УДК 004

Реализация алгоритма процедурной генерации для заполнения игровых пространств в Unity

Фатеенков Данила Витальевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В статье рассмотрен процесс написания алгоритма генерации объектов на игровой локации для игры, созданной в Unity. Для реализации используется язык программирования C# и функции, представленные в Unity API. Рассматриваются основные методы нахождения коллизий при генерации, а также методы для генерации объектов в игровом пространстве.
Ключевые слова: Unity, процедурная генерация, C#

Implementing a procedural generation algorithm to fill game spaces in Unity

Fateenkov Danila Vitalievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article describes the process of writing an algorithm for generating objects in the game location for a game created in Unity. For the implementation the C# programming language and the functions provided in the Unity API are used. The basic methods of finding collisions during generation are considered, as well as methods for generating objects in the game space.
Keywords: Unity, procedural generation, C#

1. Введение

1.1 Актуальность

Процедурная генерация – метод создания объектов или данных алгоритмически, заранее определённым способом (он может быть обоснован математически, например). В основном данные создаются на основе раннее указанных формул или значений. Для генерации часто используется псевдослучайная генерация значений, зависящая от определённых значений (например, от установленного времени на ПК пользователя).

Процедурные генерации используются во многих сферах, где имеет место создание объектов и данных, которые вручную разработать тяжело и долго: в 3D часто генерируют ландшафты на основе процедурных алгоритмов, в киноиндустрии – спецэффекты для фильмов, в музыке – шумы и другие звуки.

Данный метод разработки актуален и используя процедурные генерации можно значительно оптимизировать процесс разработки, тем самым сократив время на создание однотипных объектов или данных.

1.2 Обзор исследований

К.Б. Фёдоров рассмотрел алгоритмы моделирования ландшафтов для компьютерных игр и описал собственный подход к процессу моделирования [1]. А.А. Третьяков описал применение алгоритма Marching Cubes для процедурной генерации 3D-геометрии [2]. В статье рассмотрен сам алгоритм, а также прочие алгоритмы генерации, относящиеся к рассматриваемой задаче. К.В. Никитин, А.А. Комаров и Е.Э. Боргояков рассмотрели проблемы игрового баланса при разработке игр с внедрением алгоритмов процедурной генерации в игровой процесс [3]. А.В. Шубин и Б.А. Козар описали алгоритм процедурной генерации игровых объектов для игры, созданной в Unity3D [4].

1.3 Цель исследования

Цель – реализовать алгоритм генерации объектов на игровой локации в Unity.

2. Материалы и методы

Для достижения поставленной задачи используется язык программирования C# и программное обеспечение для разработки Unity.

3. Результаты и обсуждения

Процедурная генерация, в первую очередь, зависит от того, какие объекты необходимо разместить в игровом пространстве. Подходов для реализации алгоритма процедурной генерации много, но часто необходимо писать уникальный скрипт генерации. Это связано с различием подходов и требований к конечному результату. Одним из распространённых случаев генерации можно считать создание интерьера для помещений. Для генерации интерьера можно определить несколько подходов:

1. Генерация заранее заготовленных шаблонов с применением псевдослучайных генераторов. Это самый простой способ создания помещений. Суть состоит в том, чтобы заранее заготовить помещения с интерьерами и с помощью алгоритма добавлять их на игровую локацию. Недостаток способа в том, что уникальность ограничена заранее заданными шаблонами и многие созданные экземпляры будут идентичны. Реализация такого метода (и не только данного метода, но и последующих тоже) состоит из нескольких этапов: сначала задаются все переменные, которые необходимы для генерации. В случае рассматриваемого алгоритма генерации с использованием заготовленных шаблонов, достаточно задать только массив комнат:

```
    public GameObject[] rooms; //массив комнат
    private GameObject roomGenerate; //выбранная комната для генерации
```

Для генерации можно использовать метод “Range” из модуля “Random”, который выбирает случайный элемент из массива. Метод принимает на вход 2 параметра: минимальное и максимальное числа, которые можно выбрать. Максимальным числом можно определить длину массива, используя метод “Length”:

```
    roomGenerate = rooms[Random.Range(0, rooms.Length)];
```

Строка генерации помещается в функцию “Start” для того, чтобы генерация срабатывала при запуске и только один раз. Остаётся только определить массив комнат (см. рис. 1).

