


| | |
|---|--|
|  | <p align="center"> Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) </p> |
|---|--|

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация

КАФЕДРА Системы автоматизированного проектирования

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

| | |
|----------------------|---|
| Студент | Эвоян Эрик Боррисович |
| Группа | РК6-72Б |
| Тип практики | эксплуатационная |
| Название предприятия | «НИИ Автоматизации Производственных Процессов МГТУ им. Н.Э. Баумана» |

| | | |
|-----------------------|----------------------|---|
| Студент | _____ | <u>Эвоян Э. Б.</u> фамилия, и.о. |
| | <i>подпись, дата</i> | |
| Руководитель практики | _____ | <u>Витюков Ф.А.</u> фамилия, и.о. |
| | <i>подпись, дата</i> | |

Оценка: _____

Москва, 2022 г.

**«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ
Заведующий кафедрой _____

« ____ » _____ 20 ____ г.

З А Д А Н И Е
на прохождение производственной практики
эксплуатационная
_____ тип практики

Студент

_____ Эвоян Эрик Борисович _____ 4 курса группы РК6-72Б
Фамилия Имя Отчество № курса индекс группы

в период с 1 . 07 . 2022 г. по 31 . 08 . 2022 г.

Предприятие: «НИИ Автоматизации Производственных Процессов МГТУ им. Н.Э. Баумана»

Подразделение: _____
(отдел/сектор/цех)

Руководитель практики от предприятия (наставник):

_____ Витюков Федор Андреевич _____
(Фамилия Имя Отчество полностью, должность)

Руководитель практики от кафедры:

_____ Оглоблин Дмитрий Игоревич _____
(Фамилия Имя Отчество полностью, должность)

Задание:

Во время прохождения производственной практики студент должен:

1. Разработать все необходимые классы для реализации случайного графа;
2. Добавить возможность хождения кубов по сетке графа;
3. Разработать все необходимые класс для визуализации гранулярного фильтра

Дата выдачи задания « 1 » августа 2022 г.

Руководитель практики от предприятия _____ / Витюков Ф.А /
Руководитель практики от кафедры _____ / Оглоблин Д.И. /
Студент _____ / Эвоян Э. Б. /

Оглавление

| | |
|---|----|
| Введение (первая задача) | 3 |
| Обзор исходного кода (первая задача)..... | 4 |
| Результаты работы программы (первая задача) | 9 |
| Введение (вторая задача) | 12 |
| Обзор исходного кода (вторая задача) | 12 |
| Результаты работы программы (вторая задача)..... | 17 |
| Заключение | 19 |
| СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ | 20 |

Введение (первая задача)

Для решения задачи, необходимо установить движок UE4, а также Visual Studio, которую движок использует в качестве основной среды разработки. Затем необходимо создать проект внутри стартового окна Unreal Engine. После создания проекта можно редактировать уровень в визуальном редакторе, создавать классы визуальных скриптов Blueprints.

Для разработки использовалась Visual Studio 2017 и Unreal Engine версии 4.27

При создании проекта использовался следующий шаблон:

- 1) Game > Blank
- 2) C++, Maximum Quality, Raytracing Disabled, Desktop/Console, With Starter Content

Обзор исходного кода (первая задача)

Помимо стандартного кода, генерируемого при создании проекта, в проекте присутствуют следующие файлы, реализующие функционал для конкретной задачи:

`NetActor(.h/.cpp)` – файлы содержат объявление и описание класса `ANetActor` и его методов. Класс `NetActor` реализует функционал визуализации сетки, а также управление созданием объектов, перемещающихся по ней.

`NetGenerator(.h/.cpp)` – здесь объявляется и реализуется класс `NetGenerator`, который содержит набор алгоритмов для генерации сетки, а также реализует хранение этих данных.

`WalkerActor(.h/.cpp)` – здесь объявляется и реализуется класс `AWalkerActor`, который отвечает за поведение и анимацию объекта, перемещающегося по сетке.

NetActor.h

В данном файле объявляется класс сетки, который наследуется от класса `AActor`. В этом классе объявляется набор полей, которые могут быть изменены из визуального редактора UE4.

Список полей:

1. `NetWidth` – ширина сетки в юнитах.
2. `NetHeight` - высота сетки в юнитах.
3. `NetWidthPointCount` – количество точек сетки по ширине.
4. `NetHeightPointCount` - количество точек сетки по высоте.
5. `MaxOffsetX` – максимальное смещение точки сетки по X при генерации.
6. `MaxOffsetY` - максимальное смещение точки сетки по Y при генерации.
7. `MaxOffsetZ` - максимальное смещение точки сетки по Z при генерации.
8. `AdditionLines` – вероятность добавления дополнительного ребра к остову графа (для каждого ребра).

