高级程序设计语言（C++）

课程设计

学 部（院）：电子信息与电气工程学部

专 业：计算机科学与技术

班 级：电计 2002 班

学 生 姓 名：刘运昊

同 组 学 生：郁宏卓 白岩浩

学 号：20201072008

指 导 教 师：王鹏飞

完 成 日 期：2022.3.31

大连理工大学

Dalian University of Technology

目录

基于 Q-Learning 算法的无人机路径规划

一、实验背景和目标

1.1 实验背景

早期的无人机都是按照地面任务规划中心预先计算并设定好的航路飞行，因此一旦在既定航路段出现新的威胁，无人机将束手无策，无人机的智能航线规划无疑成为无人机导航任务中最重要的任务之一。

无人机的航线规划是指在特定约束条件下，寻找从起始点到目标点并满足无人机性能指标的可行且最优航路。规划出导航、满足任务要求、避障安全性等约束的最优航路，对提高无人机的智能航线规划性能有重要意义。

我们设计了一种实际背景，即在无人机巡航线路对应的地面上设置多个无线路由器。无人机通过飞至每个路由器上方来与路由器进行数据传输，同时避开路线中的障碍物。从而实现无人机在自主巡航中与路由器进行数据交互的操作。

图 1 无人机在自主巡航中与路由器进行数据交互

1.2 实验目标

通过对无人机航线规划框架结构的研究以及对静态全局规划和动态局部规划方法的深入探讨，对本次高级程序设计语言(C++)的课程设计实验做如下展望：

我们本次课程设计的目标是使用 Q-Learning 算法进行无人机的路径规划，检测并记录无人机到达每个无线路由器正上方时该无线路由器的信号强度值。通过实现与路由器最大数据速率传输的方式并且避开障碍物的方式，进行无人机的智能航线规划。同时利用本课程讲授的 C++语言面向对象的基本特点，对课程讲解和上机训练的有关内容实践训练。

在代码部分，我们采用了面向对象的程序设计思想。在无人机航线规划框架结构的研究中，定义了无人机类，包含飞行半径、平飞速度、垂直飞行速度、最大最小飞行高度等属性，这是类和对象的体现；在静态全局规划和动态局部规划方法的探讨中，定义了两种环境类障碍物类和路由器类作为算法的仿真环境，实现了继承与多态。

二、相关原理与工具

2.1 相关原理

2.1.1 Q-learning 算法

Q-learning 算法是一种具有 value-based 特性强化学习算法中的算法，定义 $Q(s,a)$ 为在某一时刻的状态 $s(s\in S)$ 下，采取动作 $a(a\in A)$ 动作能够获得收益的期望，环境会根据 agent 的动作反馈相应的回报 reward ，所以算法的主要思想就是将 State 与 Action 构建成一张 Q-table 来存储 Q 值，然后根据 Q 值来选取能够获得最大的收益的动作。

(一) 算法具体过程

1. 初始化 Q 值表，所有值赋为 0。
2. 基于当前 Q 值选取下一步动作，初始状态，Q 值均为 0，无人机有很大选择空间，随机选择下一步动作，随着迭代次数增加，Q 值表更新，无人机将选择更大奖励的动作。
3. 计算奖励，在采取动作 a 后，根据当前状态的动作奖励和上一状态的奖励，使用 Bellman 方程更新上一状态的$Q(s,a)$

$$Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma \max(Q(s',A)) - Q(s,a))$$

其中 $Q(s,a)$ 为状态s和动作a下的 Q 值，$R(s,a)$为状态s和动作a下对应的奖励值，$Q(s',A)$ 为状态 s下采取动作$a(a \in A)$对应的 Q 值构成的集合。

4. 重复 3 中的过程，直到迭代结束，得到最终的 Q 值表。
5. 根据 Q 值表选择无人机的最佳路径。

(二) 算法改进

随着无人机开始学习，我们希望它采取随机动作来探索更多路径。但随着 Q-Table 变得更好，Q 函数会收敛到更一致的 Q 值。这时如果每次都取预期奖励最高的行为去做，那么在训练过程中可能无法探索其他可能的行为，甚至会进入“局部最优”，无法完成。所以，设置系数，使得智能体有一定的概率采取最优行为，也有一定概率随即采取所有可采取的行动。将走过的路径纳入记忆库，避免小范围内的循环。

针对这种问题，我们采取了 ε-greedy 策略，即贪心策略和随机策略两种方式。我们设置无人机在进行动作决策的时候，在概率 ϵ 下采取随机策略，在概率 $1-\epsilon$ 下采取贪心策略，其中 ε 随着时间从 1 不断衰减。这意味着无人机开始的时候会进行完全随机的移动以最大程度地探索状态空间，然后稳定到固定的探索策略。

```
define epsilon, alpha, gamma
initiate Q-table with zeros
observe initial state s
repeat:
       select an action a
          with probability ε select random action a
          else select action a = argmax(Q(s,a'))
       carry out action a
       observe next state s' and reward r
       calculate Q_target = r + gamma*max[Q(s',A)]
       Calculate Q_delta = Q_target - Q(s,a)
       add alpha*Q_delta to Q(s,a)
       s = s'
until terminated
```

图 2 算法伪代码

2.1.2 香农定理

香农定理是在研究信号经过一段距离后如何衰减以及一个给定信号能加载多少数据后得到了一个著名的公式，它描述有限带宽、有随机热噪声信道的最大数据传输速率（或码元速率）与信道带宽、信噪比（信号噪声功率比）之间的关系，以比特每秒（bps）的形式给出一个链路速度的上限。

香农定理指出：在有随机热噪声的信道上传输数据信号时，数据传输速率 Rb 与信道带宽 W、信噪比 S/N 的关系为：

$$R_b = W \times \log_2(1 + S/N)$$

其中，R_b 是可得到的链路速度，W 是链路的带宽，S 是平均信号功率，N 是平均噪声功率，信噪比(S/N)通常用分贝(dB)表示，而分贝数=10×lg（S/N）。

通过香农定理，我们可以衡量无人机在不同的位置与路由器之间的数据传输速率，从而在奖励矩阵中针对无人机和路由器的相对位置设置不同的奖励，用以训练 Q-Table.