Для создания экземпляра шаблона, необходимо применить функцию Instantiate, которая получает на вход следующие параметры:

1. Шаблон объекта, который необходимо создать на сцене;
2. Координаты, по которым будет расположен объект.

Так как комната не будет смещаться относительно пустого объекта-генератора, то выбираются координаты объекта-генератора:

```
    var roomNew = Instantiate(roomGenerate, new Vector3(this.gameObject.transform.position.x,
this.gameObject.transform.position.y,
this.gameObject.transform.position.z), Quaternion.identity);
```

Для создания нового экземпляра ранее созданного префаба используется функция “Instantiate” библиотеки “UnityEngine”. На вход функции поступает 3 параметра: префаб, экземпляр которого необходимо создать, а также его местоположение и поворот в пространстве. Функция работает только с префабами, с обычными моделями и объектами работать не будет. “Instantiate” позволяет создавать не только 3D-объекты на игровом поле, но также и эффекты, 2D-спрайты и другие объекты, которые могут быть представлены в виде префаба в проекте.

Для определения поворота объекта использован “Quaternion.identity”. Этот кватернион означает, что экземпляр будет выровнен относительно мировой или родительской оси в игровом пространстве.

Для определения координат в пространстве используется метод “transform.position”, который содержит значения для X, Y, Z координат.

Также “Instantiate” может принимать на вход 4 параметр - “parent” в формате “Transform”. Параметр определяет родительский объект созданного экземпляра (необходимо, если многие экземпляры могут быть частью отдельного объекта или есть надо структурировать экземпляры по определённому свойству).

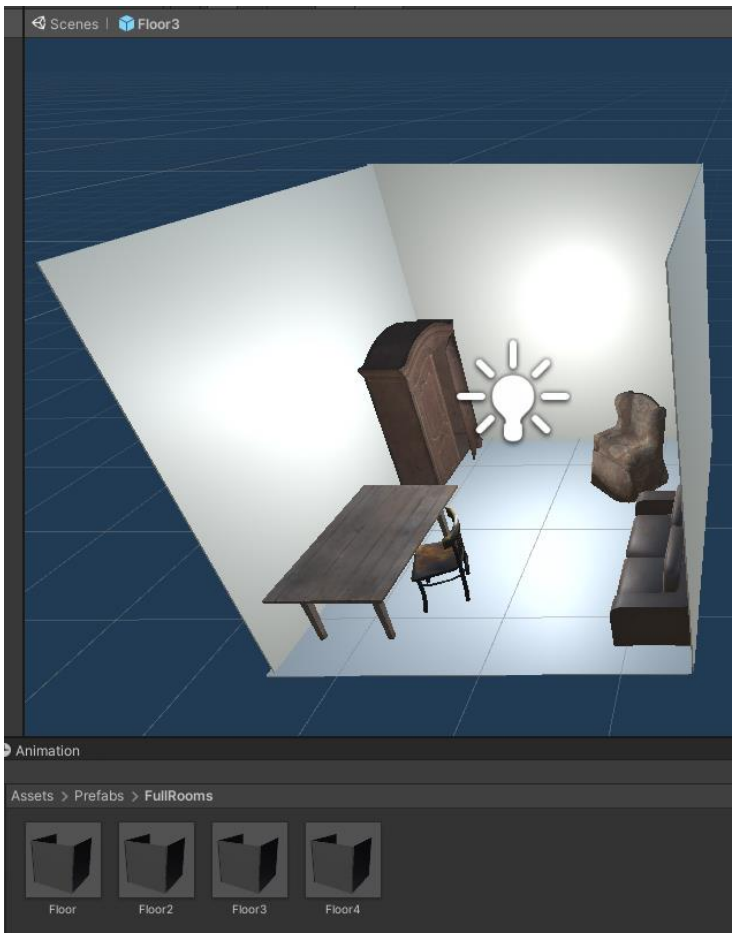

Рисунок 1. Шаблоны комнат, которые использованы для генерации

Применяя такой алгоритм, можно достичь неплохих результатов, но всё ограничивается количеством заданных шаблонов (см. рис. 2).

