Maze Escape

[**Overview 2**](#_qlinrp33f7eg)

[**Things used in project 2**](#_gybvqcv0abh1)

[**Maze generation 3**](#_6olpemwwqt3m)

[**AI 10**](#_xxnzjvxlq52m)

# Overview

The maze is randomly and procedurally generated, always having a start and a finish. The maze is always passable.

The player appears at the start of the maze, and when the player reaches the finish, he wins.

There are enemy seekers in the maze. Enemies have damage areas. While a player is inside a damage area he gets damage. Player will lose the game when his HP is below zero.

Enemies have patrol paths. Until the enemy sees the player, he follows his patrol path. If the enemy is following the player and loses sight of the player, the enemy will travel to the last seen location of the player, and return to patrol mode if the player is not noticed again.

There are three created scenes that are setted to see all different functionalities (Assets -> MazeEscape -> Scenes):

* SquareMaze - square maze generation
* RandomMaze - random directions maze generation
* ImageMaze - maze generated from image

# Things used in project

Different things that were used for this project:

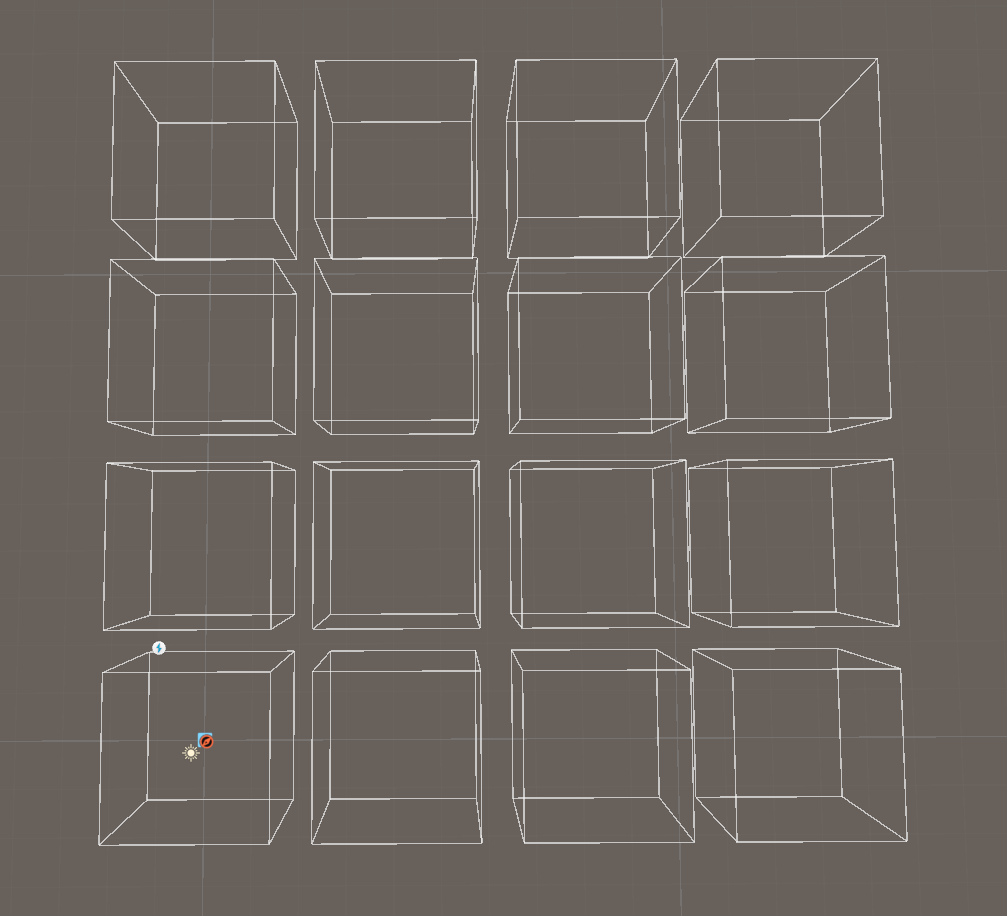
* Namespaces - for organizing code
* Assemblies - good way to separate different systems and to prevent cyclic dependencies
* Disabled domain reloading - for fast play mode loading
* Scriptable objects to store properties - very easy to change settings and can be created many different options, like creating different generation processes for different levels
* Zenject - dependency injection
* LeanTween - UI animations
* "Flood Fill" algorithm - for maze validation (<https://en.wikipedia.org/wiki/Flood_fill>)
* AI agent navigation