2.2 使用工具

本次实验中，我们主要选用的工具有：

1. Visual Studio 2019
   我们主要采用 Visual Studio 2019 来进行代码的编写，IDE 版本为 16.8.6. 使用 ISO C++ 17 标准，操作系统为 Windows 10 x64, 编译器为 msvc
2. Unreal Engine 4
   我们采用 UE4 引擎来对无人机运行环境进行 3D 建模，UE4 是 Epic 公司开发的一款游戏开发引擎。 UE4 是一套完整的构建游戏、模拟和可视化的集成工具，能够满足艺术家的愿景，同时也具备足够的灵活性，可满足不同规模的开发团队需求。
3. Airsim
   我们采用 Airsim API 进行 UE4 引擎中无人机的控制，Airsim 是一款基于 Unreal Engine 构建的无人机、汽车等模拟器的开源平台，并且可以跨平台地通过 PX4 飞行控制器进行仿真控制，在物理和视觉上逼真的模拟环境

使得它成为一款很好的平台。不仅模拟了汽车无人机等动力学模型，甚至对天气效果灯光控制也做出了非常好的模拟。

4. Fastor 张量库

   我们在代码中导入了一个名为 Fastor 的 C++ 张量库，Fastor 是现代 C++ 的基于堆栈的高性能张量（固定多维数组）库，用于在 C++ 中操作多维数组。

三、实验过程

3.1 初始化环境

首先，我们在 Unreal Engine 中为无人机创建了一个环境，该环境我们将其初始化为一个 7x7 的网格，每个网格长 10m，

其次，我们在该环境中，设置了不同的建筑作为障碍物，我们将建筑物的海拔分为三个级别，分别为 60m，150m，250m。其中高度为 60m 的障碍物 2 个，高度为 150m 的障碍物 8 个，高度为 250m 的障碍物 2 个。

最后，我们在该环境中设置五个路由器，路由器在场景中用白色球体表示，无人机的航线必须经过五个路由器，同时应该避开障碍物，以此实现无人机的最优路径。

其中，建筑物和路由器位置都是随机的。

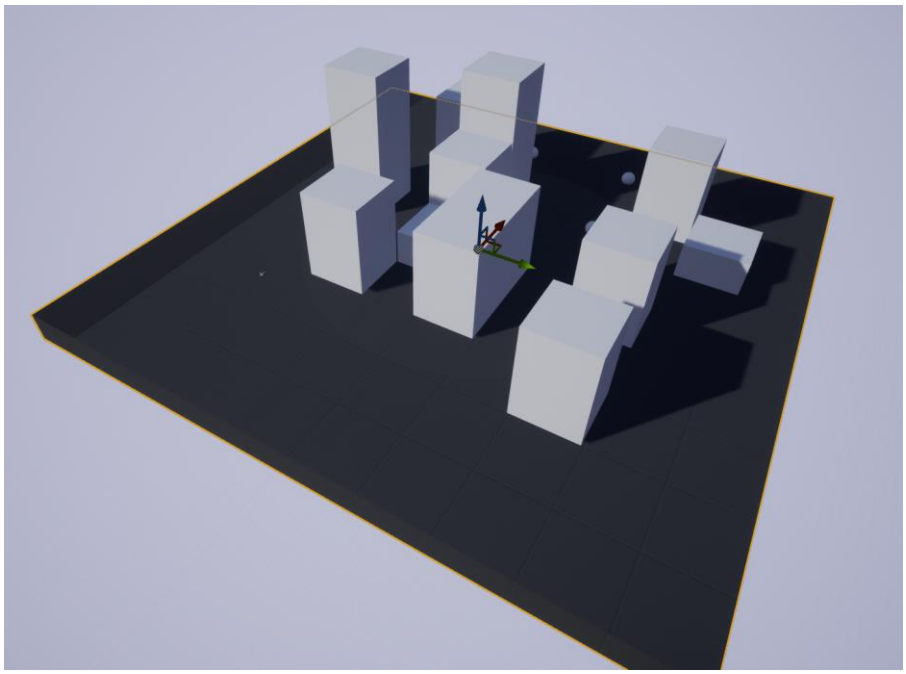

图 3 地图场景

3.2 初始化无人机

在本次实验设计中，我们定义了一台无人机，我们通过定义无人机的燃料以反馈无人机的飞行半径。然后我们初始化了无人机的飞行速度与平飞速度均为 3 米/秒，初始化无人机的垂直飞行速度为 3m/s。同时，我们设置了无人机的三种飞行高度，分别为 50m，125m，200m。最后，我们设置无人机可以执行六种动作，分别为向前行驶，向后行驶，向左行驶，向右行驶，向上行驶，向下行驶。

