Оглавление

Цель работы

Ознакомиться с синтаксисом базовых функций Unreal Engine 4. Лабораторная работа предполагает создание и визуализацию графа, а также визуализацию движения объектов по графу.

Задание

В среде визуализации трёхмерной графики на основе Unreal Engine 4, реализовать

a) визуализацию движущегося по графу объекта.

b) Для этого задачу декомпозировать на следующие:

c) Генерация графа: вершины графа – точки регулярной прямоугольной сетки в плоскости XY. Для каждой точки задаются предел смещения disperce, значения которого меньше половины шага сетки. В программе граф хранится в удобной для разработчика форме;

d) Реализация алгоритма Дейкстры (или A*);

e) Движение объекта по графу.

Требования:

- Объект при старте программы появляется в одной из вершин графа.
- Выбирается конечная точка «путешествия» для объекта на графе.
- Ищется кратчайший путь к этой точке.
- Объект продолжает движение до достижения цели.
- Выбирается новая точка назначения.
- Объект путешествует по графу бесконечно.
- По графу может «путешествовать» несколько объектов. Объекты могут проходить сквозь друг друга, не представляя препятствий для движения.
- Объект представляется шаром.
- Граф рисуется в виде обычных линий чёрного цвета.

- Найденный кратчайший путь рисуется с помощью линий заранее выбранных цветов, отличных от чёрного и каждое ребро пути заметно отличается от соседних по цвету
- В настроечном файле должны быть доступны следующие параметры:
    - SizeX и SizeY — Количество вершин по вертикали и горизонтали
    - Disperce — предел смещения вершины
    - Walkers — Количество объектов (Далее волкеры), движущихся по сетке

Вводная часть

В данной лабораторной работе нам предстояло ознакомиться с синтаксисом базовых функций движка Unreal Engine 4 и научиться создавать в нём движущиеся объекты.

Unreal Engine — игровой движок, разрабатываемый и поддерживаемый компанией Epic Games. Хотя движок первоначально был предназначен для разработки шутеров от первого лица, его последующие версии успешно применялись в играх самых различных жанров, в том числе стелс-играх, файтингах и массовых многопользовательских ролевых онлайн-играх.

Для реализации поиска кратчайшего пути был выбран алгоритм Дейкстры.

Разбор реализации

Алгоритм генерации графа

Для генерации графа был использован следующий алгоритм: вершины в первом ряду соединяются друг с другом. После чего происходит проход по всем вершинам графа слева направо сверху вниз. С каждой вершиной производится следующее:

1) Проверяется соединена ли вершина с какой-нибудь вершиной. Если не соединена, то соединяется с вершиной сверху. Верхние вершины уже однозначно соединены, поэтому случай без верхних вершин не обрабатывается

2) Случайно выбираются рёбра, соединяющие данную вершину с соседними. Для вершин в нечётном ряду рёбра выбираются из набора рёбер 0,1,3 (См. Рисунок 1), для врешин в чётном из 1,2,3 (См. Рисунок 1) если данные рёбра провести возможно.

3) Выбранные рёбра рисуются (добавляются в представление графа)

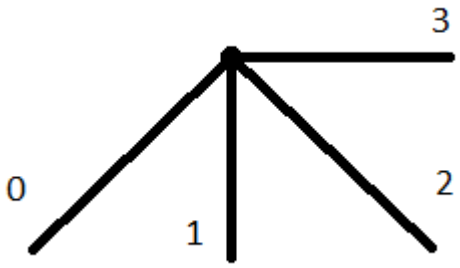

Рисунок 1 — нумерация возможных для рисования рёбер.

Такой алгоритм гарантирует построение связного графа без самопересечений, при этом давая довольно случайные результаты.

Отдельно стоит заметить, что при отображении графа он (или камера) может быть повёрнут, из-за чего может показаться, что использовался другой алгоритм.

Объект ACSphere:

Актор для сферического Instanced объекта. Используется только для наследования другими объектами.

Листинг заголовочного файла:

```
UCLASS()
class LAB2_KOZLOV_API ACSphere : public AActor
{
	GENERATED_BODY()
public:
	ACSphere();
		bool SetPosition(float x, float y, float z);
		bool SetPosition(FVector vector);
		void MoveAbsolute(FVector vector);
		void MoveRelative(FVector vector);
		void MoveToPosition();
		void Init(FVector point);
		void AddNewInstance();
		void AddNewInstance(FVector point);
		void SetColor(FLinearColor _color);
		void UpdateColor(FLinearColor _color);
protected:
		UInstancedStaticMeshComponent* Mesh;
		UMaterial* Material;
		float scale = 1.0f;
		FVector position = { 0,0,0 };
		void SetScale(float newScale);
		void UpdateMaterial(FString reference);
		UPROPERTY()
			FLinearColor color = { 0,0,0 };
private:
	UPROPERTY()
		FString MeshReference = "StaticMesh'/Game/StarterContent/Props/MaterialSphere.MaterialSphere'";
		FString MaterialReference = "Material'/Game/StarterContent/Materials/M_Colorized.M_Colorized'";
};
```