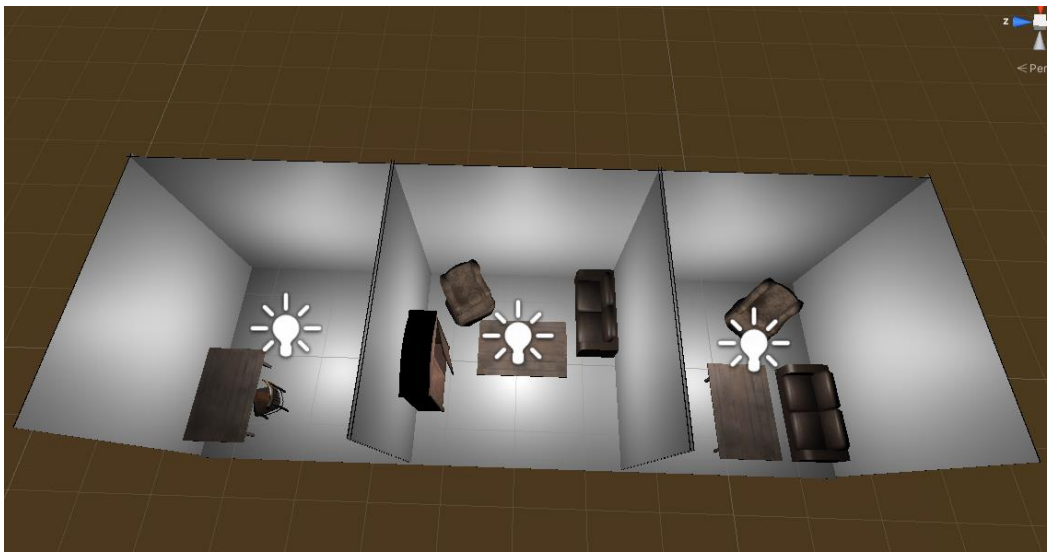

Рисунок 2. Пример работы генерации с использованием заранее заготовленных шаблонов

Остаётся только определить расположение комнат. Для этого достаточно создать любой невидимый для игрока объект, поместить в него

скрипт генерации и на том месте, на котором установлен невидимый объект, будет создаваться комната.

Алгоритм очень прост в реализации, но неудобен, если необходимо создать как можно больше уникальных локаций. Неудобство достигается тем, что разработчик должен вручную готовить шаблоны и если шаблонов будет много, то процесс разработки может затянуться. Для того, чтобы избежать такой ситуации, можно усовершенствовать алгоритм и для генерации использовать массив не полностью готовых комнат, а массив пустых помещений, в которые будет добавляться интерьер.

2. Генерация пустых помещений с последующим добавлением интерьера. Данный способ можно разделить на 2 подхода: использование шаблонов интерьера комнат и полноценная генерация интерьера для каждой комнаты.

Первый подход предполагает использование заранее заготовленных шаблонов. Недостаток в связи с уникальностью комнат также остаётся, но за счёт использования нескольких шаблонов интерьера для одной комнаты повышается и уникальность. Можно также определить тип помещения, что позволит разделить шаблоны на группы и использовать конкретные шаблоны для определённых типов комнат.

Для определения типа комнат можно также использовать псевдослучайный генератор. Для начала задаётся массив (желательно глобальный, чтобы все комнаты могли обращаться к одному массиву, а не создавать новый локальный при каждой генерации), в котором хранятся метки, определяющие тип помещения. В качестве примера можно задать следующие метки:

1. Жилое помещение – для комнат такого типа характерны следующие объекты: столы, кровати, диваны, шкафы и другие;
2. Рабочее помещение – для комнат такого типа характерны следующие объекты: столы, кровати, диваны, шкафы и другие;
3. Складское помещение – для комнат такого типа характерны следующие объекты: столы, кровати, диваны, шкафы и другие.