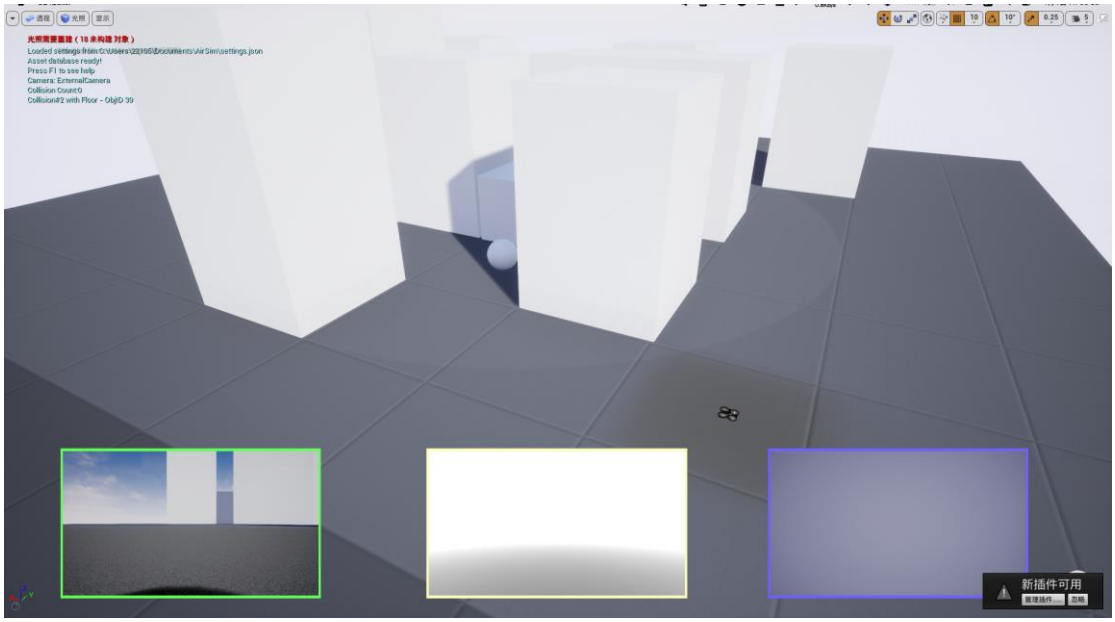

图 4 无人机场景初始化

3.3 训练无人机

为了优化无人机的路径，我们使用 Q-learning 算法。Q-learning 算法使用未来奖励来影响给定状态的当前动作，因此，我们将它作为最大化奖励的算法。

首先我们以无人机的起飞位置作为 Airsim 坐标系的坐标原点，调用 Airsim API 获取每个障碍物和路由器在 Airsim 坐标系中的坐标原点。然后根据环境的不同在奖励矩阵中设置不同的奖励或惩罚。

然后我们设置五个路由器为无人机的飞行目标，为每个目标路由器训练了 2000 次，整个程序训练 10000 次。根据 Q-learning 算法，我们初始化 Q-table，将 Q 值表均赋值为 0 选择一个动作，执行动作测量奖励并在每次测试后结束并更新 Q-table。

四、实验结果

训练结束后，我们根据得到的 Q-Table，使用 Q-learning 算法来进行无人机的路径规划。并通过 Airsim API 控制无人机在 UE4 引擎中进行效果展示，取得了较好的效果