Поддерживает:

- задачу позиции актора методом SetPosition,
- относительное и абсолютное перемещение методами MoveRelative и MoveAbsolute соответсвенно
- Добавление нового инстанса актора на позицию актора или со смещением от него методами AddNewInstance
- Задачу и обновления цвета инстансов методами SetColor и UpdateColor соответсвенно

Доп. Информация:

За радиус сферы отвечает параметр scale, наследующие объекты перед отображением могут поменять его для изменения размера инстансов.

Объект ADot:

Объект вершины графа. Унаследован от ACSphere. Т.к. использует инстансы может одним объектом отображать все вершины графа.

Листинг заголовочного файла:

```
UCLASS()
class LAB2_KOZLOV_API ADot : public ACSphere
{
	GENERATED_BODY()

public:
		ADot();
private:
	UPROPERTY()
		float DotScale = 0.1f;

	UPROPERTY()
		FLinearColor DotColor = {1,1,0};
};
```

Структура FEdgeInfo:

Содержит информацию для построения ребра:

- Start — Начальная точка ребра
- End — Конечная точка ребра
- Color — Цвет ребра
- Height — Параметр «Уровня» ребра. Чем выше уровень, тем заметней ребро. В данной реализации означает диаметр цилиндра, используемого для визуализации ребра.

Находится в заголовочном файле объекта CEdge

Объект CEdge:

Объект для ребра графа. Использует Instanced Mesh, поэтому одним объектом может отображать все рёбра графа. В объекте не предусмотрено обновление созданных инстансов, то есть рёбпра «статичны»

Листинг заголовочного файла:

```
USTRUCT()
```

```
struct FEdgeInfo {
	GENERATED_BODY()
	FVector start;
	FVector end;
	FLinearColor color;
	float height = 0.01;
};
UCLASS()
class LAB2_KOZLOV_API ACEdge : public AActor
{
	GENERATED_BODY()
public:
	// Sets default values for this actor's properties
	ACEdge();
		void UpdateMaterial(FString reference);
		void SetStart(FVector _start);
		void SetEnd(FVector _end);
		void SetColor(FLinearColor _color);
		void UpdateColor(FLinearColor _color);
		int DrawStaticEdge(FVector start, FVector finish, FLinearColor color);
		int DrawStaticEdge(FLinearColor color);
		int DrawStaticEdge();
		void DestroyEdge(int instanceId);

protected:
		UInstancedStaticMeshComponent* Mesh;
		UMaterial* Material;
		FVector start;
		FVector end;
		TArray<FEdgeInfo> instanceInfo;
		float height = 0.01;
		float level = 0;
private:
		FLinearColor color;
		FString MeshReference =
"StaticMesh'/Game/StarterContent/Shapes/Shape_Cylinder.Shape_Cylinder'";
		FString MaterialReference =
"Material'/Game/StarterContent/Materials/M_Colorized.M_Colorized'";
```

Объект поддерживает:

- Обновление материала методом UpdateMaterial
- Задание начала и конца для следующего инстанса ребра методами SetStart и SetEnd соответсвенно. Если перед созданием нового не обновить часть параметров, то они будут теми же, что и у прошлого.
- Задание и обновление цвета инстанса методами SetColor и UpdateColor соответсвенно

- Создание нового инстанса с ранее заданными или переданными параметрами методами DrawStaticEdge
- Уничтожение инстанса методом DestroyEdge

Структура FCMatrix:

Структура для создания двумерного массива, так как изначально эта функция не поддерживается средствами Unreal Engine

Объект AGraph:

Объект для хранения и отображения графа. Для отображения используются объекты ADot и ACEdge. Граф хранится в виде матрицы смежности. Каждой точке задаётся её номер (id). Под координатами вершины в графе понимается пара чисел означающих строку и столбец в которых находится вершина. Нумерация столбцов и строк начинается с нуля.