Объявление массива таких меток выглядит следующим образом:

```
public static string[] tags = new string[3] {“Living”, “Work”, “Warehouse”};
```

Если данный массив выделен в отдельный скрипт, то необходимо получить его, ссылаясь на его расположение (обратиться к классу, в котором он был определён):

```
tag = TagsArray.tags[Random.Range(0, TagsArray.tags.Length)];
```

Объявление переменных также изменится:

```
    public GameObject[] rooms; //массив комнат
    public GameObject[] interiors; //массив интерьеров
    private GameObject roomGenerate; //выбранная комната для генерации
    private string tag; //метка
```

Массив интерьеров должен включать в себя только те шаблоны, которые актуальны для комнаты с выбранной меткой. Для этого можно определить массив шаблонов для каждой метки заранее в том же скрипте, в котором определены и метки:

```
    public static string[] tags = new string[3] {"Living", "Work", "Warehouse"};
    public static GameObject[] living_interior;
    public static GameObject[] work_interior;
    public static GameObject[] warehouse_interior;
```

Получившийся скрипт определения массивов для каждого типа комнат стоит поместить в пустой объект на игровом пространстве и в него добавить все необходимые шаблоны (стоит учесть то, что статические переменные нельзя изменять через интерфейс Unity, поэтому стоит либо ссылаться на шаблоны через C# функции, либо добавить временные переменные-массивы, которые будут содержать шаблоны и передавать их в статические переменные перед началом генерации).

Также появляется проблема разных размеров пустых комнат. В таком случае необходимо создать шаблоны для возможных размеров, но это увеличит потребление памяти при генерации и может замедлить скорость работы алгоритма (так как появляется необходимость формировать массив шаблонов под конкретный размер комнаты).

Нужно учесть также то, что расположение объектов интерьера не закреплено за определённым родительским элементом (если родителем является пустой объект), что может вызвать конфликты при создании экземпляров: объекты интерьера новых экземпляров могут появляться не по глобальным координатам (которые будут определены алгоритмом) и тем самым нарушить работу алгоритма генерации.

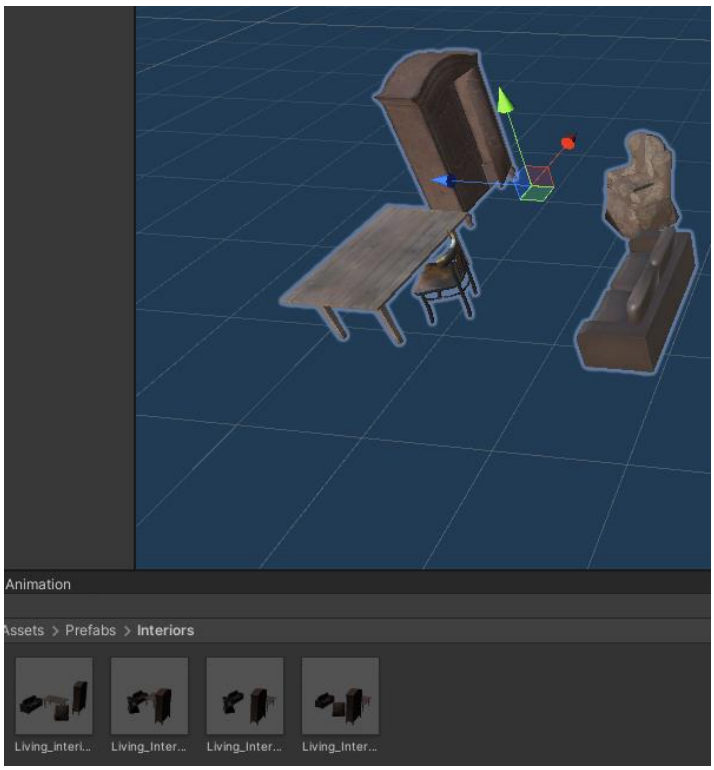

Рисунок 3. Шаблоны интерьеров перед генерацией

Внеся все необходимые изменения, можно получить более совершенную генерацию интерьеров (см. рис. 4).