9. `CrossWays` – если `true`, то в графе будут создаваться ребра, лежащие на обоих диагоналях квадрата, состоящего из соседних четырех вершин графа, составляющих квадрат. Если `false`, то будет создаваться только одна диагональ.
10. `LineWidth` – ширина трубок, составляющих сетку.
11. `LineHeight` – высота трубок, составляющих сетку.
12. `NumberOfWalkers` – количество объектов, которые будут располагаться в узлах сетки.
13. `Velocity` – скорость объектов в юнит/сек

NetActor.cpp

В данном файле находятся определения следующих методов:

В конструкторе класса `ANetActor` происходит инициализация объекта-генератора инстансов кубов, которые будучи трансформированными создают визуальное отображение сетки.

`void generate()`- инициализирует генератор сетки.

`void createWalkers()` – создаёт объекты (кубы) которые будут анимироваться на графе.

`void draw ()` – отрисовка линий сетки

Помимо этого, переопределяются два унаследованных метода:

В методе `void PostRegisterAllComponents()` вызывается метод `generate` и `draw()`. Этот метод нужен для отрисовки сетки в визуальном редакторе Unreal Engine 4 на этапе настройки параметров объекта.

`void BeginPlay()` – метод, в котором вызываются методы `generate()`, `createWalkers()` и `draw()`.

NetGenerator.h

В этом файле объявляются три класса.

NetLine – класс, хранящий вершины отрезка.

NetNode – структура, хранящая данные вершины графа, необходимые в процессе работы алгоритма Дейкстры.

NetGenerator – класс-генератор сетки. Хранит список вершин графа с координатами NetPoints и список инцидентности NetIndices.

NetGraph.cpp

В этом файле реализуются следующие методы класса NetGenerator:

В конструкторе NetGraph происходит генерация вершин и ребер остова графа по алгоритму Прима и добавление дополнительных ребер.

std::vector<NetLine> getLines() - генерирует список ребер графа с координатами вершин каждого ребра в виде массива отрезков.

std::vector<FVector> getPoints() - возвращает список координат вершин графа.

Метод std::vector<NetLine> getPath(int start_ind, int end_ind) генерирует список отрезков, составляющих кратчайший маршрут из точки start_ind до точки end_ind вдоль существующих ребер графа с использованием алгоритма Дейкстры.

WalkerActor.h

Содержит объявление класса AWalkerActor, унаследованного от класса Aactor. Класс содержит следующие значимые поля:

NetGenerator* Generator — ссылка на генератор, получаемый откласса NetActor.

std::vector<NetLine> LinesList;— список отрезков маршрута объекта вдоль ребер сетки.

int start_point, end_point - индексы начальной и конечной точек.

bool path_created — true, если путь сгенерирован, false, если нет.

int cur_line_index - индекс текущего отрезка в списке LinesList, на котором в данный момент находится объект.

FVector cur_pos, next_pos — индекс текущей и следующей вершины графа, между которыми находится объект.

float velocity — скорость объекта в юнит/сек.

FLinearColor line_color — цвет линий пути объекта.

Класс содержит также набор set методов для некоторых полей.

WalkerActor.cpp

В этом файле реализуются следующие методы класса AWalkerActor:

В конструкторе класса AWalkerActor происходит инициализация статического меша (куба) для визуализации объекта и объекта для отрисовки линий UlineBatchComponent.

void createPath() - генерирует следующую конечную точку для объекта и получает список отрезков пути с помощью объекта Generator класса NetGenerator.

Переопределение унаследованного метода void Tick(floatDeltaTime) — в этом методе вычисляется текущая позиция объекта.

Результаты работы программы (первая задача)



Рис. 1 Остовный граф. Рендеринг линий во viewport редактора до запуска.