Листинг заголовочного файла:

```
USTRUCT()
struct FCMatrix
{
	GENERATED_BODY()
	TArray<int> arr;
};
UCLASS()
class LAB2_KOZLOV_API AGraph : public AActor
{
	GENERATED_BODY()
public:
	// Sets default values for this actor's properties
	AGraph();
		void Init(int size_x, int size_y, float _disperse);
		FVector GetVertex(int id);
		FVector GetVertex(int x, int y);
		FVector GetCoords(int id);
		int GetSizeX();
		int GetSizeY();
		TArray<FVector> GetVertexes();
		TArray<FCMatrix> GetEdges();
private:
	UPROPERTY()
		int sizeX = 0;
	UPROPERTY()
		int sizeY = 0;
	UPROPERTY()
		int disperse = 0;
		TArray<FVector> vertexes;
		TArray<FCMatrix> edgeMatrix;
```

```
        FLinearColor edgeColor = { 0,0,0 };
        int MakeIDfromCoords(int x, int y);
        void DrawGraph(UWorld* world);
        float GetRandomDisperse(float norm);
        bool IsConnected(int id);
        TArray<int> ChoseOptions(TArray<int> options);
        void GenerateEdges();
        void DrawEdges(UWorld* world);
};
```

Объект поддерживает:

- Инициализацию и отображение графа с заданными параметрами (Количество точек по горизонтали, количество точек по вертикали и предел отклонения методом Init
- Получение координат вершины в мире по её координатам или id в графе методами GetVertex
- Получение координат вершины в графе по id методом GetCoords
- Получения количества точек по горизонтали и вертикали методами GetSizeX и GetSizeY соответсвенно.
- Получение массива с координатами в мире всех вершин графа методом GetVertexes
- Получение графа в виде матрицы смежности методом GetEdges

Объект ADynamicEdge:

Унаследован он ACEdge. Поддерживает обновление созданных инстансов рёбер. Используется для рисования пути волкера.

Листинг заголовочного файла:

```
UCLASS()
class LAB2_KOZLOV_API ADynamicEdge : public ACEdge
{
        GENERATED_BODY()

public:
        ADynamicEdge();

        UFUNCTION()
                void UpdateStart(FVector newStart, int instanceId);

        UFUNCTION()
```

```
		void UpdateEnd(FVector newEnd, int instanceId);

private:
		void UpdateEdge(int instanceId);
};
```

Объект поддерживает:

Обновление инстанса ребра с новым началом или концом методами UpdateStart и UpdateEnd соответсвенно.

Объект AGraphWalker:

Объект волкера. Унаследован от ACSphere. Ищет кратчайший путь методом Дейкстры, рисует к нему путь и движется по нему, параллельно обновляя ребро, по которому движется, чтобы оно «стиралось» за ним и удаляет инстансы пройденных рёбер.

Листинг заголовочного файла:

```
UCLASS()
class LAB2_KOZLOV_API AGraphWalker : public ACSphere
{
	GENERATED_BODY()

public:
	AGraphWalker();

		void SetParams(int _sizeX, int _sizeY, TArray<FVector> graphVertexes, TArray<FCMatrix>
_edges, int startId, int endId, int _speed);

		void FindShortestPath(int id1, int id2);

		void DrawPath();

		virtual void Tick(float DeltaTime) override;

private:
		bool GetShortestPath(int id, int end, int lenLeft);

		TArray<int> GetPathVertexes();

		void UpdateMovingVector();

	UPROPERTY()
		float speed = 1;

		int lastVert;

		int nextVert;
```

```
        int finishVert;

        FVector destination;

        FVector movingVector;

        int sizeX = 0;

        int sizeY = 0;

        TArray<FVector> graphVertexes;

        TArray<FCMatrix> edges;

        TArray<FVector2D> path;

        ADynamicEdge* edgePath;

        UPROPERTY()
            FLinearColor WalkerColor = { 1,0,1 };

        bool came;

        float WalkerScale = 0.2f;
};
```

Объект поддерживает:

- Задание параметров для начала движения (Длина по горизонтали, вертикали, координаты в мире вершин графа, представление графа в виде матрицы смежности, id начала пути, id конца пути и скорость движения)

Доп. Информация:

Метод Дейкстры реализован в методе GetShortestPath. Движение происходит с помощью унаследованной функции Tick. Путь отображается с помощью инстансов объекта ADynamicEdge

Объект UConfigReader:

Объект для чтения простого конфига со строками вида <Ключ>=<Значение>

Листинг заголовочного файла:

```
UCLASS()
class LAB2_KOZLOV_API UConfigReader : public UObject
{
```

```
	GENERATED_BODY()

public:
		static TMap<FString, FString> ReadConfig(FString path);
};
```

Объект поддерживает:

- Чтение конфига из файла по переданному пути методом ReadConfig, объект класса для этого не требуется.

Объект ALab2Scene:

Управляющий объект использующий описанные ранее объекты для выполнения поставленного для работы задания. Не содержит не унаследованных от AActor методов, поэтому предоставляется листинг cpp файла.