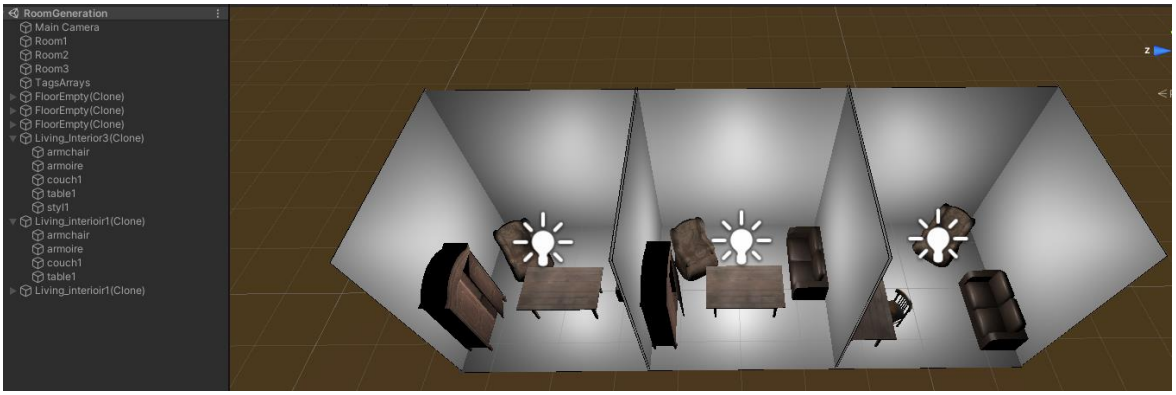

Рисунок 4. Пример работы изменённого алгоритма генерации

Но данный подход не позволит достигнуть наиболее высокой уникальности, потому что алгоритм снова будет ограничен количеством шаблонов.

Второй подход позволяет устранить данный недостаток, но он является наиболее неустойчивым из-за того, что алгоритм трудно будет использовать для формата помещений, которые не предусмотрены заранее. Отходя от шаблонов необходимо помнить, что объект генерации может иметь различную форму или представление в иерархии объектов.

Чтобы избежать проблем с представлением комнаты в игровом пространстве, стоит определить формат изначально (до начала генерации). В рамках статьи определено, что пустое помещение представлено не в виде

одного объекта, а является набором, представляющих стены, пол и потолок. Это необходимо для того, чтобы корректно определить некоторые свойства объектов для дальнейшего расположения в комнате (например, расположение кровати или шкафа возле стены).

В качестве массива для генерации используется массив объектов, которые будут расставляться в помещении, а не шаблонов. Для этого необходимо изменить код класса, в котором определены массивы шаблонов:

```
    public static string[] tags = new string[3] {"Living", "Work",
"Warehouse"};
    public static GameObject[] living_objects;
    public static GameObject[] work_objects;
    public static GameObject[] warehouse_objects;
```

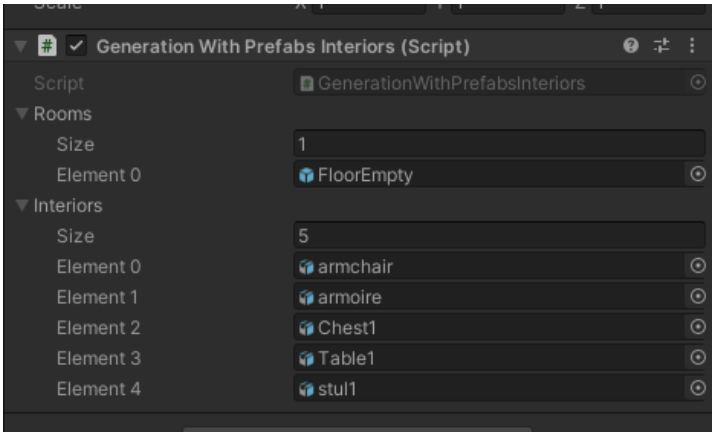

Рисунок 5. Массив объектов для жилой комнаты в настройках скрипта генерации

Процесс добавления объектов сложен, многие элементы интерьера могут иметь уникальный алгоритм определения расположения в помещении и нужно учесть обязательные для каждого типа объекты (если в этом есть необходимость). Для каждого объекта можно создать отдельный алгоритм, который будет срабатывать при добавлении объекта на сцену.

Можно отметить, что уникального для всех объектов подхода генерации нет, потому что некоторые объекты могут располагаться возле стен, другие могут быть расположены в любой точке комнаты (если нет коллизий с другими объектами), то есть необходимо учитывать определённые условия при добавлении объекта. Поэтому эффективнее всего создать отдельные для объектов алгоритмы, которые будут определять расположение объекта в комнате в зависимости от определённых параметров. Например, можно рассмотреть алгоритм установки дивана в жилой комнате. Данный объект должен располагаться возле одной из стен комнаты, поэтому в первую очередь определяется стена, возле которой он будет располагаться. Для этого выбирается один из компонентов шаблона помещения (изначально всего 5 элементов, 3 из которых – это стены). После генерируется объект, но создаётся он в стене. Связано это с тем, что алгоритмом заранее выбирается середина

элемента и сделано это для упрощения расположения объекта в комнате. После этого дивана смещается в угол комнаты и поворачивается, тем самым занимая необходимую позицию.

