroka = "";
int o;
for (int y = mazeHeight - 1; y >= 0; y--)
{
   stroka += "\n";
   for (int x = 0; x < mazeWidth; x++)
   {
      o = maze[x, y];
      stroka += " " + o + " ";
      if (maze[x, y] == 8)
      {
         startX = x;
         startY = y;
      }
      if (maze[x, y] == 9)
      {
         finishX = x;
         finishY = y;
      }
   }
}
Debug.Log(stroka);
int[,] wave = new int[mazeWidth, mazeHeight];
for (int x = 0; x < mazeWidth; x++)
{
   for (int y = 0; y < mazeHeight; y++)
   {
      wave[x, y] = 0;
   }
```

```
}
int number = 1;
wave[startX, startY] = number; //++number
string strokaW = "";  //для дебага в будущем
bool end = false;
while (end == false)
{
   for (int x = 0; x < mazeWidth; x++)
   {
      for (int y = 0; y < mazeHeight; y++)
      {
         if (wave[x, y] == number)
         {
            if (maze[x + 1, y] == 9)
            {
               wave[x + 1, y] = number + 1;
               end = true;
            }
            if (maze[x - 1, y] == 9)
            {
               wave[x - 1, y] = number + 1;
               end = true;
            }
            if (maze[x, y - 1] == 9)
            {
               wave[x, y - 1] = number + 1;
               end = true;
            }
            if (maze[x, y + 1] == 9)
            {
```

```
                    wave[x, y + 1] = number + 1;
                    end = true;
                }
                if (x > 0 && wave[x + 1, y] == 0 && maze[x + 1, y] == 0)
                {
                    wave[x + 1, y] = number + 1;
                }
                if (x < mazeWidth - 1 && wave[x - 1, y] == 0 && maze[x - 1, y] == 0)
                {
                    wave[x - 1, y] = number + 1;
                }
                if (y > 0 && wave[x, y - 1] == 0 && maze[x, y - 1] == 0)
                {
                    wave[x, y - 1] = number + 1;
                }
                if (y < mazeHeight - 1 && wave[x, y + 1] == 0 && maze[x, y + 1] == 0)
                {
                    wave[x, y + 1] = number + 1;
                }
            }
        }
    }
    if (number > mazeHeight * mazeWidth)  //если значение уже стало больше всех клеток в карте, то пути точно нет
    {
        Debug.Log("ПУТИ НЕТ");
        end = true;
    }
    number++;
}
```

```
strokaW = "";
Debug.Log("РАСПРОСТРАНЕНИЕ");
for (int yy = mazeHeight - 1; yy >= 0; yy--)
{
   strokaW += "\n";
   for (int xx = 0; xx < mazeWidth; xx++)
   {
      o = wave[xx, yy];
      strokaW += " " + o + " ";
   }
}
Debug.Log(strokaW);
Debug.Log(wave[finishX, finishY]);
Debug.Log("Более наглядный");

int currX = finishX;
int currY = finishY;
int[,] path = new int[mazeWidth, mazeHeight];
int pathLength = wave[currX, currY];  // координаты сундука, т.к. при распространении волны волна дошла до туда с таким числом равным расстоянию

while (pathLength > 0)
{
   path[currX, currY] = pathLength--;
   if (currX > 0 && wave[currX - 1, currY] == pathLength) currX--;//приоритет
   else if (currY > 0 && wave[currX, currY - 1] == pathLength) currY--;
   else if (currY > 0 && wave[currX, currY + 1] == pathLength) currY++;
   else if (currX < mazeWidth - 1 && wave[currX + 1, currY] == pathLength) currX++;
}
```

```
    // вывод массив с путем
    strokaW = "";
    Debug.Log("ПУТЬ");
    for (int yy = mazeHeight - 1; yy >= 0; yy--)
    {
       strokaW += "\n";
       for (int xx = 0; xx < mazeWidth; xx++)
       {
          o = path[xx, yy];
          strokaW += " " + o + " ";
       }
    }
    Debug.Log(strokaW);

    for (int x = 0; x < mazeWidth; x++)
    {
       for (int y = 0; y < mazeHeight; y++)
       {
          if (path[x, y] != 0)
          {
             GameObject instdPath = Instantiate(pathPrefab, new Vector3(x, y, -2), Quaternion.identity);
             instantiatedPath.Add(instdPath); //добавляем в список (со списка будет работать удаление)
          }
       }
    }
 }
```