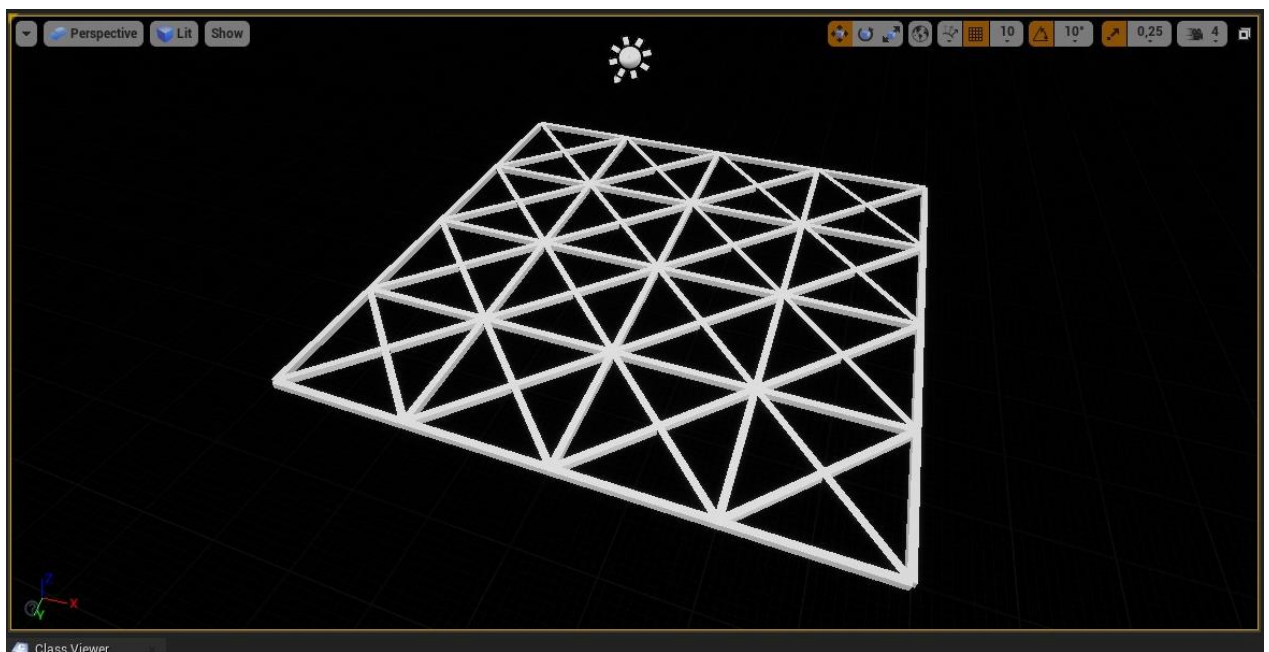


Рис. 2 Полный граф (CrossWays on).

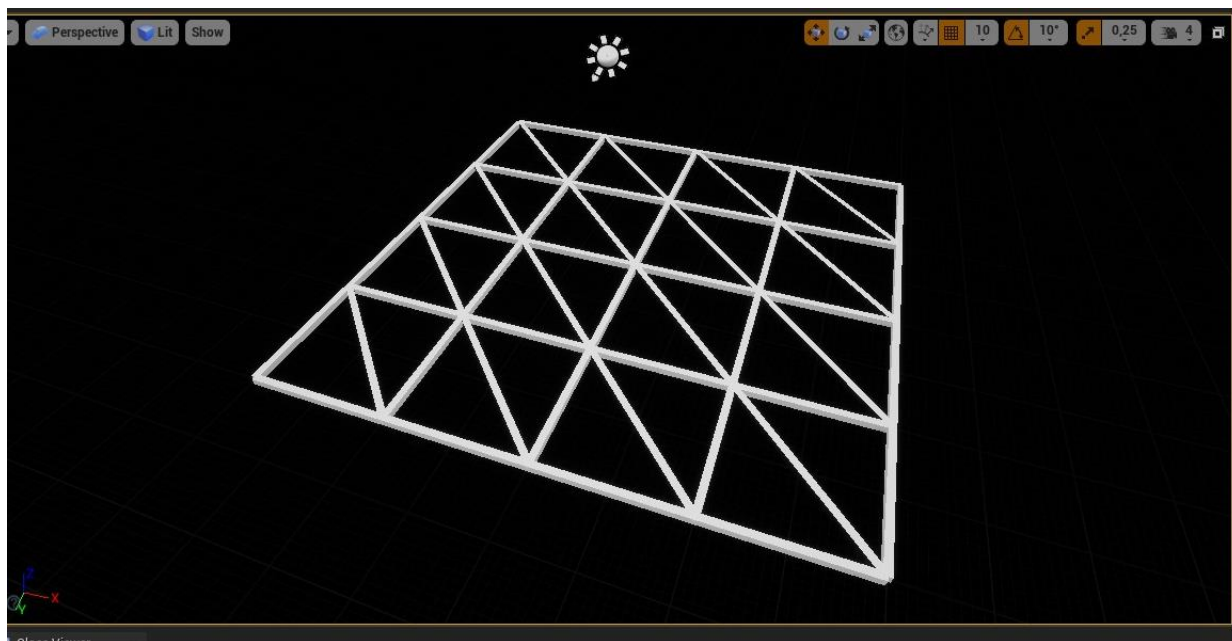


Рис. 3 Полный граф(CrossWays off).