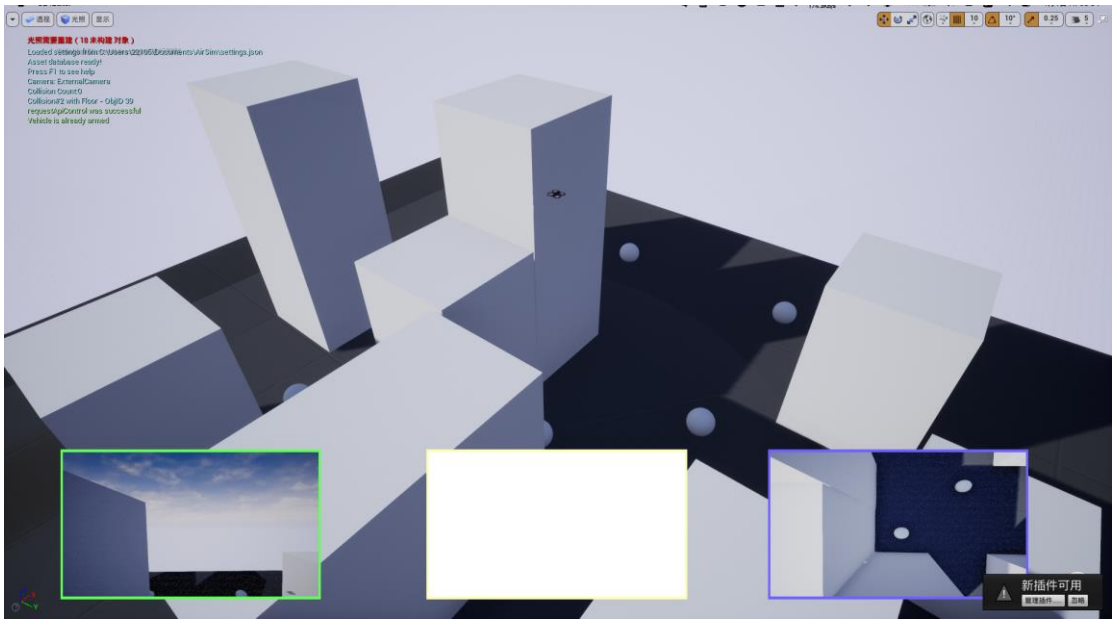

图 5 无人机路径规划效果展示-1

我们接下来更换改变了障碍物的位置，重新进行训练和验证。结果表明，无人机在不同允许环境下都能准确遍历所有路由器，因此该模型具有较好的鲁棒性。

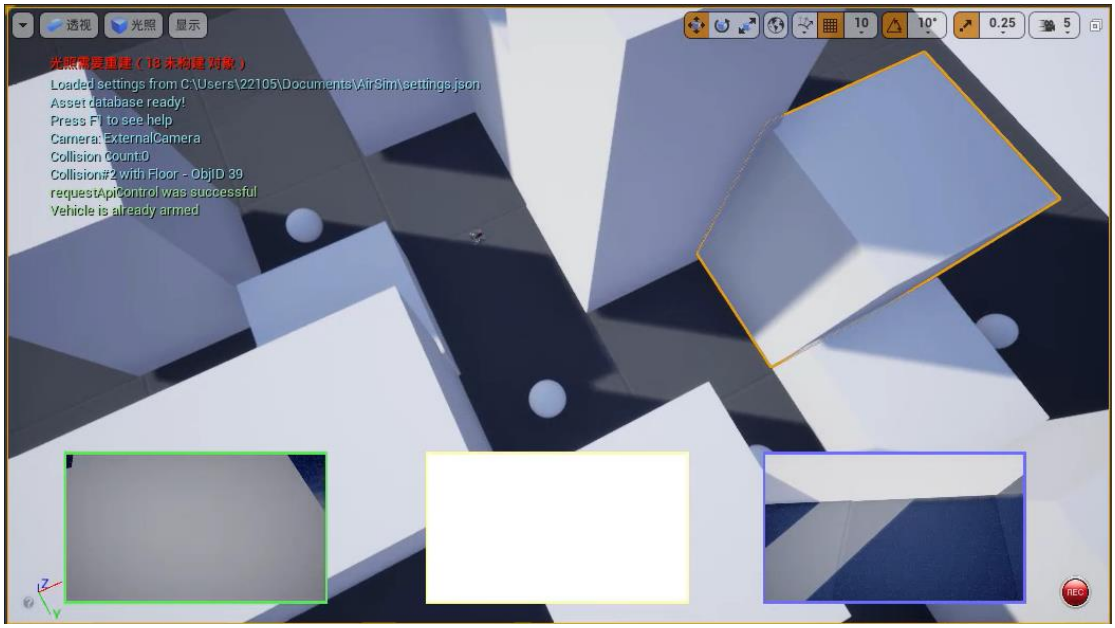

图 6 无人机路径规划效果展示-2

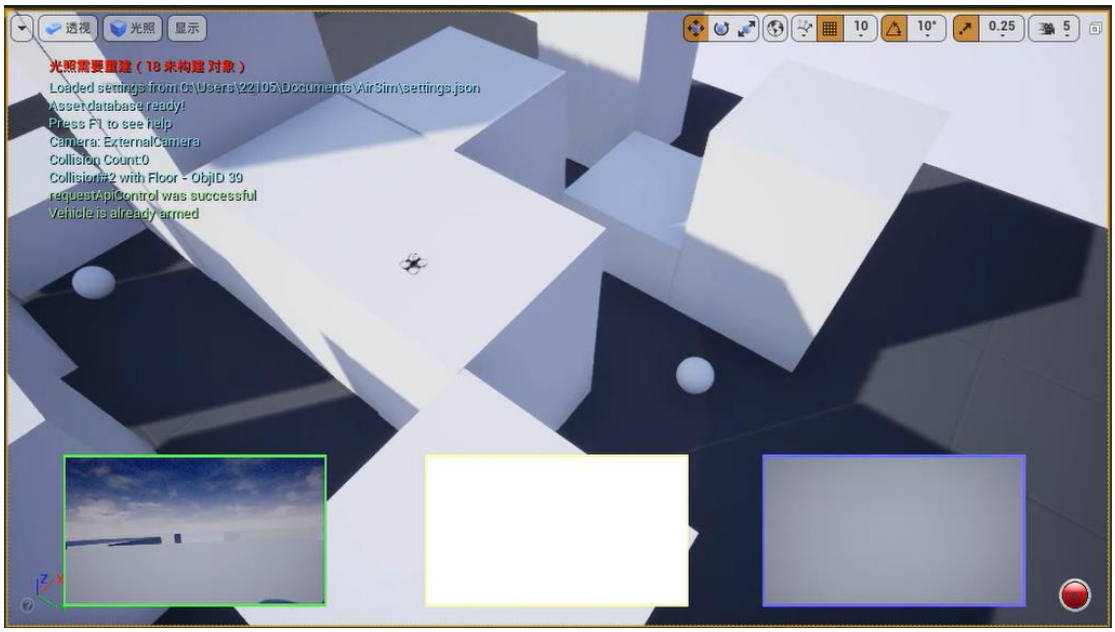

图 7 无人机路径规划效果展示-3

五、实验分析

5.1 起步阶段分析

在课程设计实验的起步阶段，我们力求寻找无人机轨迹优化的较为直观的表现形式，经讨论确定为利用 UE4, AirSim 进行算法仿真环境的搭建和无人机的引入，利用 VS2019 实现二者互联。

