Введение

В ходе обучения на втором курсе факультета ИТ в Московском Университете имени Витте я принял участие в треке Геймдев, в ходе которого присоединился к начинающей команде разработчиков игры в качестве разработчика 3Д моделей. Мои задачи заключаются не только в создании и оформлении локаций игры, но и помощи в составлении сюжета, а также внесении идей в процесс разработки. В ходе выполнения экзаменационной работы мне было предложено написать процедурную генерацию «подземелья», чтобы ускорить и разнообразить процесс генерации ландшафта для игры.

Основная часть

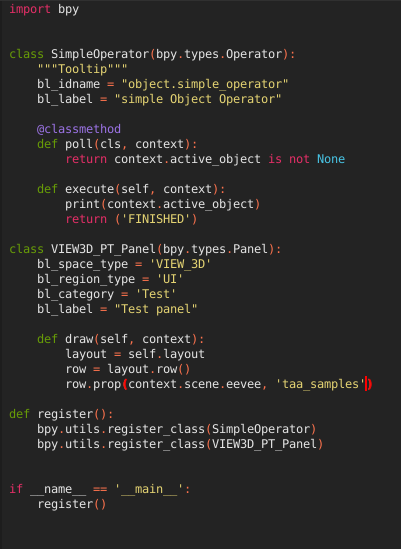
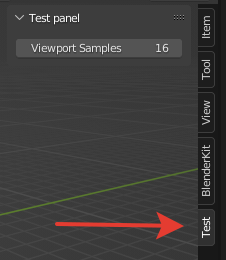
**Технологическая задача** заключалась в написании процедурной генерации пространства, схожего по структуре с лабиринтом с помощью встроенной системы скриптинга в Blender, но дальнейшее изучение данного инструмента показало, что подобный проект нереализуем с моими текущими знаниями. Этот факт заставил меня пересмотреть техзадание и перейти к изучению процедурной генерации с использованием программных средств Unity.

**Список решаемых задач:**

* 1. Разработка программы, позволяющей процедурно сгенерировать локацию для проектируемой игры.
     1. Изначально программа разрабатывалась с использованием встроенных средств Blender. В дальнейшем разработка велась с использованием Unity и встроенного языка C#.
  2. Изучение методов работы со встроенными инструментами скриптинга в Blender, программой Unity и языком программирования С#.
  3. Генерация удовлетворяющей условиям задачи структуры и наложение текстур на поверхности.
     1. Задание выполнялось с использованием написанной на C# программой, а также встроенных функций Unity.

Реализуемая архитектура.

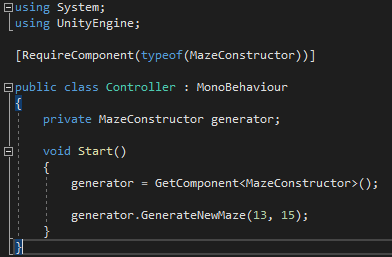
2.1. Изначально работа велась с использованием внутренних инструментов Blender, но после продолжительного исследования стало ясно, что предоставляемые инструменты непосредственно для процедурной генерации графики внутри программы гораздо удобнее и практичнее скриптинга, ввиду чего мне не удалось найти информацию хотя бы об одном подобном проекте с использованием скриптинга, а дальнейшие попытки написать код, хотя бы отдалённо отвечающий требованиям техзадания, не увенчались успехом. Всё, что мне удалось написать – это создание панели для управления мешем (объектом) в самом проекте.