Рисунок 6. Результат работы алгоритма расположения дивана в комнате

Таких объектов может быть сколько угодно, необходимо только ограничить их количество в одной комнате. Для этого можно проверять наличие коллизий с созданными объектами в комнате. Количество коллизий и будет характеризовать количество объектов в комнате. Для реализации подсчёта необходимо ввести массив, который будет содержать все коллизии. Для подсчёта можно использовать метод “Count” из библиотеки LINQ, а подсчитывать элементы с тегом “Interior” (это необязательно, если не предполагается наличие элементов с другими тегами в комнате):

```
    private Collider[] hitColliders;

    hitColliders = Physics.OverlapBox(this.transform.position,
this.transform.localScale, Quaternion.identity);
    int roomCollidedCount = hitColliders.Count(t => t.tag ==
"Interior");
```

Для того, чтобы данный код работал правильно, его необходимо поместить в заранее созданный в префабе пустой комнаты триггер, который охватывает всю площадь комнаты.

При правильной реализации данного подхода заполнения игровых локаций, можно получить большое количество уникальных комнат в уровне игры. Это может работать для любого (ранее заданного) типа локаций (см. рис. 7).

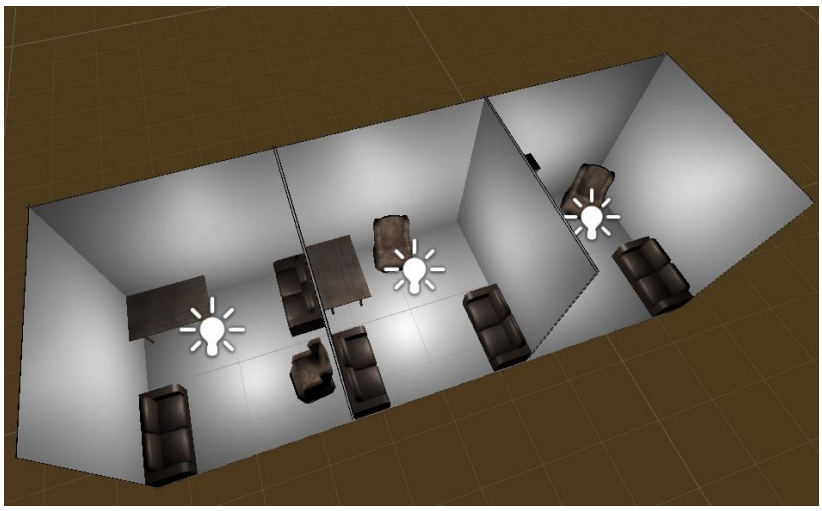

Рисунок 7. Пример работы алгоритма генерации интерьера в жилых комнатах

Таким образом, в статье были рассмотрены методы процедурной генерации для заполнения игровых локаций. Был описан код и подход к созданию уникальных уровней в игре, которая разработана в Unity.

Библиографический список

1. Фёдоров К.Б. Процедурная генерация ландшафтов // Сборник избранных статей научной сессии ТУСУР. 2018. № 1-3. С. 188-190.
2. Третьяков А.А. Процедурная генерация массивной 3d-геометрии с использованием улучшенного алгоритма Marching Cubes // Южно-Сибирский научный вестник. 2021. № 4 (38). С. 8-15.
3. Никитин К.В., Комаров А.А., Боргояков Е.Э. Игровой баланс при использовании процедурной генерации игрового содержимого // Сборник избранных статей научной сессии ТУСУР. 2021. № 1-2. С. 248-250.
4. Шубин А.В., Козар Б.А. Процедурная генерация геометрических фигур при помощи среды разработки видеоигр Unity3D // Modern Science. 2022. № 1-1. С. 430-434.
5. Unity - Manual: Unity User Manual 2020.3 (LTS) URL: https://docs.unity3d.com/Manual/index.html. - Дата доступа: 29.01.2022.