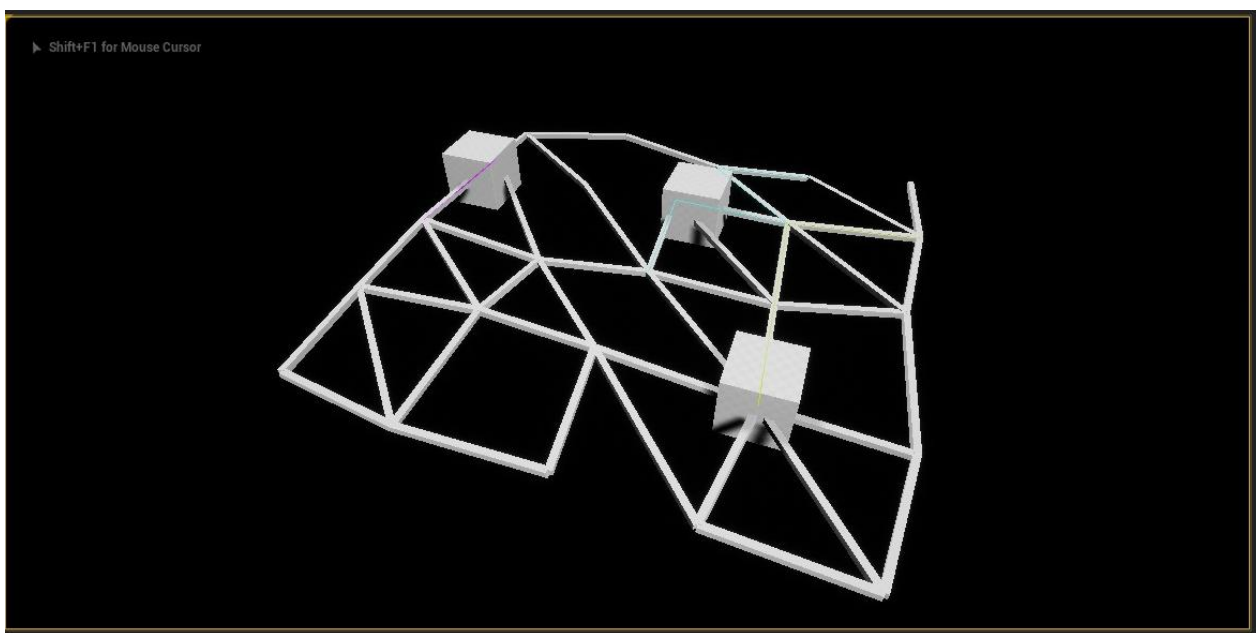


Рис. 4 Граф со смещениями узлов и частичным заполнением. Движение объектов после запуска.

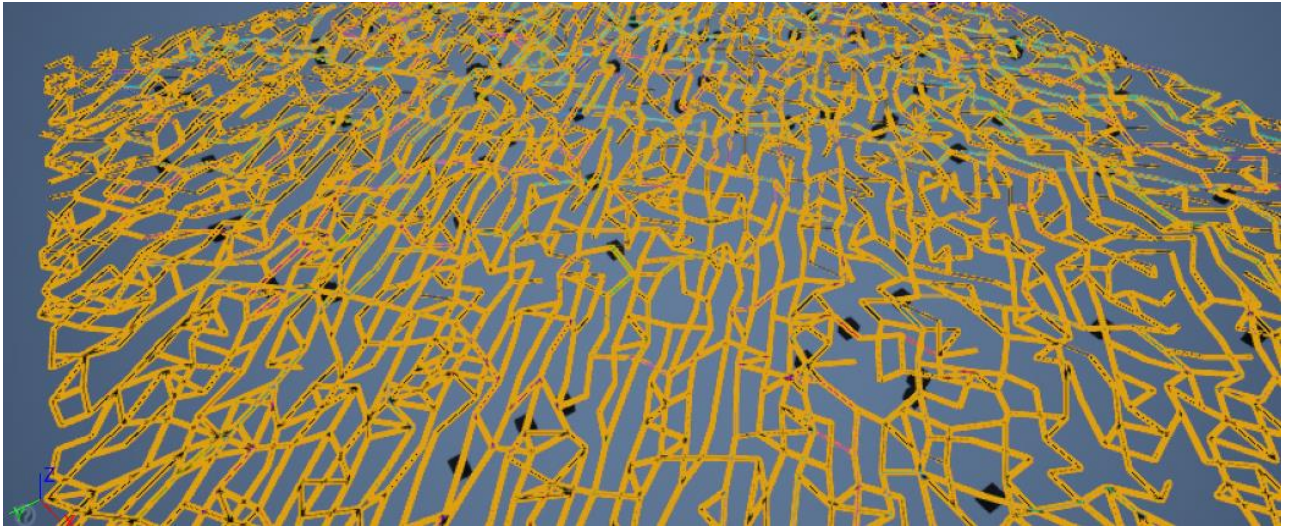


Рис. 5 Большой граф со 100 объектами.

