**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

(СПбГУТ)

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ И СИСТЕМ **(ИКСС)**

КАФЕДРА ПРОГРАММНОЙ ИНЖЕНЕРИИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ **(ПИиВТ)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Проект по дисциплине Алгоритмы и структуры данных

«Генерация и прохождение лабиринтов»

Выполнил:

Студент группы ИКПИ-15

Баранов С.Н.

Студент группы ИКПИ-15

Смирнов С.В.

Студент группы ИКПИ-15

Тарасова Д.В.

Студент группы ИКПИ-15

Целоусов Г.А.

Преподаватель:

Дагаев А.В.

**Санкт-Петербург**

**2023 г.**

Оглавление

[Введение 3](#_heading=h.gjdgxs)

[1.1. Общая информация 4](#_heading=h.30j0zll)

[1.1.1. Постановка задачи и особенности среды разработки 4](#_heading=h.1fob9te)

[Постановка задачи: 4](#_heading=h.3znysh7)

[Выбор среды разработки: 4](#_heading=h.2et92p0)

[1.1.2. Характеристика языка программирования 5](#_heading=h.tyjcwt)

[1.2. Генерация лабиринта 6](#_heading=h.3dy6vkm)

[1.2.1. Сравнительная характеристика алгоритмов генерации 6](#_heading=h.1t3h5sf)

[1.2.2. Обоснование выбора алгоритма 9](#_heading=h.4d34og8)

[1.2.3. Принцип действия алгоритмов 10](#_heading=h.2s8eyo1)

[Recursive Backtracking: 10](#_heading=h.17dp8vu)

[1.3. Прохождение лабиринта 11](#_heading=h.3rdcrjn)

[1.3.1. Алгоритм Ли (волновой алгоритм) 11](#_heading=h.26in1rg)

[1.3.2. Алгоритм следования вдоль стены 11](#_heading=h.lnxbz9)

[**2.1. Разработка лабиринта в графической среде 12**](#_heading=)

[Разработка структуры лабиринта 13](#_heading=)

[**2.2. Разработка механизмов поиска пути в лабиринте 14**](#_heading=h.pj4bv2af7efa)

[Разработка структуры поиска и отображения пути 14](#_heading=h.5eqamcgf0zp2)

[**2.3. Описание классов, методов и функций приложения 14**](#_heading=h.ge3ezr8cyft9)

# Введение

Данная работа посвящена генерации лабиринтов и поиску маршрутов выхода из них. В данном руководстве содержится вся необходимая информация касательно реализации алгоритмов построения и прохождения лабиринтов, а также вспомогательная информация касательно задействованных при разработке технологий.

# Общая информация

## Постановка задачи и особенности среды разработки

### Постановка задачи:

В ходе данной работы необходимо достигнуть следующих целей:

1. Изучить алгоритмы генерации двумерных лабиринтов
2. Изучить алгоритмы построения маршрута выхода из лабиринтов
3. Изучить способы реализации многопоточного приложения
4. Реализовать построение лабиринта в графической среде
5. Реализовать построение маршрута выхода из лабиринта в графической среде

### Выбор среды разработки:

В качестве графической среды для построения лабиринта был выбран игровой движок Unity. Выбор обусловлен следующими преимуществами:

1. Доступность. Среда разработки Unity hub является кроссплатформенным бесплатным приложением, что облегчает взаимодействие с проектом.
2. Гибкость. Данный движок обладает множеством встроенных утилит и инструментов, что позволяет упростить использование приложения.
3. Наличие визуальной среды разработки. Наличие таких возможностей, как интегрированная среда разработки и цепочка сборки, позволяет увеличить производительность разработчиков.
4. Модульная система компонентов. Организация работы в виде объединения функциональных блоков, а не наследования, упрощает работу с объектами.

В качестве среды разработки программного кода была выбрана Microsoft Visual Studio, ввиду следующих преимуществ:

1. Гибкость. За счет использования модульных языковых пакетов можно осуществить узкоспециализированную разработку на конкретных языках программирования.
2. Интегрируемость с Unity. Из-за наличия специализированных модулей языка программирования C# данная среда разработки позволяет осуществлять прямое редактирования файлов скриптов Unity.

## Характеристика языка программирования

В качестве языка программирования был выбран язык C#. Далее приведён ряд преимуществ его использования:

1. Использование в среде разработки Unity. C# является основным языком разработки на движке Unity.
2. C-подобный синтаксис. Схожесть данного языка программирования с языками C и C++ упрощает разработку на нем.
3. Возможность использования объектно-ориентированного подхода.
4. Независимость от аппаратного функционала.

# Генерация лабиринта

## Сравнительная характеристика алгоритмов генерации

Идеальный лабиринт – лабиринт без петель и без недостижимых областей; из каждой точки существует только один путь к каждой другой точке в лабиринте.

В таблице 1 представлена сравнительная характеристика ряда наиболее распространенных алгоритмов генерации лабиринтов.

*Таблица 1. Сравнительная характеристика алгоритмов генерации лабиринтов*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Алгоритм** | **% тупиков** | **Тип** | **Приоритет** | **Смещенность** | **Однородность** | **Память** |
| одномаршрутный | 0 | дерево | стены | Нет | никогда | N^2 |
| Recursive backtracking | 10 | дерево | проходы | Нет | никогда | N^2 |
| рекурсивное деление | 23 | дерево | стены | Нет | никогда | N |
| двоичное дерево | 25 | множество | оба | Да | никогда | 0 |
| алгоритм Эллера | 28 | множество | оба | Да | нет | N |
| Алгоритм Олдоса-Бродера | 29 | дерево | оба | Нет | да | 0 |
| Алгоритм Краскала | 30 | множество | оба | Нет | нет | N^2 |
| Алгоритм Прима | 32 | дерево | оба | Нет | нет | N^2 |

**Тупики.**Это приблизительный процент ячеек, являющихся тупиками в лабиринте. При достаточно высоком показателе проходов количество тупиков Recursive Backtracking может становиться ниже 1%. Наибольший вероятный процент тупиков в двухмерном ортогональном идеальном лабиринте равен 66% — это одномаршрутный проход с кучей тупиков единичной длины по обеим сторонам от него.