# Maze generation

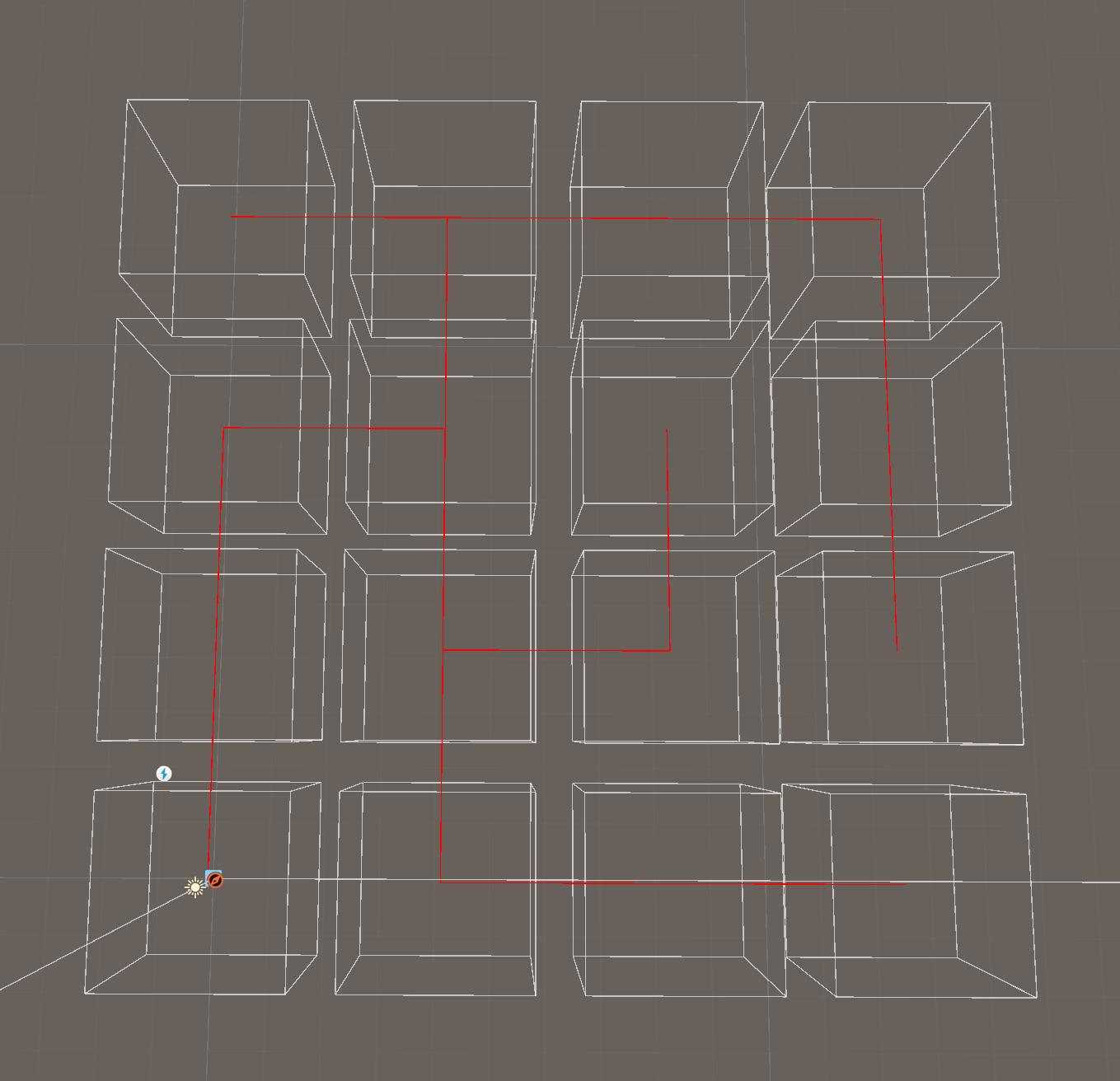
Maze generation consists of *Generation Processes*. Each process takes turns to generate the part that it is responsible for and then the next process starts.

Full generation flow:

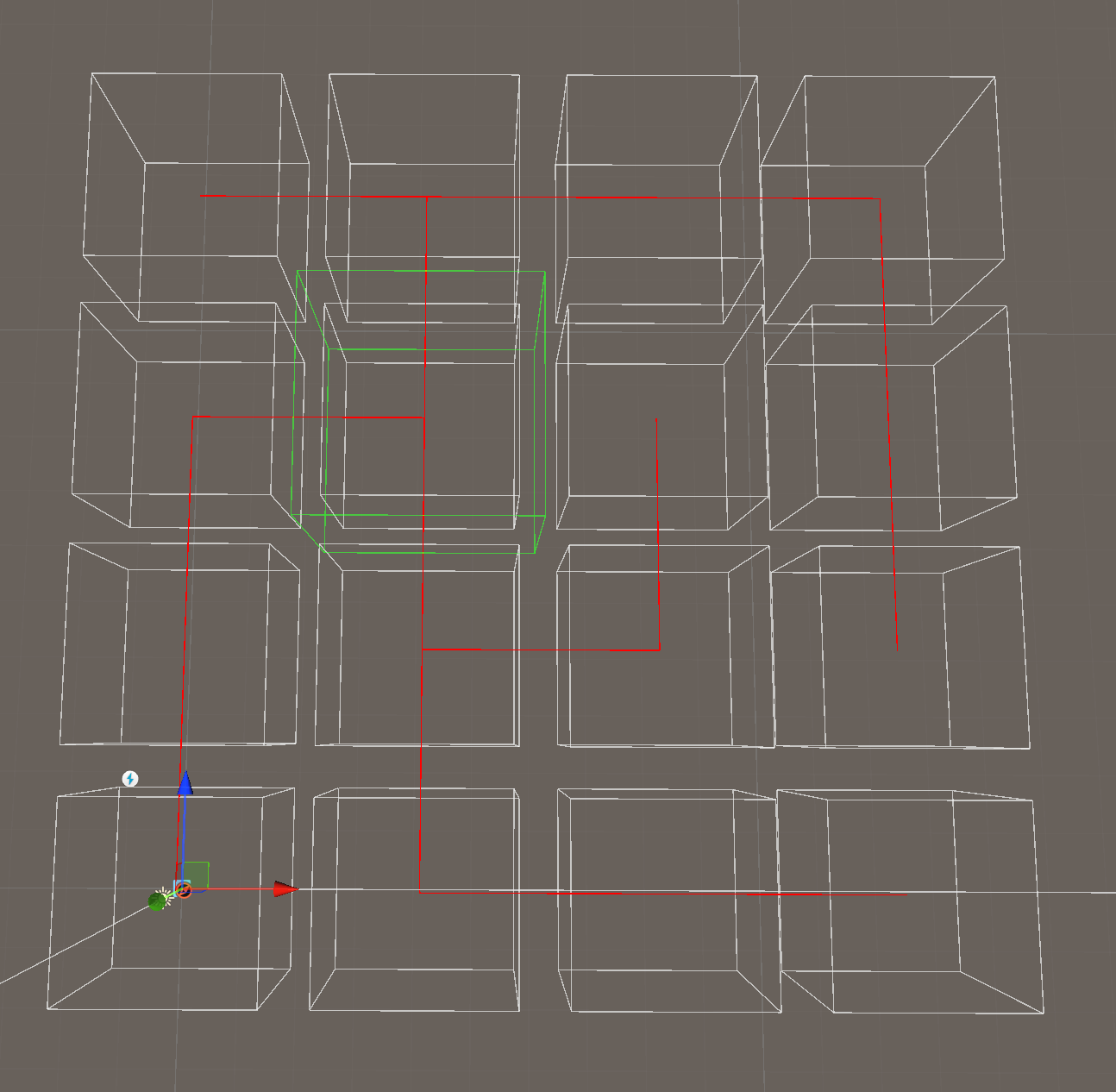
1. **Maze cell generation**. Set number of maze cells are spawned and are assigned their coordinates. (Example 4x4 square maze, cells are white squares).