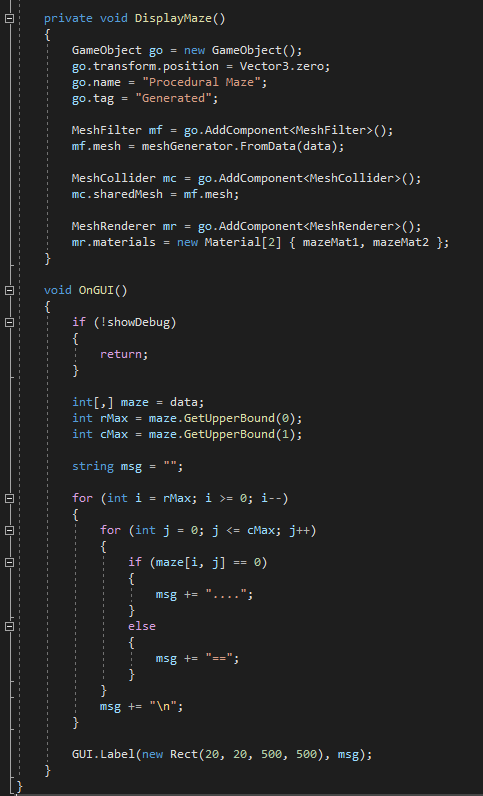
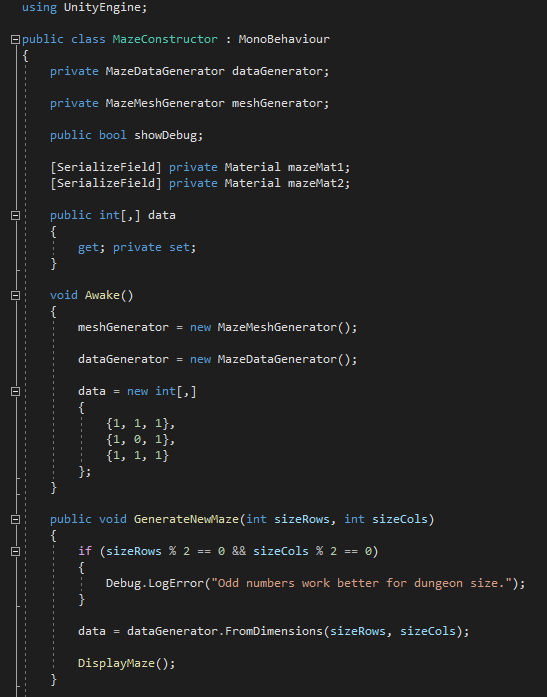
2.2. Указанные выше проблемы привели меня к изучению средств разработки при помощи Unity, где существовало множество подобных проектов, и мне было что из них почерпнуть. Результатом стала генерация простейшей сети коридоров, которые были названы лабиринтом для упрощения взаимодействия и восприятия. Архитектура представляет собой пять скриптов:



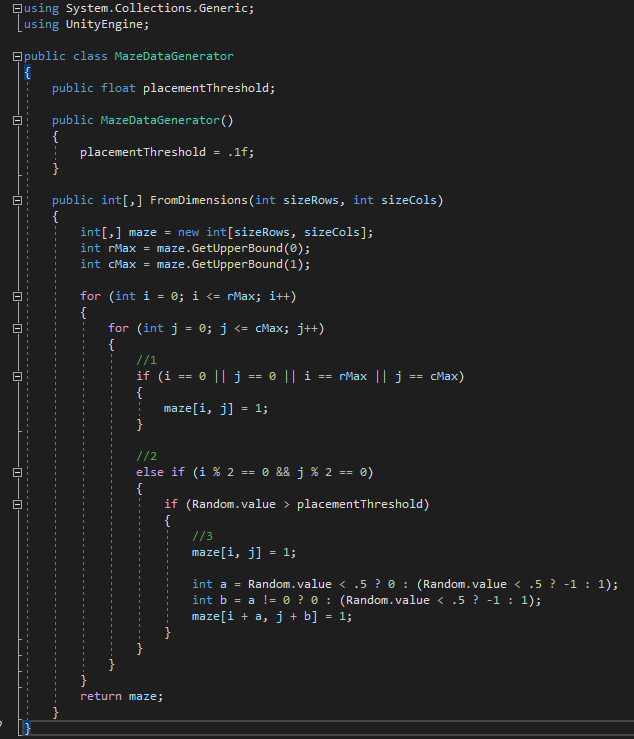
* Controller – является основным и используется на объекте для генерации самого лабиринта. Внутри скрипта осуществляется ввод размеров необходимого пространства, минусом является отсутствующая возможность введения размеров непрограммным путём (через отдельное окно), и на текущем этапе разработки изменение параметров генерации лабиринта возможно только внутри кода.



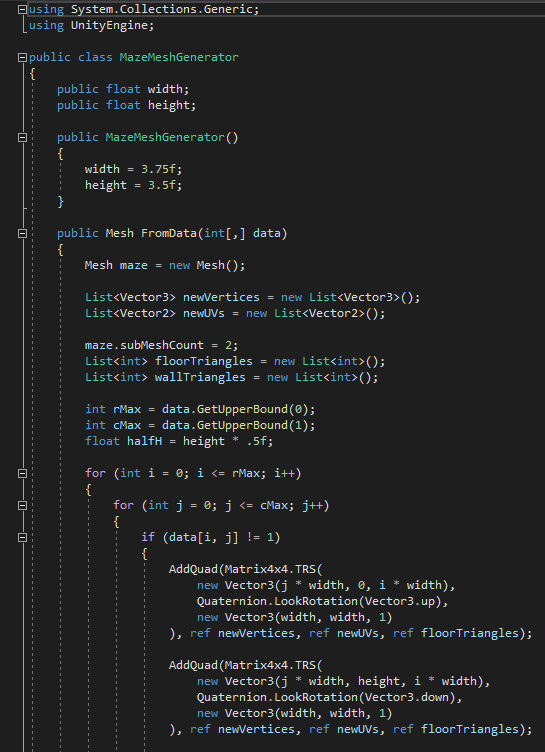
* MazeConstructor – скрипт, содержащий в себе правила генерации лабиринта. В его структуру также входят несколько вспомогательных классов, инструменты настройки материалов лабиринта и процедура, отображающая сам лабиринт.