5.2 环境搭建阶段分析

在课程设计实验的环境搭建阶段，我们利用 UE4, AirSim 搭建无人机飞行的仿真环境，除了实现环境障碍物的搭建、无人机途经路由器的部署工作等，还给人以良好的视觉感受体验。具体而言：

(1) 对于环境障碍物的搭建，将建筑的海拔划分为 60m, 150m, 250m 三个级别；

(2) 对于途径路由器的部署，随机设置五个路由器作为无人机的航线遍历目标；

对于人机交互的视觉体验，利用 UE4, AirSim 的系统展示可直接观察仿真环境。

5.3 代码编写阶段分析

在课程设计实验的代码编写阶段，我们积极查阅资料、通力协作，解决了起步阶段代码编写的困难；同时，利用 Fastor 张量库定义三维动态数组。并且将各个类的定义和实现分开，分别放入头文件和源文件中，提升了代码的可读性。

5.4 实验结果分析

针对实验结果，基于 Q-Learning 算法，我们实现了无人机的智能航线部署：无人机自行选择最优路径，遍历五个路由器并避开障碍物，从起点飞行到终点，与实验目标达成一致。

除此以外，当我们改变障碍物和路由器的位置后，均能够较好地实现航线的最优规划，因此我们的模型具有较强的鲁棒性，可以适用于不同的环境。这对现实生活中的无人机-路由器通讯具有一定实用意义。

六、小组分工

在小组中，我主要负责使用 UE4 引擎搭建无人机的 3D 场景，使用 Airsim 无人机 API 来进行无人机与 UE4 场景的交互。编写无人机类 Drone，奖励矩阵类 RewardMatrix, Q-Table 类 QMatrix，主函数中的训练函数和测试函数。最后对项目中所有的代码进行风格上的统一和重构，向代码中添加了所有注释，并编写了 README 文件.