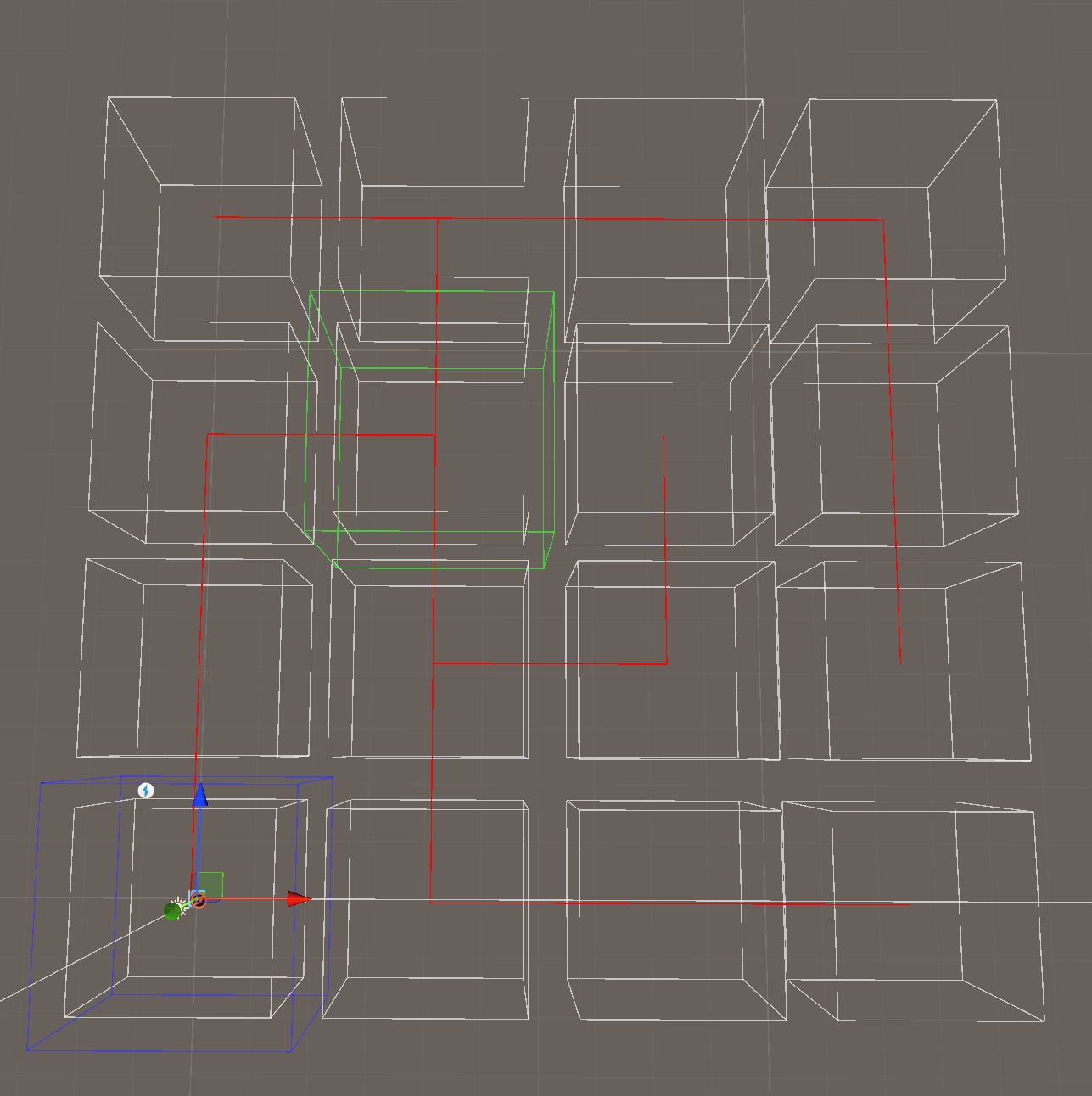
1. **Passage generation**. Passages are added for all cells to all adjustent cells. After that passages are randomly deleted. Randomization is partly decided by weights set by the user. After each passage deletion maze is validated using the "Flood Fill" algorithm the maze is still fully passable. If the maze becomes impassable, last passage deletion is reset. For example if a cell has 4 passages and randomization decides that it should have only 1 passage, random passages will be deleted one by one until 1 is left, except if by deleting some passage maze becomes impassable. (passages are red lines).