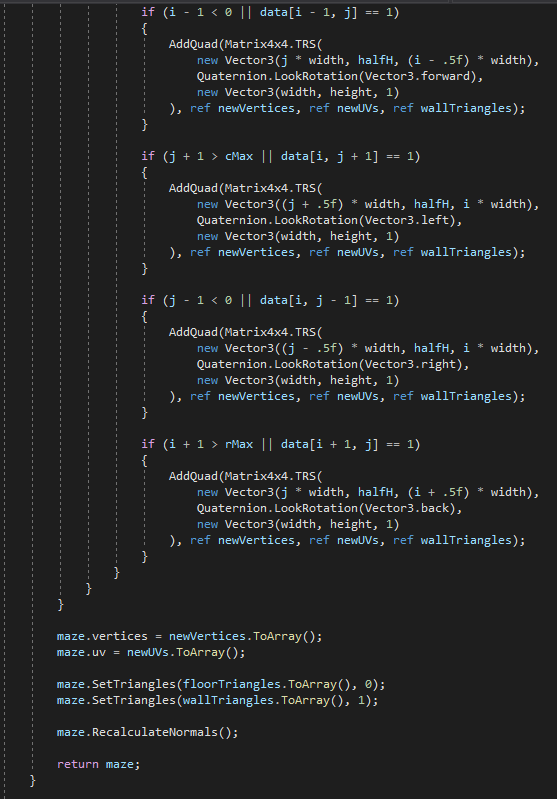
* MazeDataGenerator – класс, служащий для инкапсуляции логики генерации лабиринта. Он не используется напрямую в качестве компонента, нужен для упаковки данных из MazeConstructor в один класс для более удобного применения.



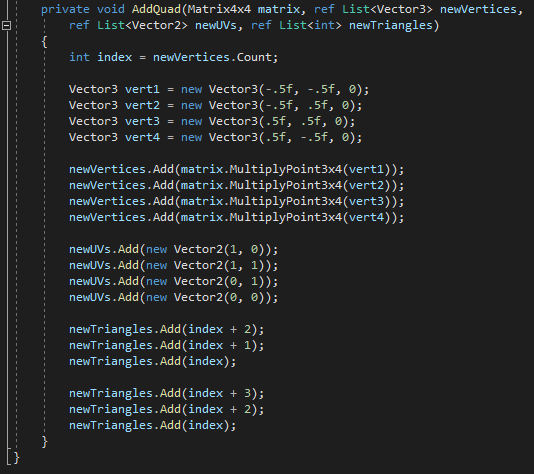
* MazeMeshGenerator – ещё один вспомогательный класс. Для точного понимания его работы необходимо знать, как строится сам лабиринт: он представляет собой сетку, состояющую из квадов, состоящих из треугольников, которая так же имеет свои параметры и, соответственно, нуждается в собственном классе. Данный скрипт самый длинный в проекте по причине большого объёма данных и параметров, необходимых для генерации сетки. MazeConstructor сначала создаёт экземпляр MazeMeshGenerator, а затем уже вызывает метод генерации сетки и, будучи привязанным к объекту, «натягивает» меш поверх сгенерированной сетки, делая лабиринт видимым.



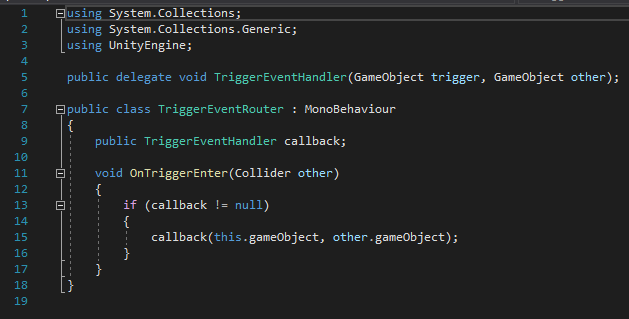
Методы AddQuad служат для создания тех самых квадов, из которых состоит сетка. Они состоят из двух треугольников, вершины которых по сути представляют собой массив с координатами. Также в начале скрипта задаются параметры ширины и высоты коридоров, но внешняя настройка также не доступна, что оставляет дополнительные задачи для доработки проекта.