**Тип.**Существует два типа алгоритмов создания идеальных лабиринтов:

* Алгоритм на основе дерева выращивает лабиринт подобно дереву, всегда добавляя к тому, что уже есть, и на каждом этапе имея правильный идеальный лабиринт.
* Алгоритм на основе множеств выполняет построения там, где ему хочется, отслеживая части лабиринта, соединённые друг с другом, чтобы соединить всё и создать правильный лабиринт на момент завершения.

**Приоритет.**Большинство алгоритмов можно реализовать или как вырезание проходов, или как добавление стен. Очень немногие можно реализовать только как один или другой подход. В одномаршрутных лабиринтах всегда используется добавление стен, потому что в них задействуется разбиение проходов стенами на две части, однако базовый лабиринт можно создать любым способом. Recursive Backtracking нельзя реализовать с добавлением стен, потому что в этом случае он склонен создавать путь решения, который следует вдоль внешнего края, где вся внутренняя часть лабиринта соединена с границей единственным проходом. Рекурсивное деление нельзя использовать для вырезания проходов, потому что это приводит к созданию очевидного решения, которое или следует вдоль внешнего края, или иначе напрямую пересекает внутреннюю часть.

**Отсутствие смещенности.**Одинаково ли воспринимает алгоритм все направления и стороны лабиринта так, что последующий анализ лабиринта не может обнаружить никакой смещенности проходов. Алгоритм двоичного дерева чрезвычайно смещён, в нём легко перемещаться в один угол и сложно в противоположный. Алгоритм Эллера склонен к созданию проходов, приблизительно синхронизируя начальные или конечные края.

**Однородность**. Генерирует ли алгоритм все возможные лабиринты с равной вероятностью.

* «Да» означает, что алгоритм полностью однороден.
* «Нет» означает, что алгоритм потенциально может генерировать все возможные лабиринты в пределах любого пространства, но не с равной вероятностью.
* «Никогда» означает, что существуют возможные лабиринты, которые алгоритм никогда не сможет сгенерировать. Учтите, что только алгоритмы с полным отсутствием смещенности могут быть полностью однородными.

**Память**. Объём дополнительной памяти или стека, необходимый для реализации алгоритма. Эффективные алгоритмы требуют только битовой карты самого лабиринта, в то время как другие требуют объёма памяти, пропорционального одной строке (N), или пропорционального количеству ячеек (N2). Некоторым алгоритмам даже не нужно иметь в памяти весь лабиринт, и они могут добавлять части лабиринта бесконечно. Алгоритму Эллера нужен объём памяти для хранения строки, но большего ему не требуется, потому что достаточно хранить только одну строку лабиринта. Для рекурсивного деления требуется стек объёмом вплоть до размера строки, но ему не нужно смотреть на битовую карту лабиринта.

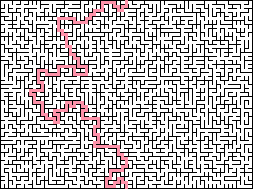
## Обоснование выбора алгоритма

Методом сравнительного анализа различных алгоритмов генерации лабиринтов был выбран алгоритм Recursive backtracking, так как он гарантирует невысокий процент тупиков и не генерирует слишком простые лабиринты.

## Принцип действия алгоритмов

### Recursive Backtracking:

Алгоритм требует стека, объём которого может доходить до размеров лабиринта. При вырезании он ведёт себя максимально жадно, и всегда вырезает проход в несозданной части, если она существует рядом с текущей ячейкой. Каждый раз, когда мы перемещаемся к новой ячейке, записываем предыдущую ячейку в стек. Если рядом с текущей позицией нет несозданных ячеек, то извлекаем из стека предыдущую позицию. Лабиринт завершён, когда в стеке больше ничего не остаётся. Это приводит к созданию лабиринтов с максимальным показателем текучести, тупиков меньше, но они длиннее, а решение обычно оказывается очень долгим и извилистым. При правильной реализации он выполняется быстро, и быстрее работают только очень специализированные алгоритмы. Recursive backtracking не может работать с добавлением стен, потому что обычно приводит к пути решения, следующему по внешнему краю, когда вся внутренняя часть лабиринта соединена с границей одним проходом.



*Рисунок 1 - Лабиринт, сгенерированный методом Recursive Backtracking*

# Прохождение лабиринта

## Алгоритм Ли (волновой алгоритм)

Волновой алгоритм - это алгоритм, который позволяет вам найти минимальный путь на графе. В основе этого метода лежит алгоритм поиска по ширине. Алгоритм предназначен для поиска кратчайшего пути от стартовой ячейки к конечной ячейке, если это возможно, либо, при отсутствии пути, выдать сообщение о непроходимости.

Работа алгоритма включает в себя три этапа: инициализацию, распространение волны и восстановление пути.

Во время инициализации строится образ множества ячеек обрабатываемого поля, каждой ячейке приписываются атрибуты проходимости/непроходимости, запоминаются стартовая и финишная ячейки. Далее, от стартовой ячейки порождается шаг в соседнюю ячейку, при этом проверяется, проходима ли она, и не принадлежит ли ранее меченной в пути ячейке. Соседние ячейки принято классифицировать двояко: в смысле окрестности Мура и окрестности фон Неймана, отличающийся тем, что в окрестности фон Неймана соседними ячейками считаются только 4 ячейки по вертикали и горизонтали, в окрестности Мура — все 8 ячеек, включая диагональные. В данном проекте используется окрестность фон Неймана.

При выполнении условий проходимости и непринадлежности её к ранее помеченным в пути ячейкам, в атрибут ячейки записывается число, равное количеству шагов от стартовой ячейки, на первом шаге это будет 1. Каждая ячейка, меченная числом шагов от стартовой ячейки, становится стартовой и из неё порождаются очередные шаги в соседние ячейки. Очевидно, что при таком переборе будет найден путь от начальной ячейки к конечной, либо очередной шаг из любой порождённой в пути ячейки будет невозможен.