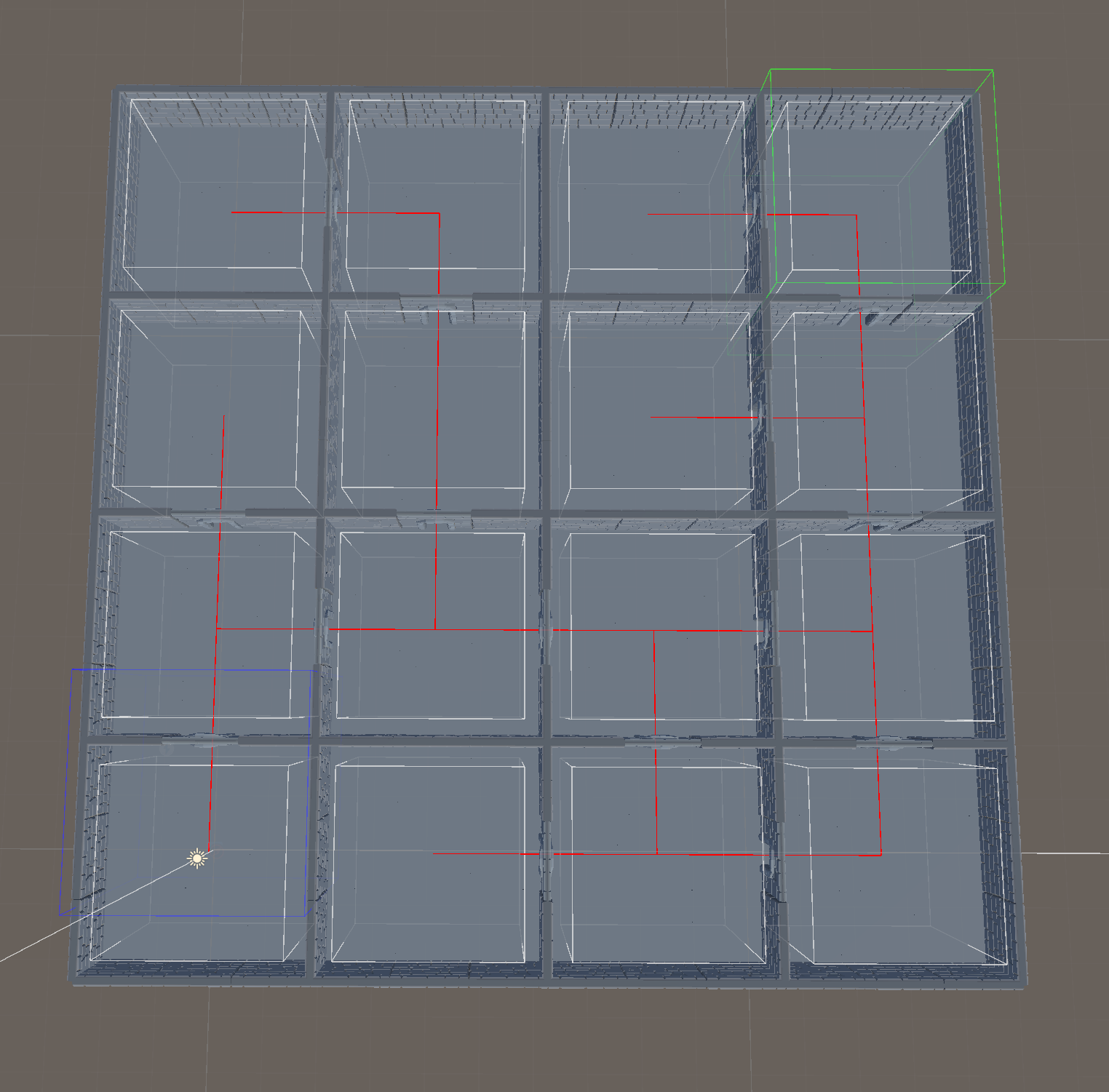
1. **Start generation.** One of all cells is randomly picked to be the starting cell (green square).



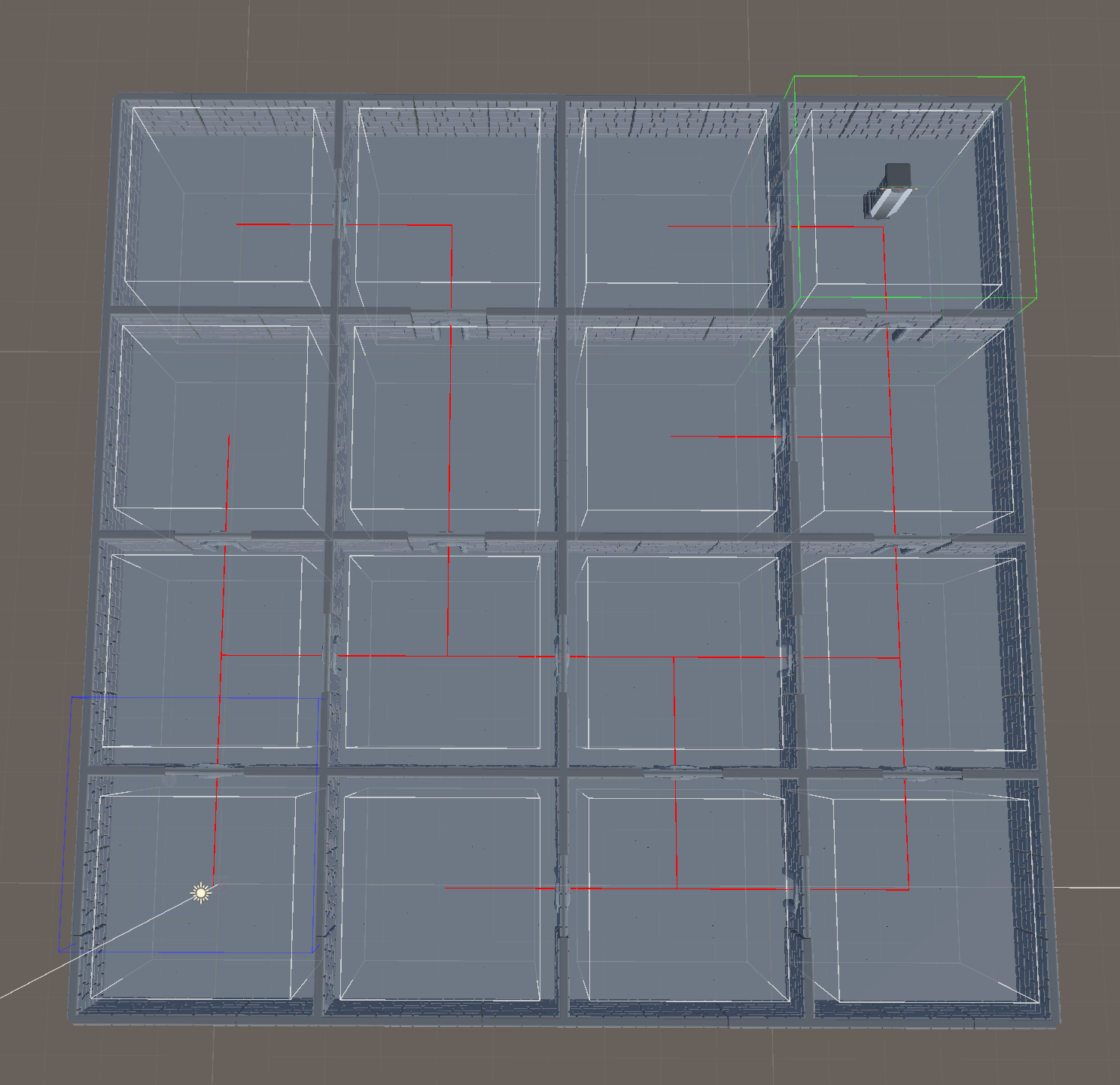
1. **Finish generation**. Finish cell is determined using the "Flood Fill" algorithm starting from the start cell to find the furthest cell (blue square).