Введение (вторая задача)

Для решения задачи, необходимо установить движок UE4, а также Visual Studio, которую движок использует в качестве основной среды разработки. Затем необходимо создать проект внутри стартового окна Unreal Engine. После создания проекта можно редактировать уровень в визуальном редакторе, создавать классы визуальных скриптов Blueprints.

Для разработки использовалась Visual Studio 2017 и Unreal Engine версии 4.27

При создании проекта использовался следующий шаблон:

- 1) Game > Blank
- 2) C++, Scalable 2D/3D Graphics, Raytracing Disabled, Desktop/Console, No Starter Content

Обзор исходного кода (вторая задача)

Рассматривать весь код целиком будет достаточно трудоемко, поэтому здесь приведён основной обзор структуры проекта.

Необходимо рассмотреть общее взаимодействие компонентов проекта.

Главный функционал представляют 5 классов, унаследованных от **AActor**. Это классы **ACylinderFilter**, **ASphereFactory**, **AWaterGenerator**, **AWaterActor**, **ASphereActor**.

ACylinderFilter – класс фильтра, в качестве главного компонента содержит компонент сцены и вложенного в него компонента процедурного меша (**UProceduralMeshComponent**). В момент инициализации объекта класса **ACylinderFilter** создается цилиндр без верхней крышки с утолщенными стенками. Цилиндр состоит из следующих частей: боковые внешняя и внутренняя стенки, нижняя и верхняя грани, ободок между внешней и внутренней стенками на верхней стороне цилиндра. Все эти части генерируются из треугольников и четырехугольников (четырёхугольники состоят из двух треугольников с индексами для четырех вершин четырехугольников). Для упрощения работы алгоритма для каждого треугольника и четырехугольника генерируются свои наборы вершин, не связанные через индексный буфер с другими треугольниками/четырёхугольниками.

Параметры цилиндра задаются в визуальном редакторе основываясь на следующих свойствах (**UPROPERTY**):


```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"GeometrySettings")
```

```
float InnerCylinderD = 300.0f; // внутренний диаметр
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"GeometrySettings")
```

```
float OuterCylinderD = 360.0f; ; // внешний диаметр
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"GeometrySettings")
```

```
float CylinderH = 300.0f; // высота цилиндра
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"GeometrySettings")
```

```
float CylinderDelta = 60.0f; // расстояние между нижними  
внутренней и внешней гранью
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"GeometrySettings")
```

```
int RadialVertices = 16; // количество вершин, аппроксимирующих  
окружность в  
основании цилиндра
```

ASphereFactory – класс генератора гранул для фильтра. Объект класса содержит компонент сцены и имеет свое положение на уровне, относительно которого генерируются шарики, симулирующие жидкость. Шарики генерируются с помощью метода **SpawnActor** класса **UWorld**. Им задается трансформация относительно компонента сцены объекта генератора и для компонентов **UPrimitiveComponent** каждого шарика поле, отвечающее за симуляцию физики устанавливается значение **true**. В момент, удаления лишних шариков, объект проходится по списку фильтров и удаляет все шарики, которые не находятся хотя бы в одном из них (проверка происходит на базе алгоритма, который вычисляет, находится ли точка центра шарика внутри одного из треугольников нижней поверхности цилиндра + между верхней и нижней гранью цилиндра, алгоритм имеет недостаток — необходимо, чтобы цилиндр располагался строго вертикально, либо можно вычислять положение центра шарика в пересчете на систему координат цилиндра, однако сейчас вычисление происходит в глобальных координатах), при этом для всех шариков, оказавшихся внутри фильтра отключается симуляция физики.

Параметры генератора задаются в визуальном редакторе основываясь на следующих свойствах (**UPROPERTY**):