6.1 UE4 引擎的搭建

由于本次实验是基于 Q-Learning 算法的无人机路径规划，因此我针对无人机的具体场景使用了 Unreal Engine 4 进行场景搭建，选取的是空白模板

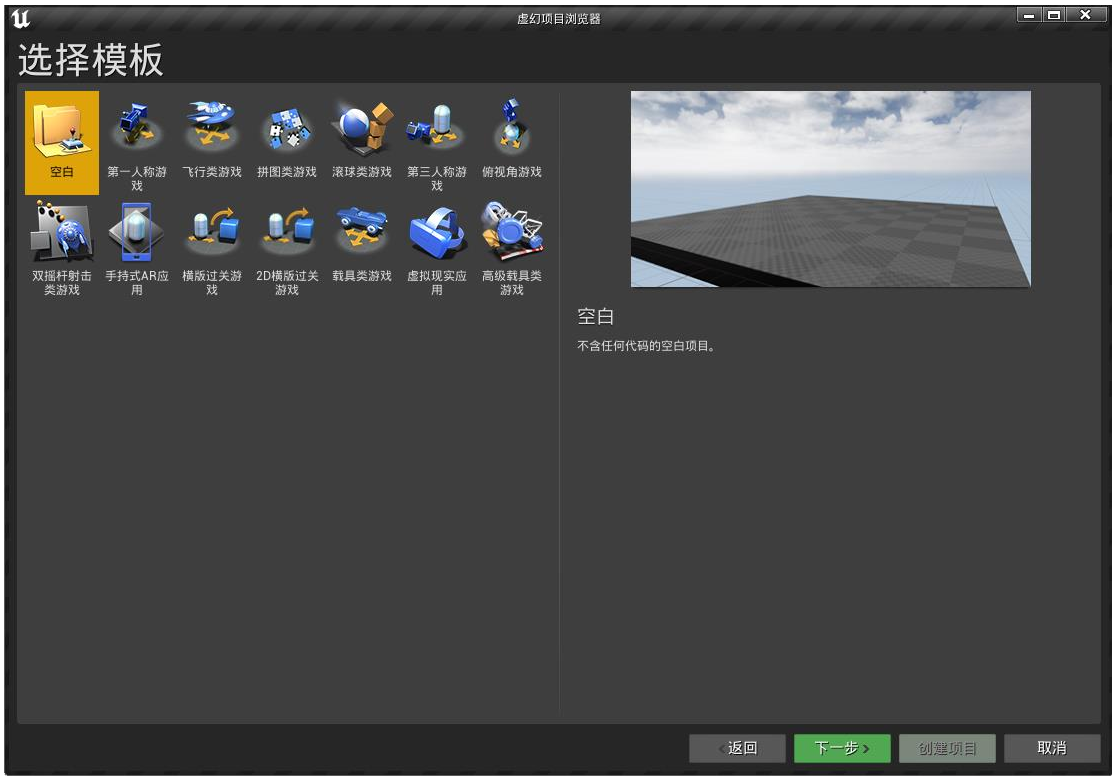

图 8 UE4 引擎模板选择

在 UE4 引擎中我在 BluePrint 中改变了默认的材质，将单个网格的长度设为 10m，将无人机飞行的地图设置为 7 x 7 网格，便于后续的建模和计算。

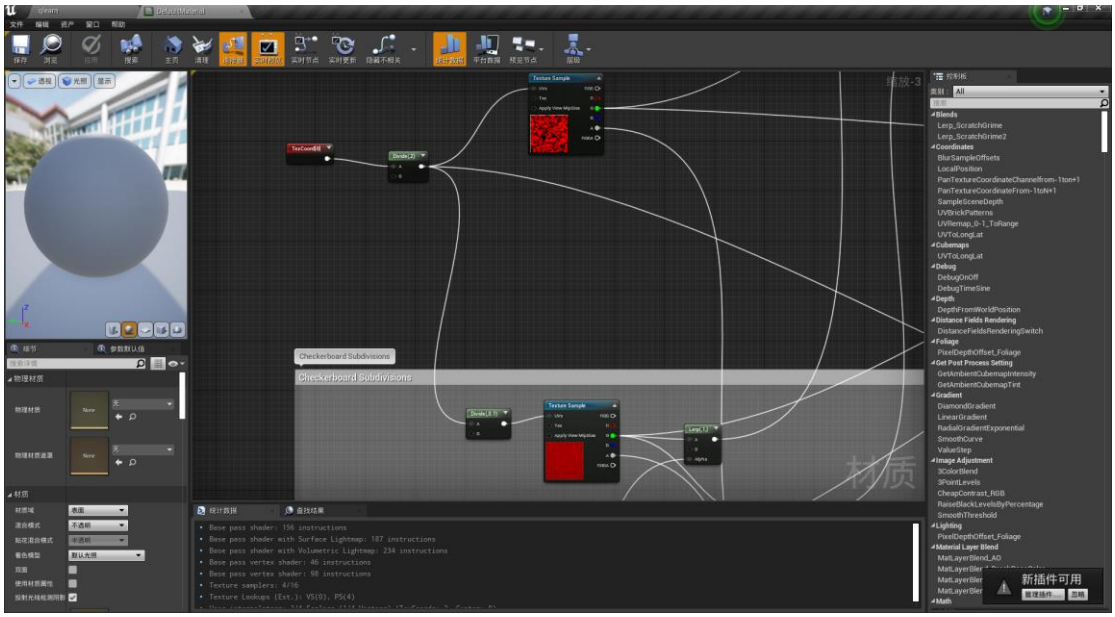

图 9 材质 BluePrint 参数修改

6.2 Airsim API 的使用

Airsim 是 Mircrosoft 开发的一套 UE4 的插件，用以在 UE4 中对无人机进行建模和控制。AirSim 封装了一些 API 接口，使用这些 API 接口，可以用程序跟仿真进行交互。例如可以使用 API 来获取图片、无人机状态、控制无人机/车辆的运动等。

我对于 Airsim API for C++ 的常用函数总结如下：

```
#include <iostream>
// 引入Airsim头文件
#include "vehicles/multirotor/api/MultirotorRpcLibClient.hpp"
using namespace std;
int main()
{
    // 创建一个 client 实例作为无人机的运行客户端
    msr::airlib::MultirotorRpcLibClient client;
    // 开始控制无人机
    client.enableApiControl(true);
    // 解锁无人机
    client.armDisarm(true);
    // 获取名为 "route1_0" 物体在 Airsim 坐标系中的坐标
    auto pos = client.simGetObjectPose("route1_0").position;
    // 起飞
    client.takeoffAsync()->waitOnLastTask();
    // 以1.5m/s的速度移动到坐标为(0, 0, -3)的位置
    client.moveToPositionAsync(0, 0, -3, 1.5)->waitOnLastTask();
    // 以3m/s的速度移动到坐标为(10, 0, -3)的位置
    client.moveToPositionAsync(10, 0, -3, 3)->waitOnLastTask();
    // 降落
    client.landAsync()->waitOnLastTask();
    // 上锁无人机
    client.armDisarm(false);
    // 停止控制无人机
    client.enableApiControl(false);

    return 0;
}
```

图 10 Airsim API 总结

6.3 无人机类的编写

在无人机类中，我模拟了无人机的操作，并通过 Airsim API 在 UE4 引擎中实现了具体场景的展示。

在代码具体编写的过程中，我的主要思想是采取抽象和封装的面向对象思想，将无人机的高度、位置、将要执行的动作等参数设为私有成员，并在公有成员中定义操作它们的函数。这样做的好处是向类的使用者隐藏了代码的具体执行细节，使得程序更加易读，函数的调用更加清晰明了。

同时，我重载了 takeAction 函数，以实现无人机在训练和测试两种情况下不同的动作执行策略。

```
#include "vehicles/multirotor/api/MultirotorRpcLibClient.hpp"
#include"QMatrix.h"
/*
    无人机类，在 Q-learning 算法中模拟了无人机的操作，并通过 Airsim API 在 UE4 引擎中实现了无人机的运行.
*/
class Drone {
    private:
        // 无人机的当前高度
        int currentAltitude;
        // 无人机的当前位置
        int currentState;
        // 无人机当前将要执行的动作
        int action;
        // 无人机的速度
        int speed;
        // 无人机的燃料
        int fuel;
    public:
        Drone() : currentAltitude(0), currentState(0), action(-1), speed(3), fuel(constants::num_steps_in_episode) {}
        // 重载了默认构造函数，以无人机的高度和位置初始化
        Drone(int altitude, int state) : currentAltitude(altitude), currentState(state), action(-1), speed(3), fuel(constants::num_steps_in_episode) {}
        // 用高度，位置，和当前位置设置无人机的状态
        void setStatus(int altitude, int state, int act);
        // 获取无人机当前的位置
        int getState();
        // 获取无人机当前的高度
        int getAltitude();
        // 获取无人机将要执行的动作
        int getAction();
        // 减少无人机的燃料
        void decreaseFuel();
        // 获取无人机当前的燃料剩余量
        int getFuel();
        // 更新无人机的位置
        void updateState();
        // 更新无人机的高度
        void updateAltitude();
        // 训练时，根据 ε-greedy 策略选择无人机在当前环境下采取的动作
        void chooseAction(QMatrix& qMatrix, double epsilon);
        // 验证时，选择无人机在当前环境下采取的动作
        void chooseAction(QMatrix& qMatrix);
        // 通过 Airsim API 在 UE4 引擎中实现无人机的飞行.
        void takeAction(msr::airlib::MultirotorRpcLibClient& client);
};
```