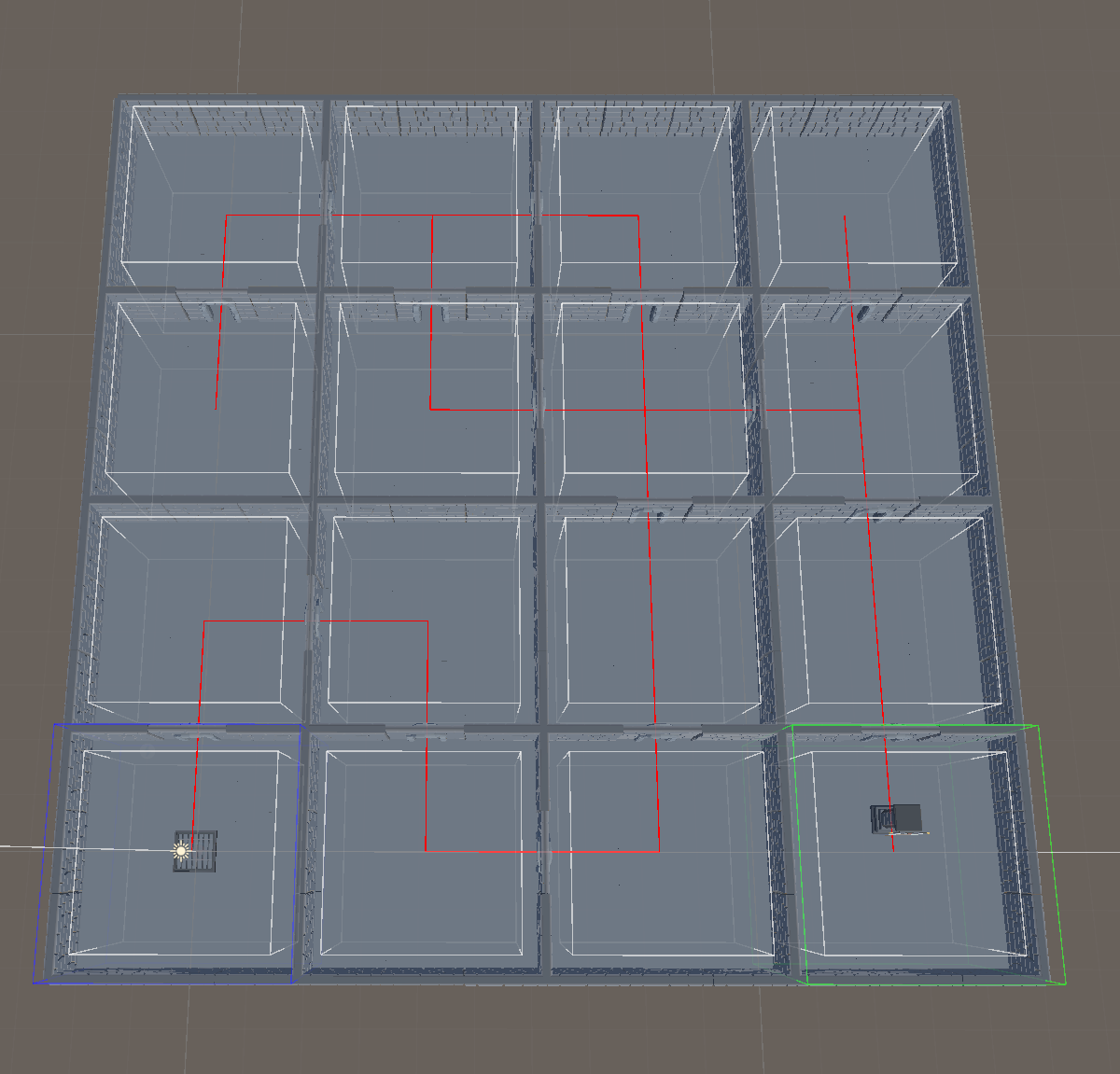
1. **Cell visuals generation**. Visuals are spawned from available prefabs to match entrances.



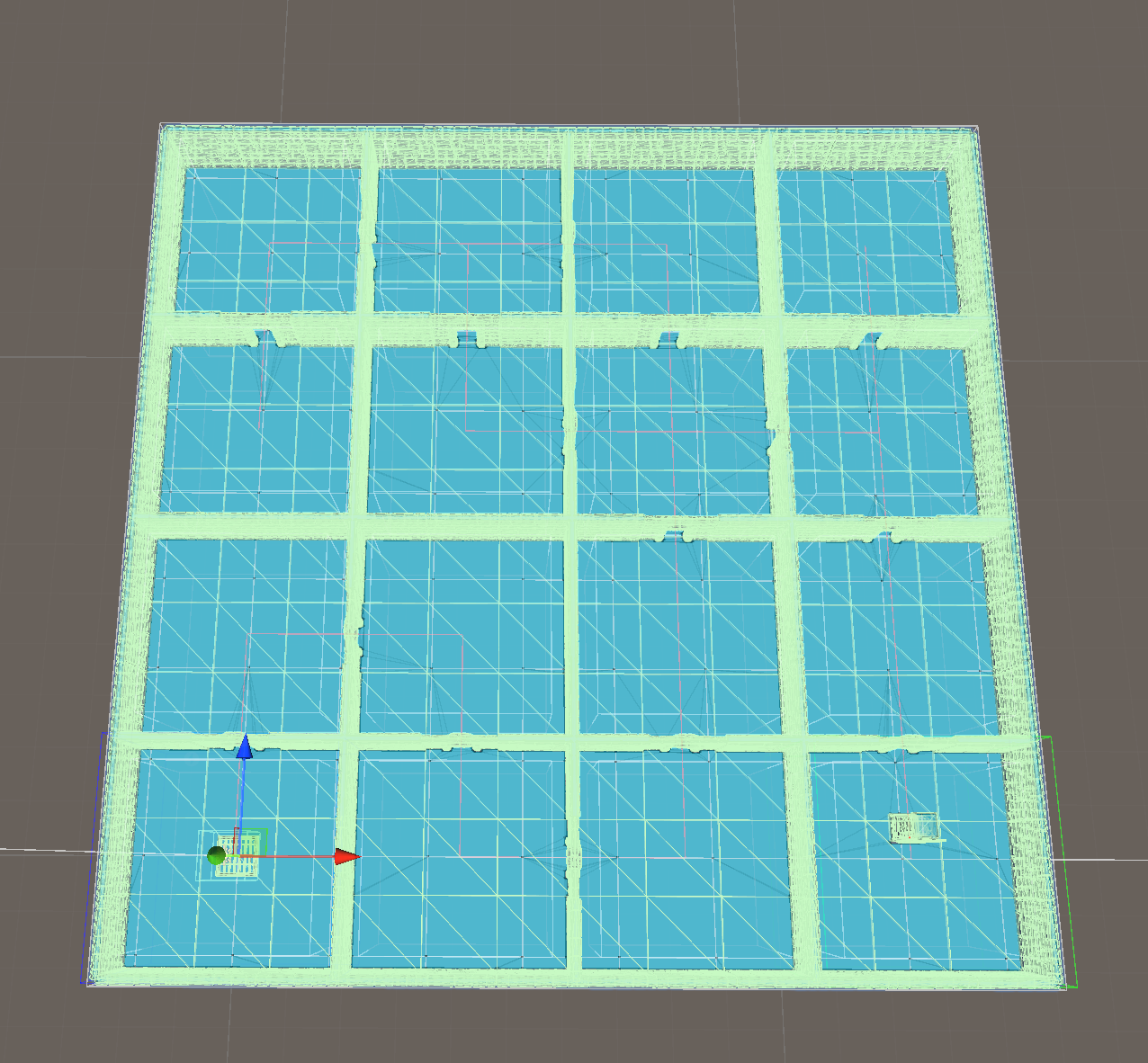
1. **Start visual generation**. Start visual prefab are spawned in the start cell.