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"SpawnSettings")  
float SpawnDensity = 100.0f; // расстояние между генерируемыми  
шариками
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"SpawnSettings")  
float SpawnOffsetsTolerance = 5.0f; // диапазон случайного  
смещения шарика  
относительно начального положения спауна
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"SpawnSettings")  
int SpawnBoxElementsCount = 5; // число шариков по ребру облака  
шариков в форме куба
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"SpawnSettings")  
float SphereDiameter = 75.0f; // начальный диаметр шариков
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"SpawnSettings")  
float SphereDiameterTolerance = 20.0f; // диапазон случайного  
изменения начального размера шарика
```

AWaterGenerator — класс генератор частиц жидкости. На основе заданных параметров (по аналогии с **ACylinderFilter**) генерирует тонкостенный цилиндр без верхней и нижней граней. Причем нормали треугольников, составляющих боковую грань, вывернуты внутрь для корректной работы коллизии. По аналогии с **ASphereFactory** в определенный момент начинает генерировать шарики, но в отличие от последнего шарики генерируются по одному слою в некоторый промежуток времени, причем

один слой шариков имеет радиальную форму. До наступления момента начала генерации шариков, у всех объектов данного класса отключена видимость и коллизия, чтобы не мешать засыпанию гранул в фильтры.

Параметры генератора жидкости задаются в визуальном редакторе основываясь на следующих свойствах (**UPROPERTY**):

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float CylinderD = 300.0f; // диаметр цилиндра
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float CylinderH = 600.0f; // высота цилиндра
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float CylinderDelta = 30.0f; // расстояние между локальным началом  
координат и  
нижней части цилиндра
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    int RadialVertices = 16; // количество вершин, аппроксимирующих  
окружность в  
основании цилиндра и задающих количество шариков (вдоль окружности) в  
одном слое
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float SphereDiameter = 10.0f; // начальный диаметр шариков
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float SphereDiameterTolerance = 60.0f; // диапазон случайного  
изменения размера  
шариков (только в большую сторону)
```



```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float SpawnTime = 5.0f; // интервал времени, в течении которого  
происходит генерация шариков
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float SpawnEvery = 0.3f; // промежуток времени между генерацией  
слоев шариков
```

```
UPROPERTY(EditAnywhere, BlueprintReadWrite, Category =  
"WaterSpawnerSettings")  
    float SpawnDensity = 60.0f; // расстояние (вдоль радиуса) между  
шариками
```

Классы **AWaterActor** и **ASphereActor** практически идентичны между собой и используются классов, на основе которых инстанцируются гранулы и частицы жидкости.

События удаления выкатившихся гранул и начала генерации частиц жидкости обрабатываются по нажатию на клавиши **P** и **R** соответственно.

Так же в проекте настроена сцена, на которой уже размещены все необходимые объекты и настроены их параметры, помимо этого настроено отображение подсказок в виде элементов интерфейса и настроены материалы для объектов.

Результаты работы программы (вторая задача)

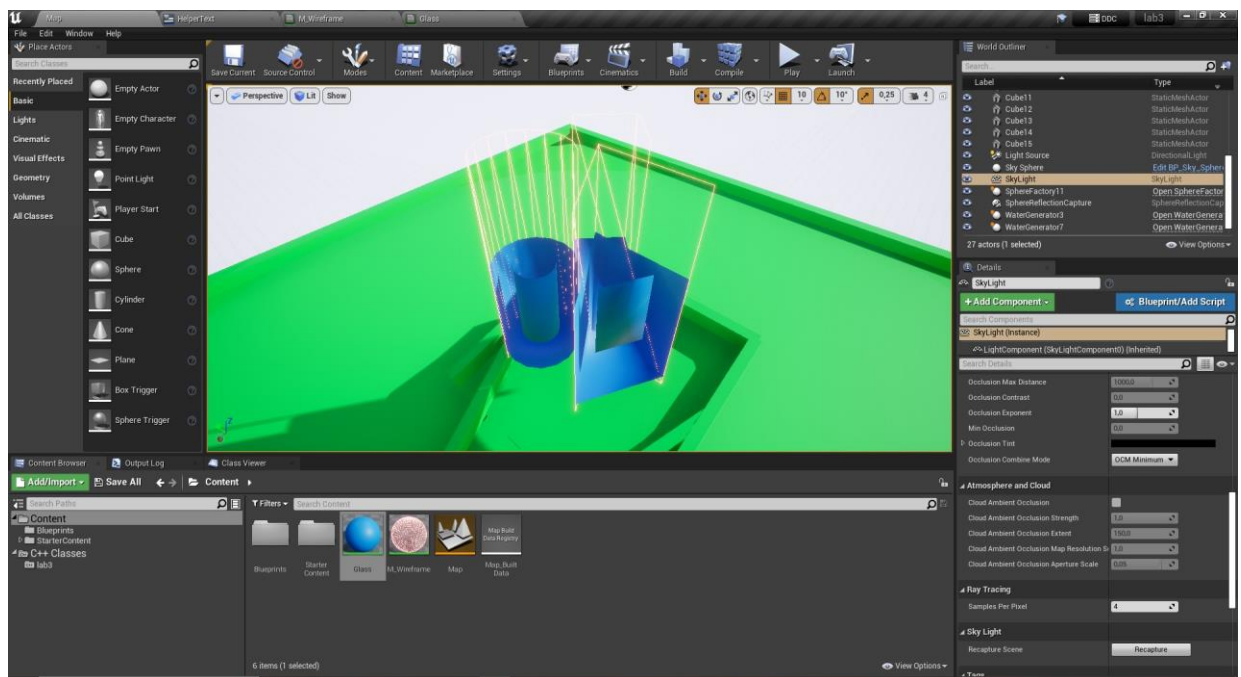


Рисунок 1. Сцена до запуска с размещенными объектами.