Восстановление кратчайшего пути происходит в обратном направлении: при выборе ячейки от финишной ячейки к стартовой на каждом шаге выбирается ячейка, имеющая атрибут расстояния от стартовой на единицу меньше текущей ячейки. Очевидно, что таким образом находится кратчайший путь между парой заданных ячеек.

## Алгоритм следования вдоль стены

Одним из наиболее широко известных методов поиска решения лабиринтов является метод следования вдоль стены, также известный как “правило левой/правой руки”. В данном проекте было выбрано следование вдоль правой стены. Этот метод основан на внешней связности лабиринта — все стены должны быть соединены с внешней границей лабиринта. Если это так, то всегда можно найти выход из лабиринта, непрерывно следуя либо по левой, либо по правой стороне на протяжении всего лабиринта. Однако в тех случаях, когда не все стены соединены с внешними границами, этот метод не всегда будет находить решение. Этот метод/алгоритм полезен, если нам известно, что стены лабиринта связаны между собой. Алгоритм очень эффективен, так как не требует маркировки развилок или перезапуска при неудачных попытках; алгоритм просто следует по левой или правой стороне лабиринта.

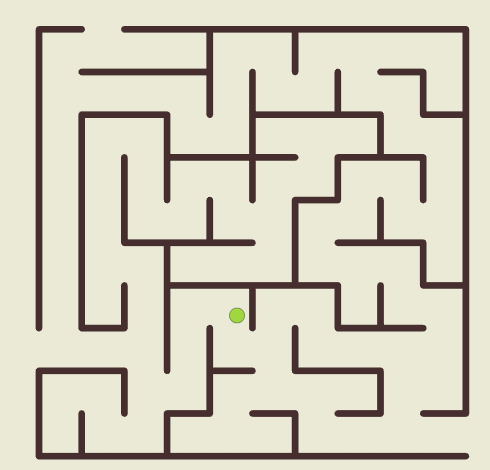
# 2.1. Разработка лабиринта в графической среде

## Разработка структуры лабиринта

Лабиринт является множеством клеток. Клетка - наименьшая структурная единица лабиринта. В целях оптимального распределения памяти клетка содержит в себе только левую и нижнюю стены.



*Рисунок 2 - Клетка*



*Рисунок 3 - Пример лабиринта*

Лабиринт представлен в виде двумерного массива клеток, с помощью которого осуществляется взаимодействие с отображаемым на экране лабиринтом.

Стартом в лабиринте всегда является левый нижний угол (с координатами (0, 0)). Выходы генерируются случайно, однако один из трех выходов гарантированно является самой удаленной клеткой от старта. Для перемещения по лабиринту реализован объект-игрок, управление которым осуществляется с клавиатуры.

Были разработаны следующие структуры лабиринтов:

* идеальный лабиринт с тупиками (с 3 выходами)
* лабиринт с циклами (с 3 выходами)

# 2.2. Разработка механизмов поиска пути в лабиринте

## Разработка структуры поиска и отображения пути

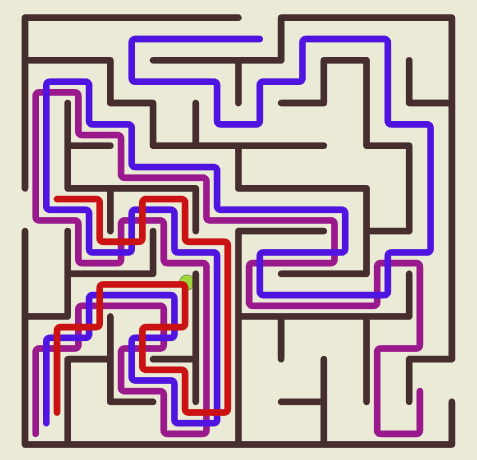
Поиск пути в лабиринте осуществляется соответствующими алгоритмами (см. подробное описание в п. 2.3). Поиск пути к каждому выходу происходит в отдельном потоке.

Для отображения процесса работы алгоритма реализована структура Floor, “закрашивающая” клетки, по которым проходит алгоритм.



*Рисунок 4 - пример “закрашенных” клеток*

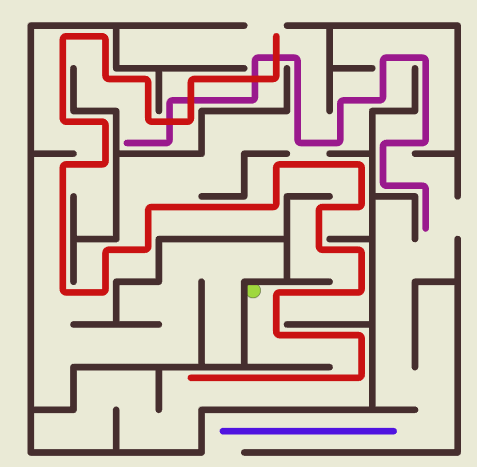
Отображение пути в лабиринте реализовано при помощи встроенного компонента Unity - LineRenderer. Он отображает линию-путь от начала лабиринта до выхода из него, полученную в результате работы алгоритма поиска маршрута. Отрисовка путей происходит параллельно благодаря многопоточности.



*Рисунок 5 - лабиринт с отрисованными линиями-путями*

# 2.3. Реализация многопоточности

Многопоточность реализована с помощью встроенного механизма Unity - Coroutine (сопрограмма). Для его работы все функции поиска пути имеют специальный тип IEnumerator. Благодаря этому механизму параллельно отрисовывается 3 пути, а также реализуются паузы в коде (при помощи ключевого слова yield).



*Рисунок 6 - демонстрация многопоточности (каждая линия отрисовывается в своем потоке)*

# 2.4. Описание классов, методов и функций приложения

Класс Cell : MonoBehaviour - класс, характеризующий клетку как игровой объект.

Атрибуты:

public GameObject WallLeft - левая стенка клетки;

public GameObject WallBottom - нижняя стенка клетки.