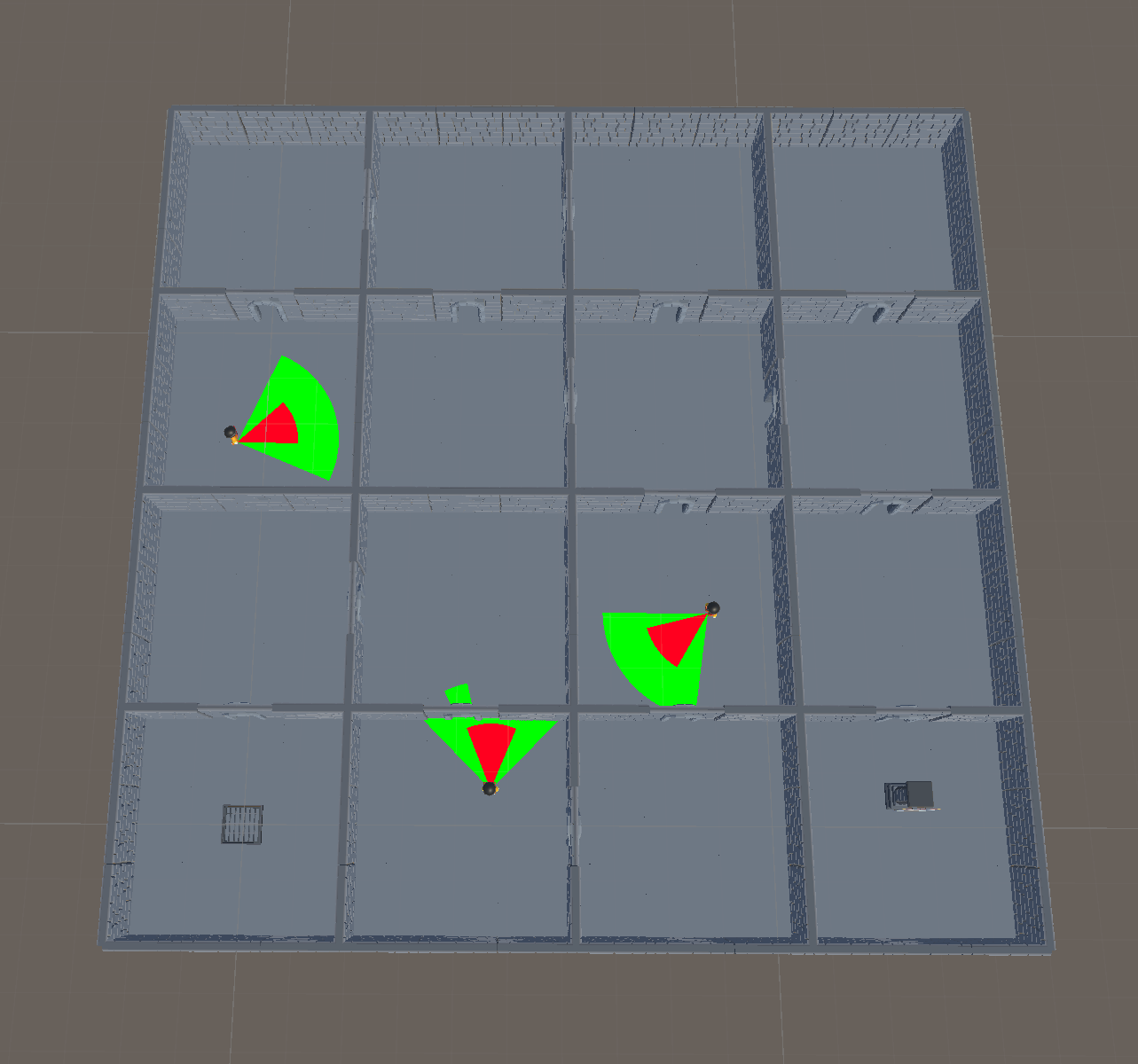
1. **Finish visual generation**. Finish visual prefab are spawned in the finish cell.