图 11 *Drone.h*

6.4 奖励矩阵和 Q-Table 类的编写

奖励矩阵类和 Q-Table 类是算法的核心，奖励矩阵类用于存储无人机在不同环境下可以得到的奖励或惩罚。Q-Table 类用于存储无人机在不同环境下选择不同动作所带来的收益。

在代码编写的过程中我引入了一个名为 Fastor 的 C++ 张量库，它是一个模板类，可以根据我们的需要制作不同维度的张量类。同时它提供了类似 Python 中 numpy 库支持的高阶张量的切片、索引、广播功能，为高维数组的操作带来了极大方便。

我的思路是首先根据这个模板类制作出一个(3, 49, 6)的张量类，3 个维度分别表示无人机的高度，无人机的位置和无人机的动作。然后由这个类派生出奖励矩阵类和 Q-Table 类，继承了 Fastor 张量库的便捷操作。最后再分别向这两个类添加新的成员函数，以分别完成 Q-learning 算法和香农定理。

```
#include<vector>
#include<math.h>
#include"Fastor/Fastor.h"
#include"constants.h"
#include"Router.h"
#include"Barrier.h"

/*
    奖励矩阵类, 继承自 Fastor 张量库, 存储无人机在不同环境下可以得到的奖励或惩罚,
    在 Q-learning 算法中用来计算核心的数据结构 Q-Table.
*/


class RewardMatrix : public Fastor::Tensor<double, constants::num_altitudes, con-
stants::num_states, constants::num_actions> {
    public:
        RewardMatrix();
        // 以当前的路由器和障碍物集合为参数，初始化奖励矩阵
        void initReward(Router& route, std::vector<Barrier>& vecBarr);
        // 由无人机和路由器的相对位置确定无人机获得的奖励
        double calculateReward(int altitude, int goalState, double xPos, double yPos);
        // 计算路由器对无人机的仰角
        std::pair<double, double> calcualteTheta(int altitude, int goalState, double xPos,
double yPos);
};
```

图 12 *RewardMatrix.h*

```
#include"Fastor/Fastor.h"
#include"constants.h"
#include"RewardMatrix.h"


/*
    Q 矩阵类，继承自 Fastor 张量库，存储无人机在不同环境下选择不同动作所带来的收益，
    是 Q-learning 算法的核心.
*/

class QMatrix : public Fastor::Tensor<double, constants::num_altitudes, con-
stants::num_states, constants::num_actions> {
    public:
        QMatrix();
        // 静态方法，获取无人机在确定环境下选择不同状态的收益最大值
        static double maxQValue(QMatrix& qMatrix, int currentState, int currentAltitude);
        // 核心函数，根据无人机在当前环境下选择的动作来更新 Q-Table
        void updateQMatrix(RewardMatrix& reward, int currentState, int nextState, int cur-
rentAltitude, int nextAltitude, int action);
};
```

图 13 *QMatrix.h*

6.5 训练函数和测试函数的编写

训练和测试的相关函数存储在功能函数文件 *uavQLearning.h* 中 。这些函数定义了使用 Q-learning 算法对无人机进行训练和验证的方法。

我的基本思路是首先将地图中所有检测到的障碍物、路由器，以及每个路由器对应的奖励矩阵和 Q-Table 存入 Vector 数组中。然后初始化一个 Airsim 客户端类 client 实例和一个无人机类 drone 实例。

在训练函数中核心是 3 层循环：第一层循环是遍历每个路由器，对 lenRoute 个路由器分别训练对应的 Q-Table；第二层循环是对于一个确定的路由器，训练 num_episodes 轮，无人机每次都从起点出发；第三层循环是对于每一轮训练，如果无人机燃料没有耗尽，进行下一个动作，并且更新 Q-Table。

在验证函数中核心是 2 层循环：第一层循环时遍历每个路由器，第二层循环是对于一个确定的路由器，无人机从到达上一个路由器的状态出发，在燃料没有耗尽的情况下不断根据 Q-Table 采取动作，并且在到达该路由器时循环结束。

```
#include"Drone.h"

/*
    功能函数文件，定义了使用 Q-learning 算法对无人机进行训练和验证的方法.
*/


// 获取地图中所有障碍物存入 vector 数组中
void getBarrList(std::vector<Barrier>& vecBarr, msr::airlib::MultirotorRpcLibClient& cli-
ent);

// 获取地图中所有路由器存入 vector 数组中
void getRouterList(std::vector<Router>& vecRouter, msr::airlib::MultirotorRpcLibClient&
client);

// 对于每个路由器，建立一个关于它的奖励矩阵，存入 vector 数组中
void getRewardMatrixList(std::vector<RewardMatrix>& vecRewardMatrix, std::vector<Router>&
vecRoute, std::vector<Barrier>& vecBarr);

// 对于每个路由器，建立一个关于它的 Q-Table，存入 vector 数组中
void getQMatrixList(std::vector<QMatrix>& vecQMatrix, int lenRouter);

// 传入地图中的所有障碍物和路由器，以及它们对应的奖励矩阵和 Q-Table，以及无人机，进行训练
void train(std::vector<Barrier>& vecBarr, std::vector<Router>& vecRouter,
    std::vector<RewardMatrix>& vecRewardMatrix, std::vector<QMatrix>& vecQMatrix, Drone&
drone, msr::airlib::MultirotorRpcLibClient& client);

// 无人机根据 Q-Table 选择最佳的路径
int getOptimalRoute(QMatrix& qMatrix, Drone& drone, Router& router, int startState, int
startAltitude, msr::airlib::MultirotorRpcLibClient& client);

// 传入地图中的所有路由器，以及它们对应的 Q-Table，以及无人机，进行验证，结果通过 Airsim
API 在 UE4 引擎中展示
void validate(std::vector<QMatrix>& vecQMatrix, std::vector<Router> vecRouter, Drone& drone,
msr::airlib::MultirotorRpcLibClient& client);
```