Класс MazeGeneratorCell - класс, характеризующий клетку как логический объект.

Атрибуты:

public int X - координата клетки по оси Х;

public int Y - координата клетки по оси У;

public bool WallLeft - логическое представление левой стены клетки;

public bool WallBottom - логическое представление нижней стены клетки;

public bool Visited - информация о том, была ли посещена клетка;

public int DistanceFromStart - расстояние от текущей клетки до старта.

Класс Maze - класс, характеризующий лабиринт

Атрибуты:

public MazeGeneratorCell[,] cells - двумерный массив, состоящий из клеток лабиринта;

public List<Vector2Int> finishPosition = new List<Vector2Int>(3) - список с координатами финишей.

Класс HintRenderer : MonoBehaviour - класс, отвечающий за поиск и отрисовку пути

Атрибуты:

public MazeGenerator MazeSpawner - объект класса MazeGenerator;

private LineRenderer componentLineRenderer - объект класса LineRenderer;

public Floor FloorPrefab - объект класса Floor;

public List<List<Floor>> floors - двумерный список объектов класса Floor;

public static float HintRenderTimeout = 0.1f - константа, отвечающая за время паузы при исполнении кода (для наглядности отрисовки).

Методы:

private void Start() - инициализация объекта класса LineRenderer. Вызывается автоматически при объявлении объекта данного класса;

public void Clear() - удаление ранее отрисованных путей с экрана;

public IEnumerator<object> DrawPath(int x, int y) - отрисовка пути (не является алгоритмом поиска);

public IEnumerator<object> Lee0(int bx, int by) - алгоритм Ли (волновой алгоритм) поиска пути и последующая отрисовка;

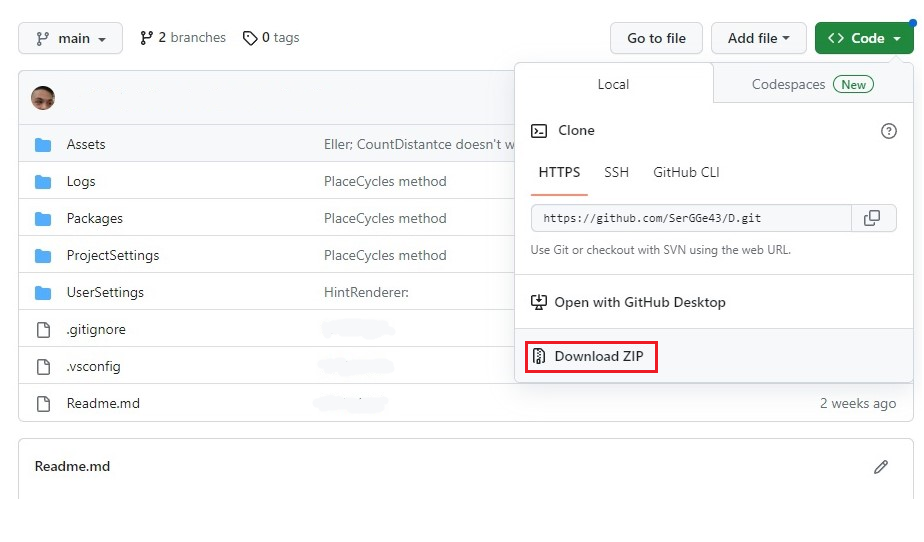
public IEnumerator<object> WallPath(int bx, int by) - алгоритм следования вдоль стены и последующая отрисовка пути.

# 2.5. Руководство пользователя

### Установка и запуск проекта:

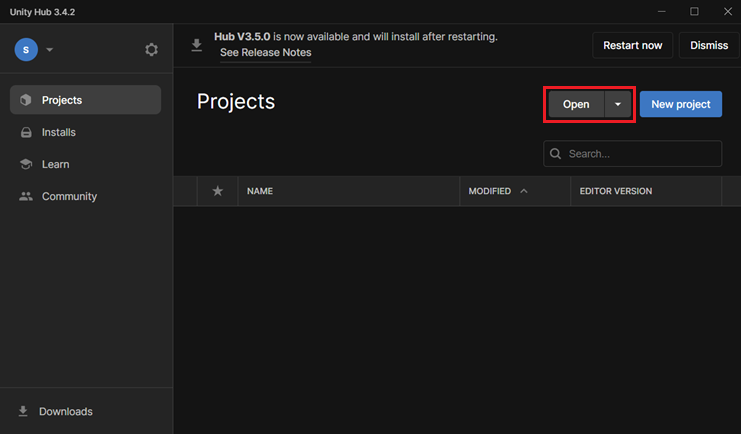
Перед работой с проектом необходимо установить среду разработки Unity. Сделать это можно на официальном сайте, в разделе загрузки (https://unity.com/ru/download). Присутствуют версии для операционных систем Windows, Mac и Linux.

После установки среды разработки необходимо скачать архив, содержащий проект, с платформы GitHub и распаковать его в любое удобное расположение.



*Рисунок 7 – Вид репозитория при скачивании проекта*

Далее необходимо открыть среду Unity и в разделе projects (проекты) выбрать опцию open (открыть).