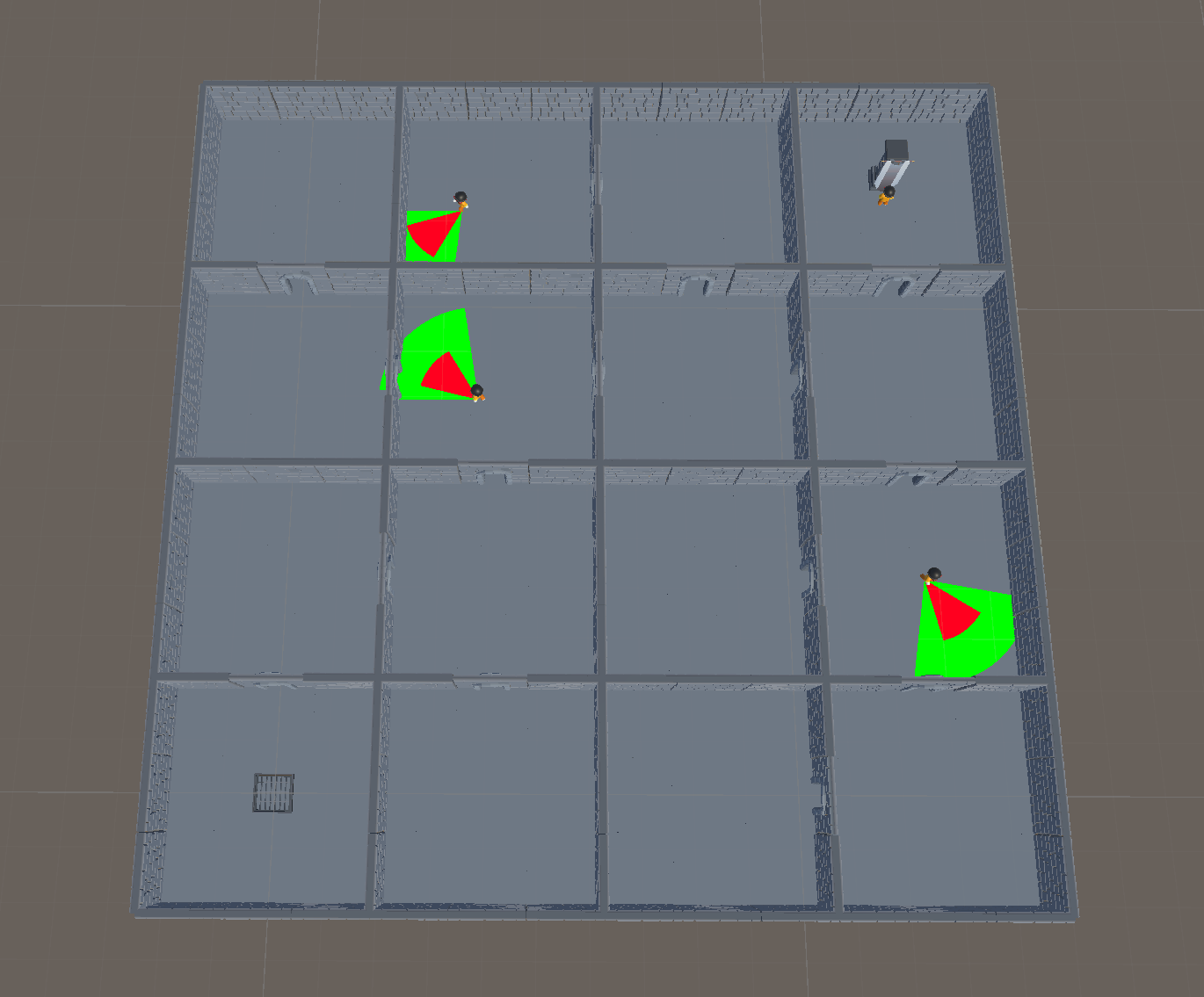
1. **Nav mesh generation**. Nav mesh generation for AI (blue ground).



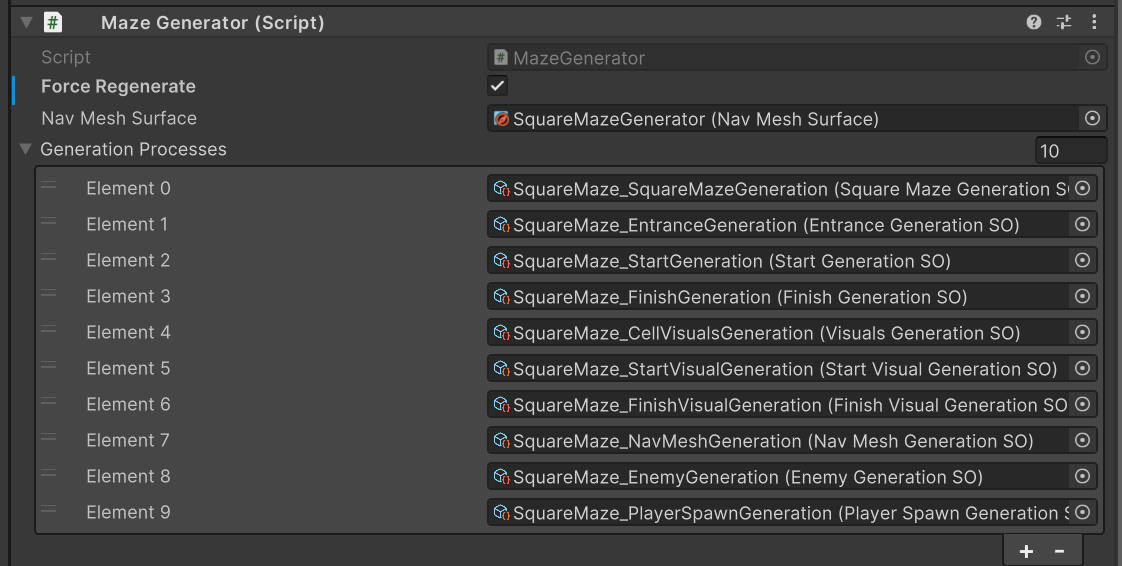
1. **Enemy generation**. Enemies are spawned in random cells except the start cell.



1. **Player spawn generation**. Player prefab is spawned and the game is started.