Здесь можно заметить параметр RecalculateNormals(), что позволяет подготовить сетку для освещения. Также возникает вопрос: зачем использовать такую сложную структуру сетки? Два треугольника, на которые делится квад, позволяют создать две подсетки и задавать необходимый материал в зависимости от положения: материал стены или материал пола/потолка.

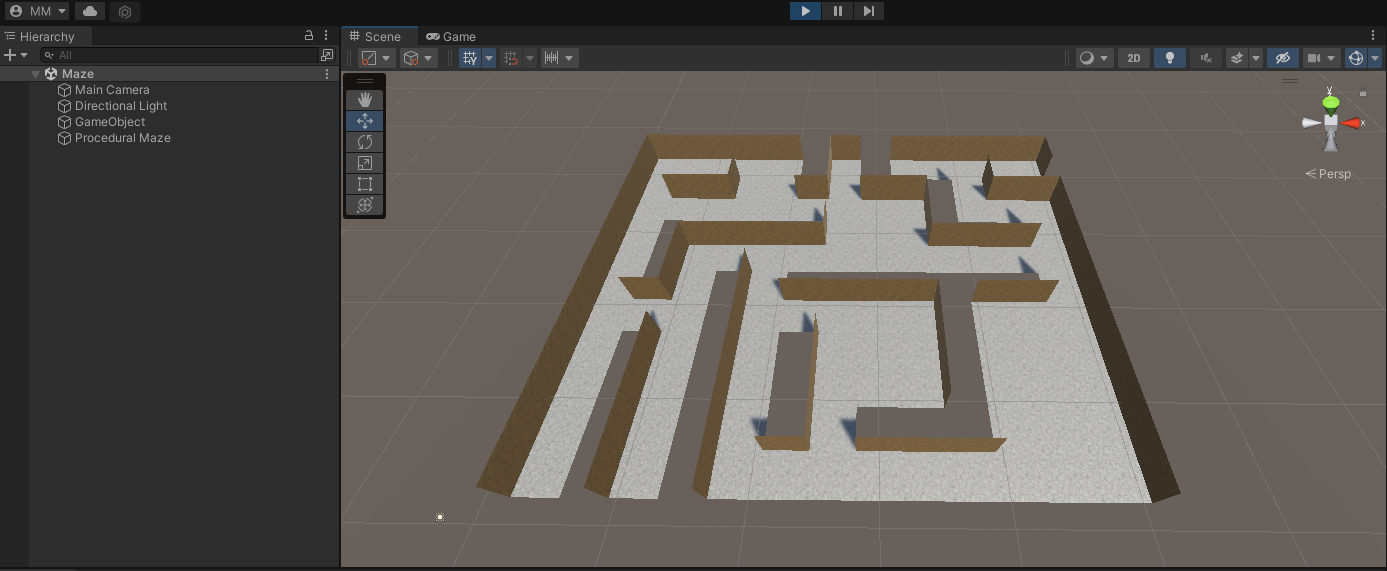


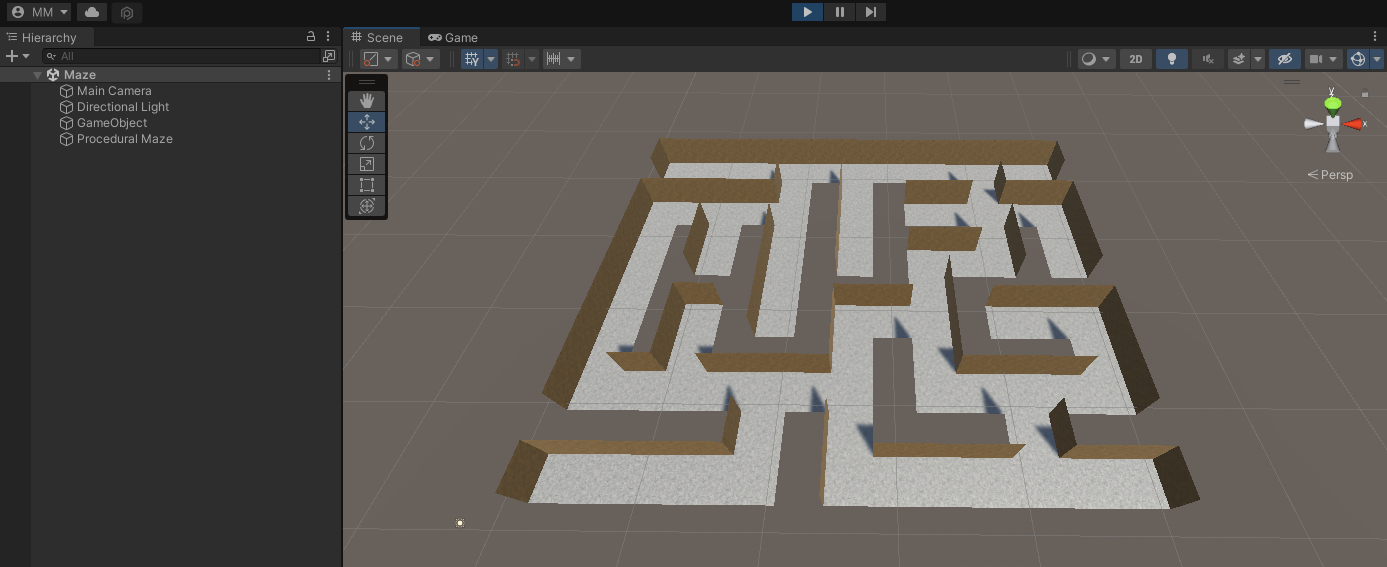
* TriggerEventRouter – скрипт, который служит в качестве регистратора определённых событий, в данном случае взаимодействия с коллайдером. В дальнейшем сюда можно помещать некоторые триггеры, которые смогут разнообразить взаимодействие с локацией.



Описание работы

3.1. При запуске проекта Unity в сцене появляется новый объект – лабиринт, сгенерированный с нуля и всегда построенный заново:

****



Из-за нормалей, направленных внутрь лабиринта мы не видим потолок и стены, находящиеся с нашей стороны. При перемещении внутрь лабиринта мы видим все стены и пол с потолком:



В проекте не настроено освещение, из-за чего наблюдаются нереалистичные эффекты.

3.2. К сожалению, на данном этапе построение лабиринта в данном проекте имеет сильную слабость: не осуществляются проверки на «проходимость», и, хотя вероятность достаточно мала, иногда можно наблюдать следующий результат:



Как видно на скриншоте, один блок находится в обособленном пространстве без возможности доступа. Для простейших проектов это не стало бы сильной проблемой, но, если вместе с подобной структурой будет производиться генерация определённых взаимодействий, необходимых для дальнейшего продвижения, это приведёт к поломке геймплея и непроходимости уровня.

Вывод