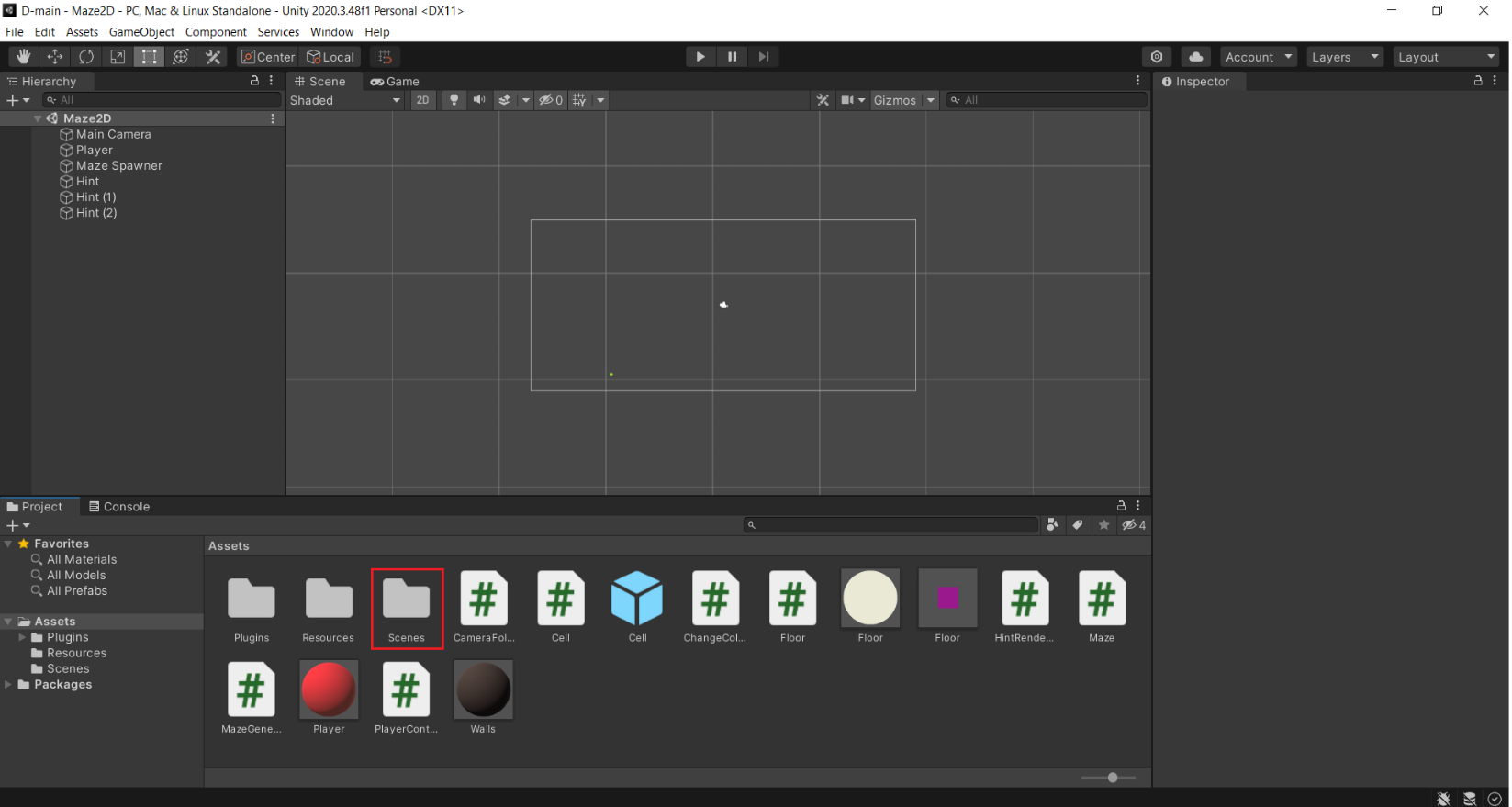


*Рисунок 8 – Окно управления проектами среды Unity*

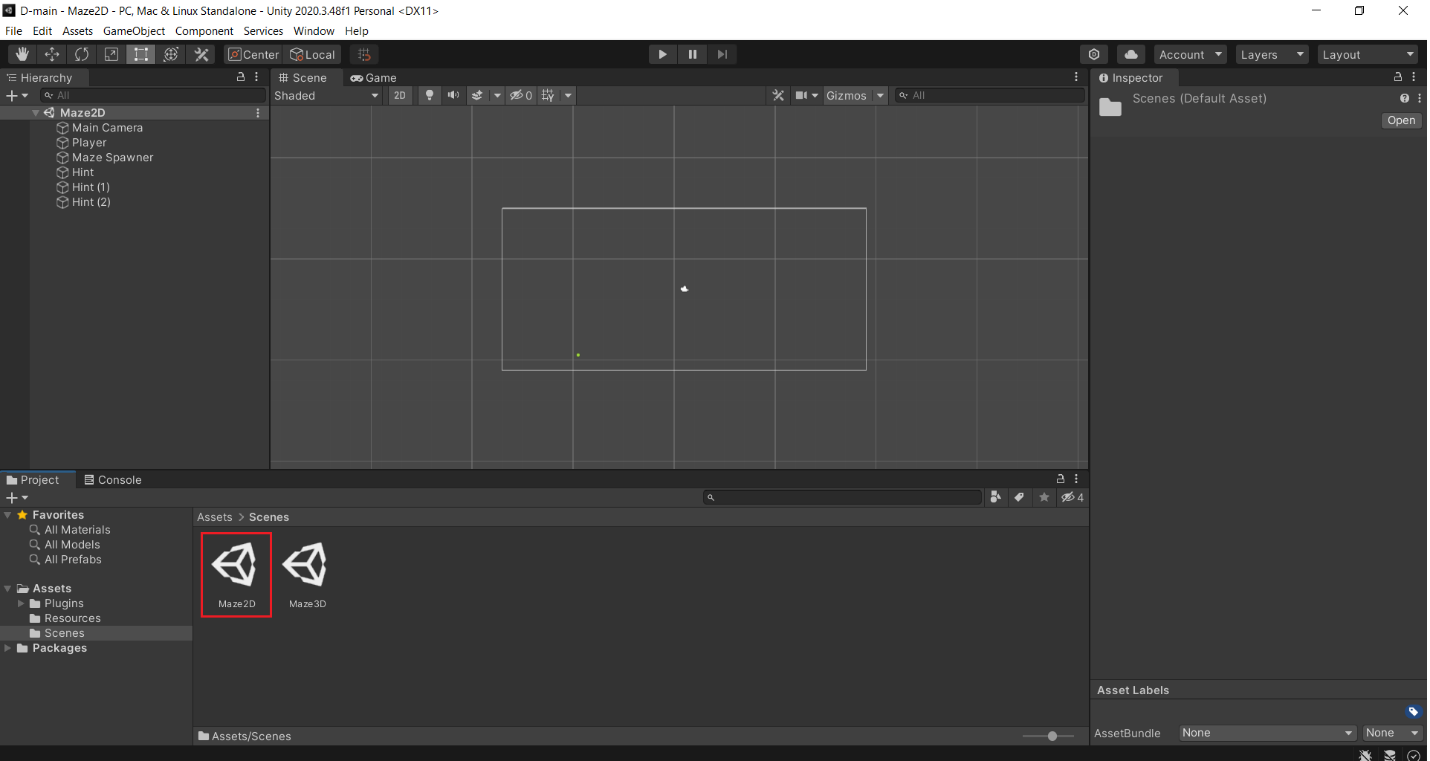
Далее во всплывающем окне необходимо выбрать папку проекта, распакованную из архива и подтвердить действие кнопкой open (открыть).