Листинг cpp файла:

```
ALab2Scene::ALab2Scene()
{
	PrimaryActorTick.bCanEverTick = false;

	SetRootComponent(CreateDefaultSubobject<USceneComponent>(TEXT("SceneComponent")));
	SetActorLocation(FVector(0, 0, 0));
}

// Called when the game starts or when spawned
void ALab2Scene::BeginPlay()
{
	TMap<FString, FString> cfg = UConfigReader::ReadConfig(FPaths::ProjectDir() + FString("\\Config\\Lab2.ini"));
	//if (cfg.Num() == 0) return;

	UWorld* world = GetWorld();

	AGraph* graph = world->SpawnActor<AGraph>();

	FString* sizeXstr = cfg.Find(L"SizeX");
	FString* sizeYstr = cfg.Find(L"SizeY");
	FString* dispercestr = cfg.Find(L"Disperce");
	FString* walkerCountstr = cfg.Find(L"Walkers");

	int sizeX = sizeXstr == NULL ? FMath::RandRange(3, 6) : FCString::Atoi((TCHAR*)**sizeXstr);
	int sizeY = sizeYstr == NULL ? FMath::RandRange(3, 8) : FCString::Atoi((TCHAR*)**sizeYstr);
	int disperce = dispercestr == NULL ? 35 : FCString::Atoi((TCHAR*)**dispercestr);
	int walkerCount = walkerCountstr == NULL ? FMath::RandRange(1, 3) : FCString::Atoi((TCHAR*)**walkerCountstr);
```

```
	graph->Init(sizeX, sizeY, disperce);

	for (int i = 0; i < walkerCount; ++i) {
		int start = FMath::RandRange(0, sizeX * sizeY - 1);
		int end = start + FMath::RandRange(1, sizeX * sizeY - 3);
		end = end % (sizeX * sizeY);
		AGraphWalker* walker = world->SpawnActor<AGraphWalker>();
		walker->SetParams(graph->GetSizeX(), graph->GetSizeY(), graph->GetVertexes(), graph->GetEdges(), start, end, 1);
	}
}
```

Описание работы: объект пытается прочитать конфиг Lab2.ini в папке Config проекта, после чего заносит прочитанные параметры в переменные, если часть параметров (или сам конфиг) не были найдены, то им задаётся случайное значение в заданном диапазоне. После чего по полученным параметрам создаётся граф, а после него волкеры.

Результаты работы программы

Результатом работы программы является граф SizeX на SizeY. Без пересечений, по которому по разноцветным путям движутся волкеры

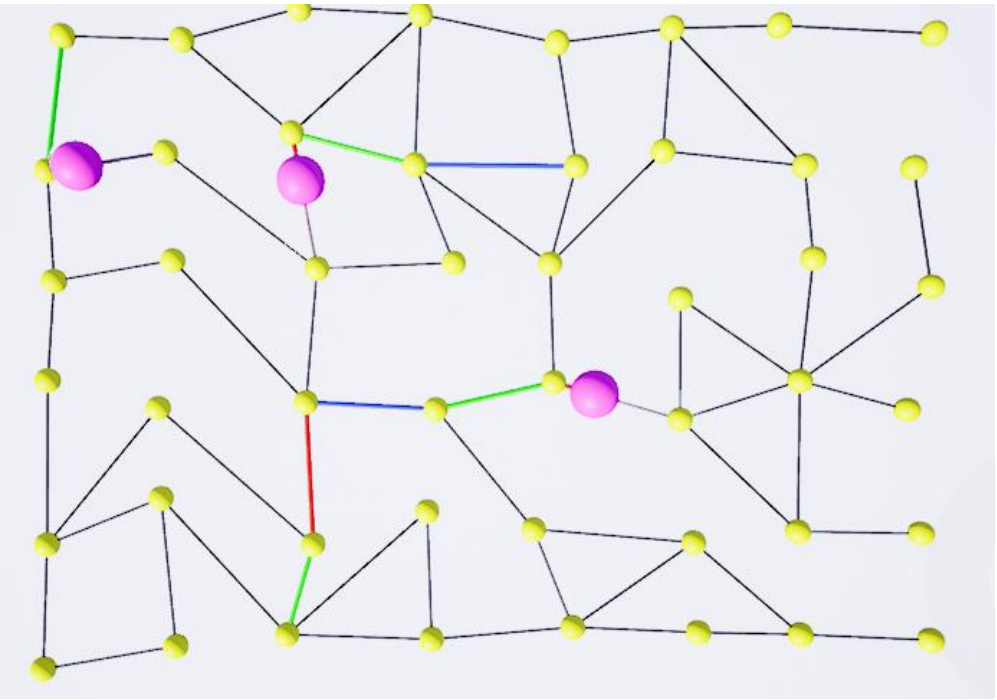

Рисунок 2. Пример объектов на на графе с путями их движения

Выводы

В ходе данной лабораторной работы мы ознакомились с принципами работы с движком UnrealEngine и узнали как в нём создавать движущиеся объекты. Также был разработан и реализован алгоритм построения связного графа без самопересечений, а для поиска кратчайшего пути в построенном графе был реализован алгоритм Дейкстры

В результате лабораторной работы был создан актор, отображающий случайный связный граф с двигающимися по нему объектами.