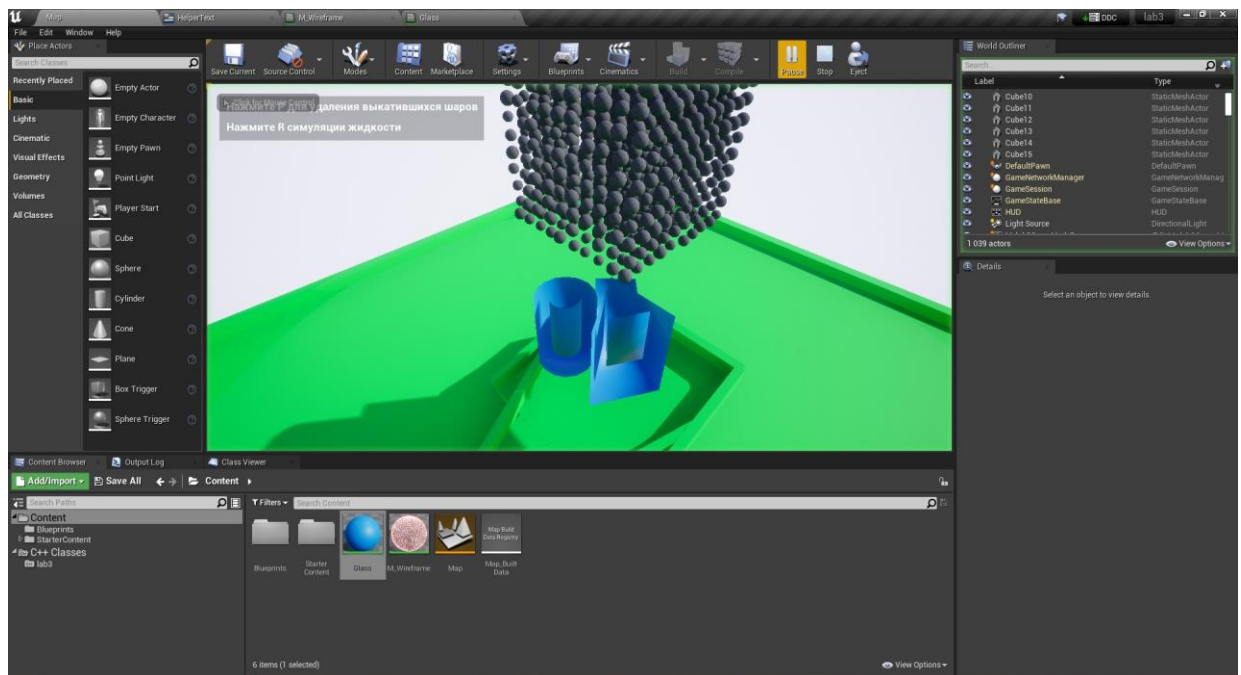


Рисунок 2. Генерация гранул.