### Инструкция пользователя:

После запуска проекта необходимо запустить сцену. Для этого переходим папку scenes (сцены) и двойным нажатием открываем сцену Maze2D.

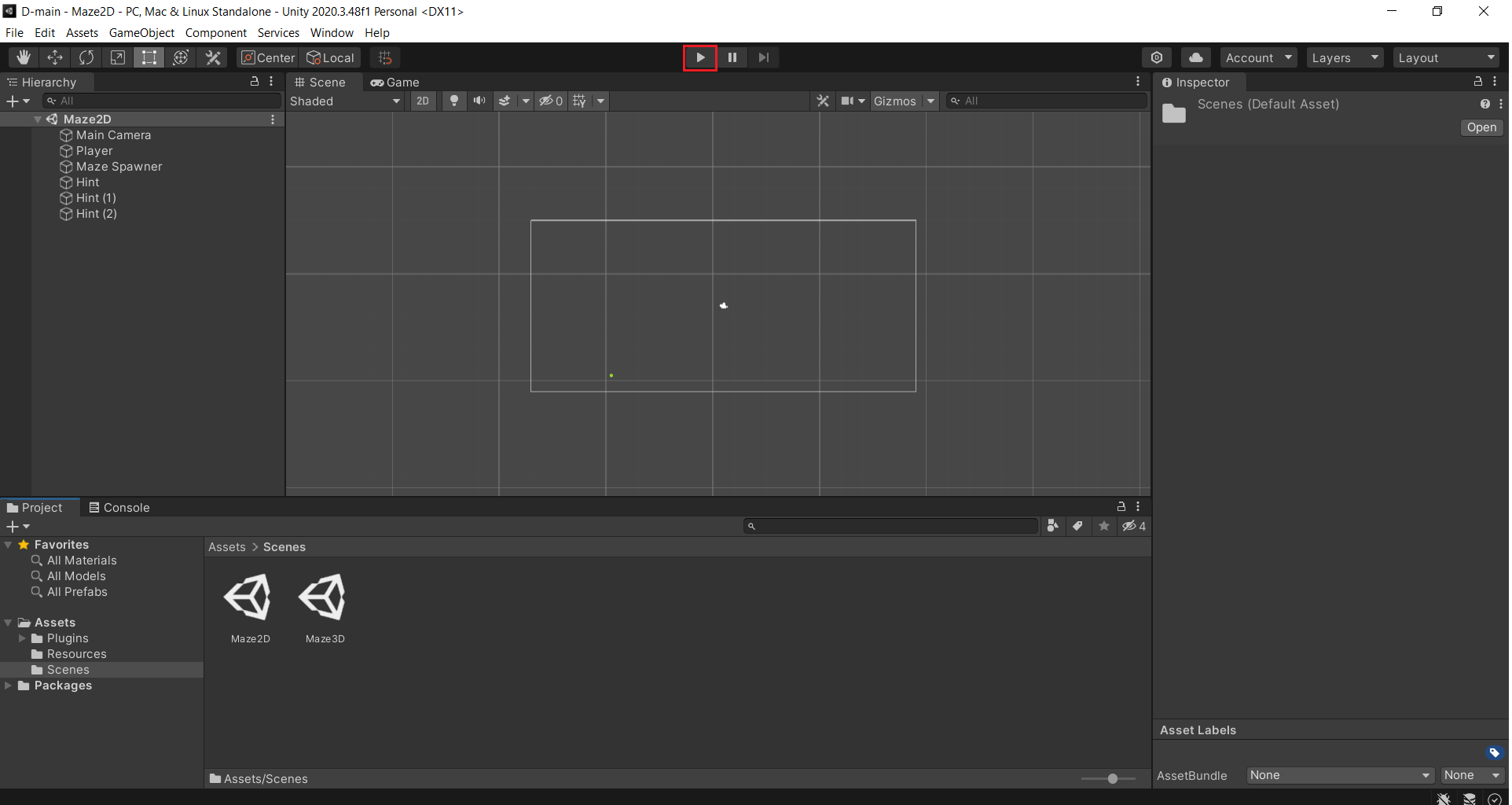


*Рисунок 9 – Выбор папки*

**

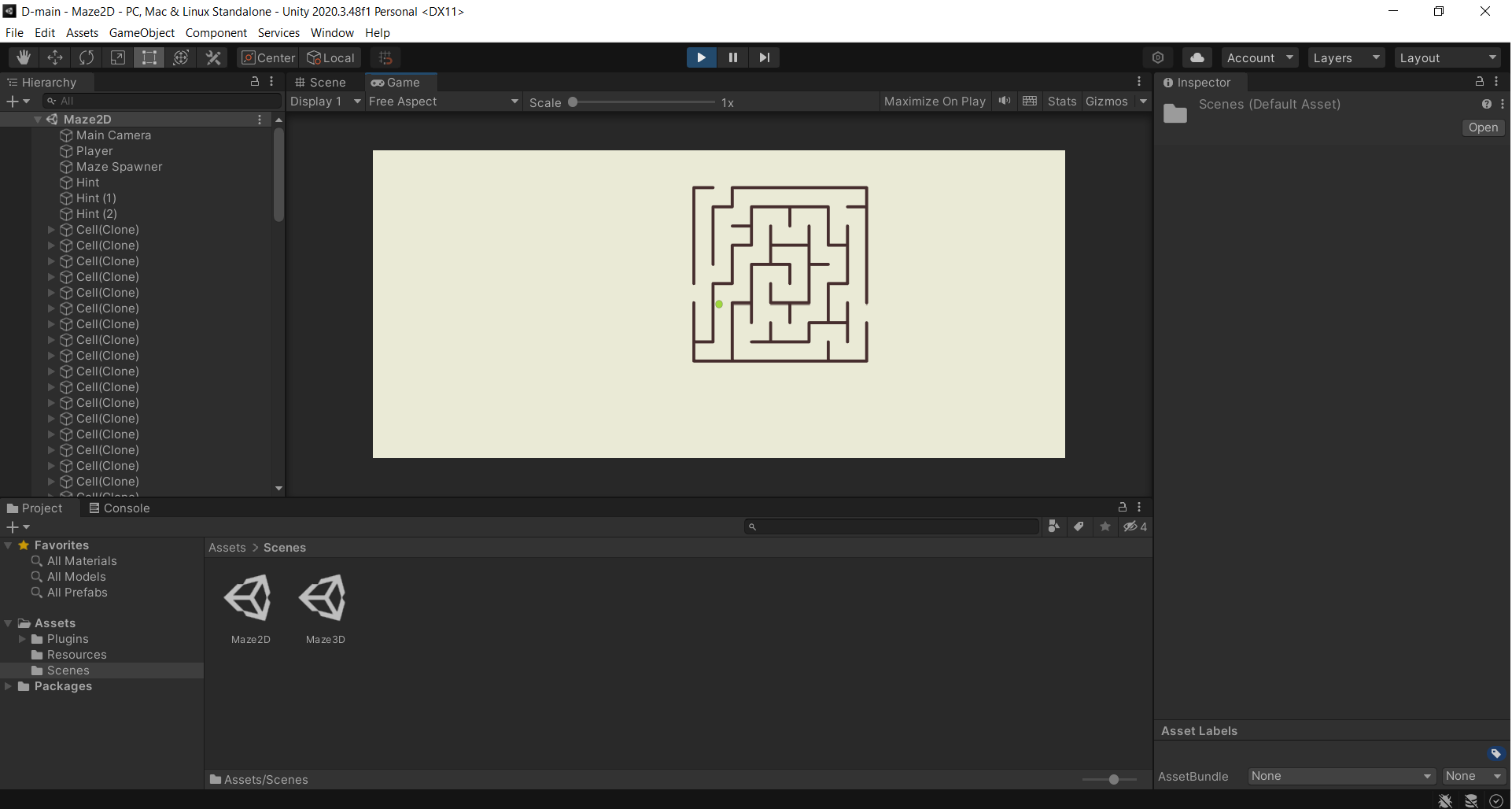