Как видно из отчёта, проект имеет множество перспектив развития. Как минимум необходимо исправить ряд недочётов, как мелких, так и существенных. Лично мне хотелось бы видеть не только пустые коридоры, но и объекты внутри, а также комнаты, лестницы и прочие объекты, которые серьёзно дополнили бы генерацию уровня. Хотя Blender предоставляет гораздо более простые способы генерирования графики с помощью встроенных инструментов, подобная работа также имеет ряд преимуществ, ведь её проще интегрировать в проект и управлять ей непосредственно изнутри проекта, не заставляя дизайнера каждый раз генерировать структуры в Blender и импортировать это в Unity. В ходе выполнения работы я изучил не только способы генерации графики в Unity, но и познакомился с несколькими инструментами Blender, которые существенно могли бы упростить некоторые задачи в моих последующих проектах.

Библиография

* https://stdpub.com/unity3d/proczedurnaya-generacziya-urovnej-i-labirintov-v-unity – процедурная генерация лабиринта в юнити.
* https://www.kodeco.com/2425-procedural-level-generation-in-games-using-a-cellular-automaton-part-1 – интересный сайт с информацией, где я нашёл способ избавиться от изолированных пространств, а также иной способ генерации более сложных структур, пусть и в 2Д.
* https://blender3d.com.ua/category/scripting/ – сайт со скриптингом в блендере, но на устаревшей версии.
* https://render.ru/ru/Blender\_Blog/post/23344 – сайт со скриптингом в блендере на текущей версии, с помощью него я пытался изучить сам инструмент.
* https://docs.blender.org/api/current/bpy.props.html – сайт с документацией для библиотеки python bpy, используемой в скриптинге блендера.
* https://habr.com/ru/articles/495588/ – полезная ссылка, найденная мной уже после завершения работы над проектом, сильно заинтересовавшая способом генерации помещений не только из коридоров.