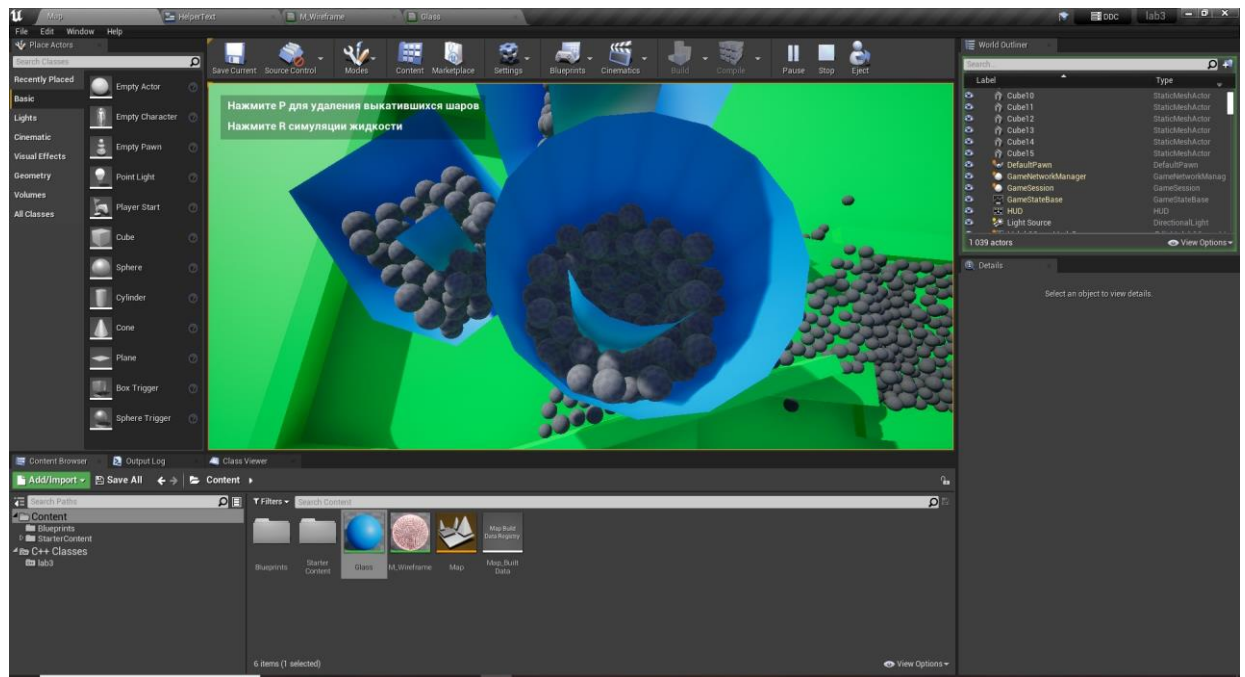


Рисунок 3. Гранулы засыпались в фильтры.

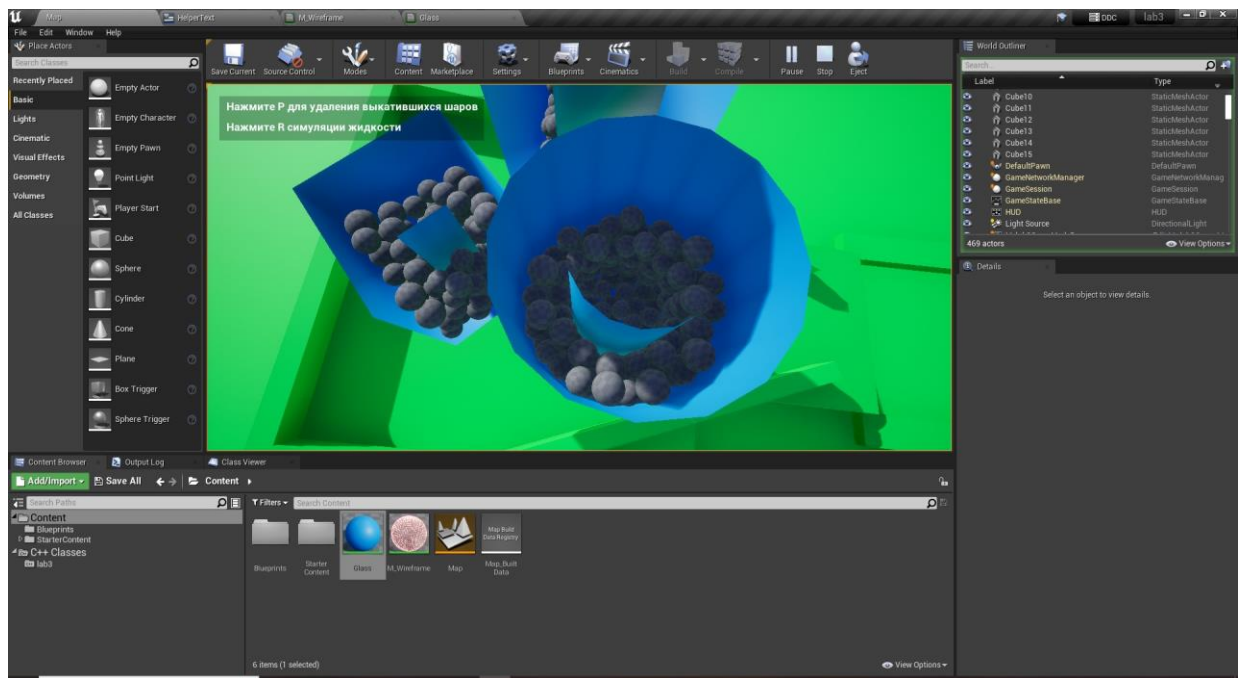


Рисунок 4. Удаление лишних гранул.

Заключение

В процессе прохождения практики были получены навыки работы с Unreal Engine 4, получен опыт разработки на языке C++; улучшено представление об организации проектов.

В результате были решены следующие задачи: были реализованы отрисовка случайного графа и хождение по нему кубов, движущихся по алгоритму, Прима. Была визуализирована симуляция гранулярного фильтра.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Unreal Engine 4 Documentation // Unreal Engine Documentation URL: <https://docs.unrealengine.com/>. Дата обращения: 1.07.2021;
2. Display aspect ratio // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Display_aspect_ratio/. Дата обращения: 07.08.2021;
3. Geometry instancing // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Geometry_instancing. Дата обращения: 12.07.2022.