*Рисунок 10 – Выбор сцены*

Далее необходимо запустить выбранную сцену с помощью кнопки в верхней части экрана.



*Рисунок 11 – Запуск сцены*

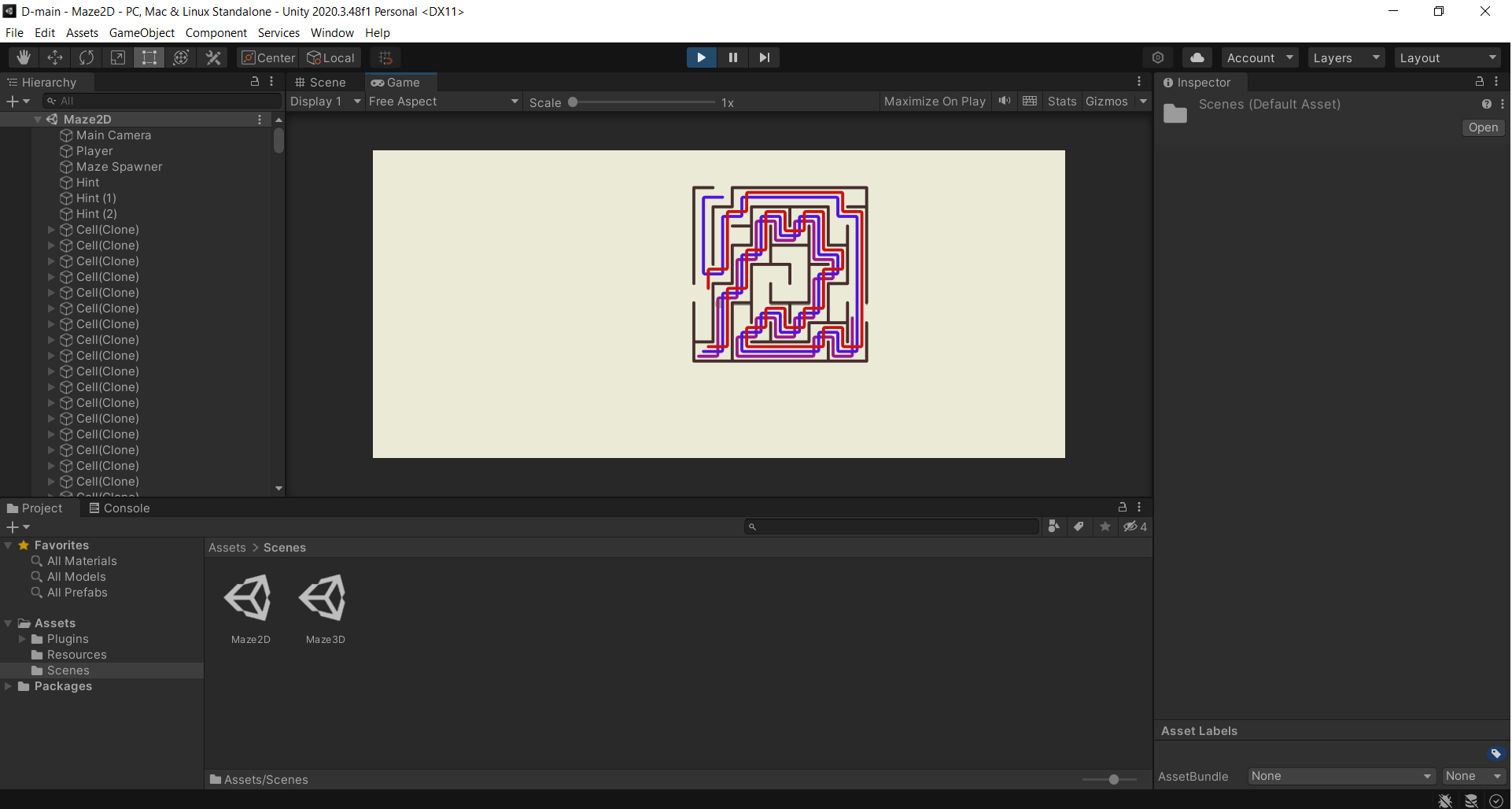
После этого на экране в реальном времени отрисуется лабиринт и появится точка-пользователь.