图 14 *uavQlearning.h*

6.6 代码风格统一和注释的编写

在实验的最后，我对整个项目的代码进行了重构并且添加了注释。将环境基类 Enviroments、障碍物类 Barrier、路由器类 Router、奖励矩阵类 RewardMatrix、Q-Table 类 QMatrix、无人机类 Drone 的定义和实现分别放入同名的头文件和源文件中。添加了一个名为 *constants.h* 的常量文件用以保存项目中用到的所有常量。

在头文件中的注释是对于代码功能的描述，对于类的整体描述，我采用了 /* */ 注释，对于类的成员定义，我采用了 // 注释。

在源文件中的注释是对一些具体的实现细节进行描述，采用 // 注释。

七、总结

在本次的高级程序设计语言(C++) 的课程设计实验中，我们的实验过程中体现了课程 C++ 语言面向对象的基本特点。我们对课程讲解和上机训练的有关内容进行了实践训练。

我们在本次课程设计实验的代码部分，体现了面向对象语言的基本特点：定义的无人机类是类和对象的体现。定义的两种环境类路由器和建筑障碍物继承自环境基类，是继承与多态的实现。除此之外，我们利用 Fastor 张量库作为基类，派生出 Q-Table 和 RewardMatrix 两种类，提升了代码的可读性和易用性。

我们在本次课程设计实验的算法部分，积极查阅 Q-Learning 相关资料、合作学习，将算法应用到无人机轨迹优化的实际问题中，还考虑了为避免局部最优、引入环境变量后智能航线的再规划等提出的算法改进。

我们本次课程设计实验的成果部分，完成了无人机智能航线的规划，对实验中引入的其他环境变量，如改变路由器位置等操作，均能较好地做出反馈；同时，利用 UE4，AirSim 搭建的仿真环境给人以良好的人机交互视觉感受。

综上，通过小组成员的通力协作、互帮互助，基于 Q-Learning 算法的无人机路径规划问题得以解决；组内成员各尽其职、收获颇丰，并期待有更多程序设计训练的机会和挑战！

参考文献

[1] Practical Reinforcement Learning — 02 Getting started with Q-learning (https://towardsdatascience.com/practical-reinforcement-learning-02-getting-started-with-q-learning-582f63e4acd9)
[2] 通信常识：波特率，数据传输速率和带宽的相互关系 (https://blog.csdn.net/xchbx/article/details/11537951)
[3] Epic Games Unreal Engine 4 Documentation (https://docs.unrealengine.com/4.27/en-US/)
[4] Microsoft Airsim Documentation (https://microsoft.github.io/AirSim/)
[5] Fastor-wiki (https://github.com/romeric/Fastor/wiki/Getting-Started)
[6] Optimizing-UAV-trajectory-for-maximum-data-rate-via-Q-Learning (https://github.com/khinthandarkyaw98/Optimizing-UAV-trajectory-for-maximum-data-rate-via-Q-Learning)
[7] airsim 详细教程(1): win10 配置 airsim 仿真环境(2021.8.12 更新) (https://zhuanlan.zhihu.com/p/267321662)
[8] airsim 系列(五) - 控制四旋翼的飞行(core api) (https://zhuanlan.zhihu.com/p/307956920)
[9] Q learning - epsilon greedy update (https://stackoverflow.com/questions/48583396/q-learning-epsilon-greedy-update)
[10] How to Set Up a Grid for Level and Game Design Metrics (https://www.youtube.com/watch?v=KSXshiA59OA&ab_channel=TylerMcCombs)
[11] 6.9 — Sharing global constants across multiple files (using inline variables) (https://www.learncpp.com/cpp-tutorial/sharing-global-constants-across-multiple-files-using-inline-variables/)

附录

主函数

```
#include"uavQLearning.h"

int main()
{
    std::cout << "Please press Enter to arm the drone"; std::cin.get();
    msr::airlib::MultirotorRpcLibClient client;

    client.enableApiControl(true);
    client.armDisarm(true);

    std::vector<Barrier> vecBarr;
    std::vector<Router> vecRoute;
    std::vector<RewardMatrix> vecRewardMatrix;
    std::vector<QMatrix> vecQMatrix;
    Drone drone;

    std::cout << "Please press Enter to train"; std::cin.get();
    train(vecBarr, vecRoute, vecRewardMatrix, vecQMatrix, drone, client);
    std::cout << "Please press Enter to validate"; std::cin.get();
    validate(vecQMatrix, vecRoute, drone, client);

    client.armDisarm(false);
    client.enableApiControl(false);

}
```

图 15 *main.cpp*