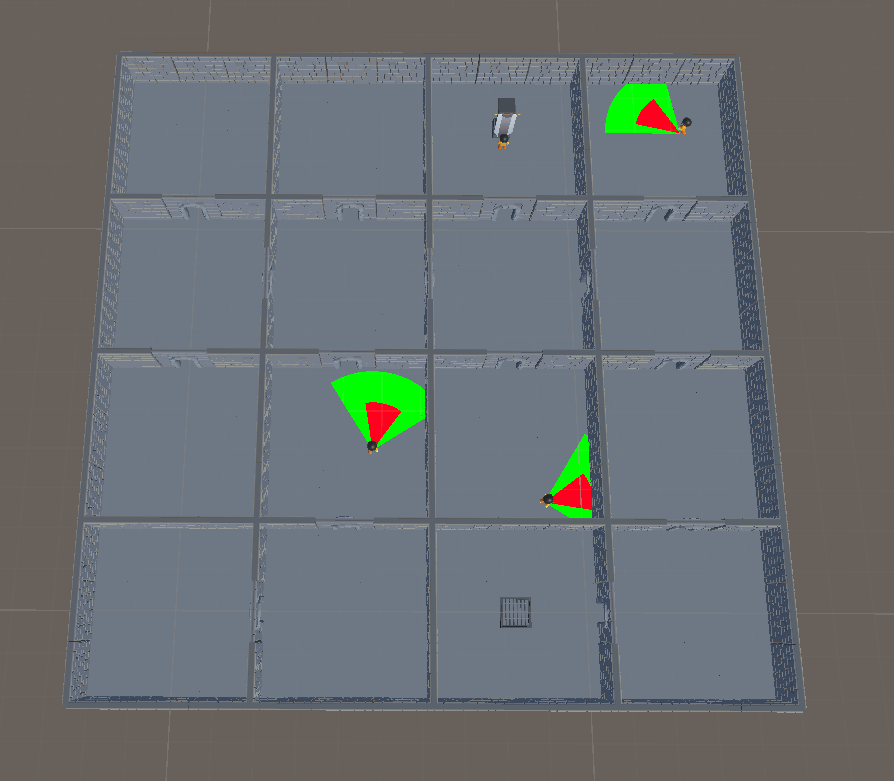


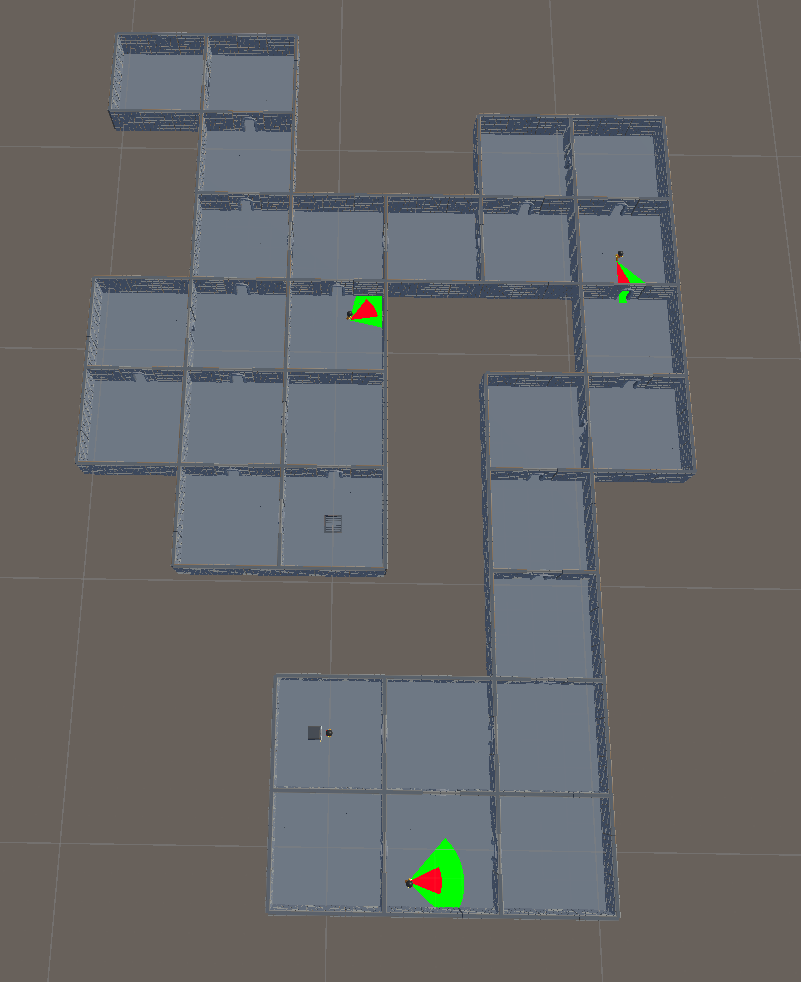
All of the previous processes can easily be changed to different ones, removed or more processes can be added. All generation variables are exposed in the inspector and can be easily changed in processes' scriptable objects. The idea was to create a modular system that can be easily modified and expanded.



In this example, the cell generation process generates square maze, but it can be switched with another one. There are few included in project:

* SquareMazeGeneration - example
* RandomCellGeneration - generates cells in random directions
* ImageGeneration - generates maze using provided image







Some processes depend on other processes. To handle their dependencies objects with needed interfaces are injected using Zenject dependency injection This way I prevent racing conditions and also it can further be upgraded to start processes asynchronously and make processes flow even faster. At this time there are four interfaces:

* ICellGeneration - interface to get generated cells
* IEntranceGeneration - interface to get generated passages between cells, entrances generation depends on *ICellGeneration*
* IStartCell - interface to get start cell, start cell generation depends on *ICellGeneration*
* IFinishCell - interface to get finish cell, finish cell generation depends on *ICellGeneration, IEntranceGeneration, IStartCell*

# AI

Enemies are spawned inside mazes to patrol them and follow/kill players. As any other process, enemies spawning can be disabled. Enemies have three states:

* Patrolling - default state when enemy walks around cell that he is spawned (patrolling cell can be changed after chasing player)
* Chasing - when player is seen, enemy starts to follow player
* Confused - when player is lost and last seen position is reached, enemy goes to confused state to look around, if player is not seen, goes to patrolling state

Enemies have two different action areas:

* View cone - cone in which player can be seen, by default cone’s color is green, changed to yellow when player is seen
* Damage cone - red cone which does damage to player when inside area