*Рисунок 12 – Отрисованный лабиринт*

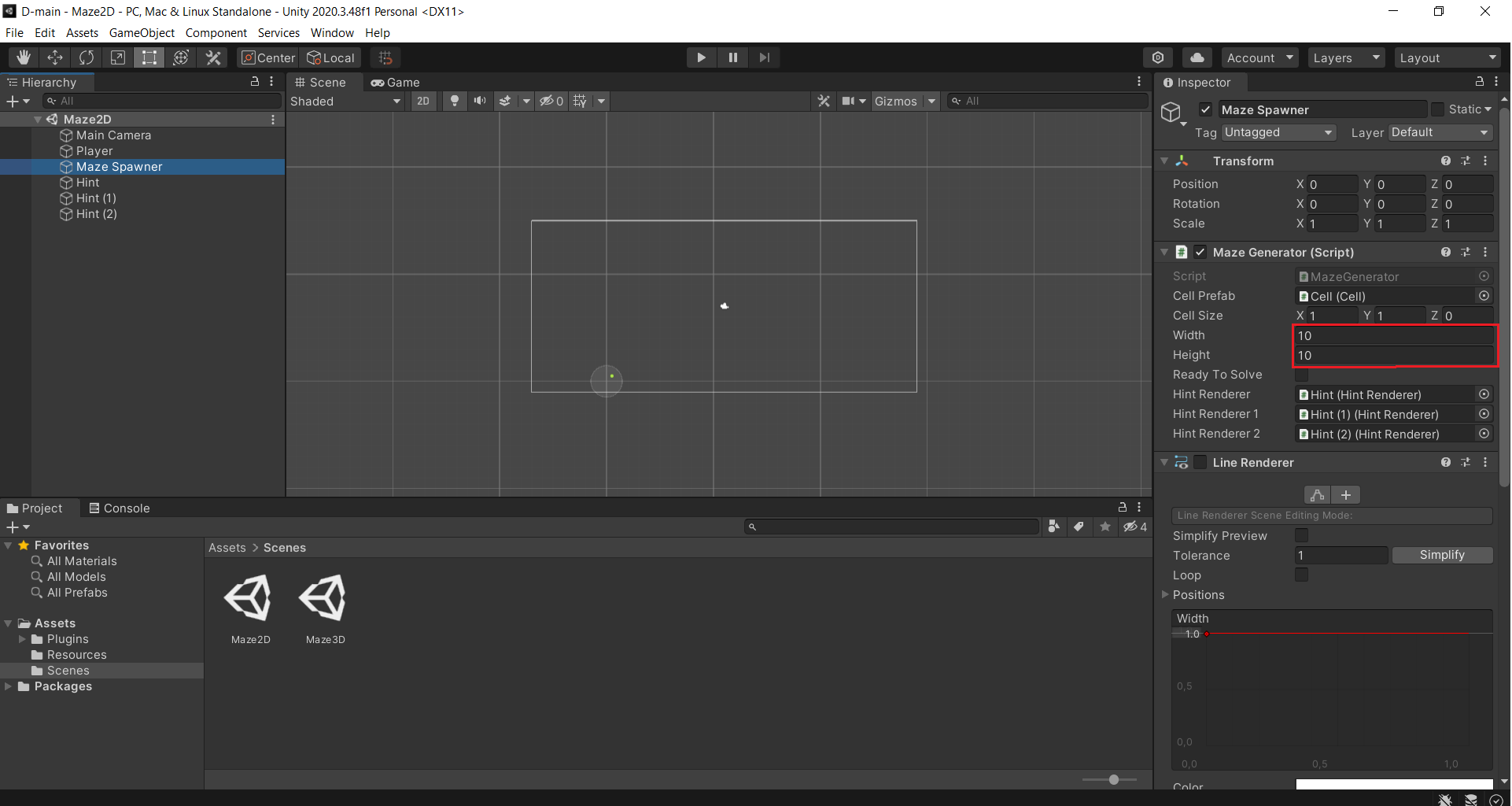
При этом пользователь может перемещаться по лабиринту при помощи клавиш-«стрелочек» на клавиатуре.

Для запуска поиска выхода из лабиринта необходимо нажать клавишу L.



*Рисунок 13 – Лабиринт со сгенерированными маршрутами выхода*

Генерация лабиринтов большего размера возможна путем изменения параметров width и height у объекта Maze Spawner



*Рисунок 14 – изменение размера лабиринта*

# Выводы по проделанной работе

В ходе работы над данным проектом были получены следующие результаты:

1. Изучены и реализованы алгоритмы генерации лабиринтов
2. Освоены методы нахождения выхода из лабиринта
3. Изучены способы работы со средой разработки Unity в сфере визуализации работы алгоритмов
4. Реализованы механизмы многопоточности среды Unity применительно к процедуре поиска выхода из лабиринта
5. Получена оценка «отлично» автоматом за экзамен по дисциплине АиСД