足球机器人基地大作业考核

任务时间：**2021/9/12 中午12:00 ~ 2021/9/25 中午12:00**

选题一、制作自己的物理引擎

任务背景及限制

物理引擎是当今游戏、电影特效、虚拟现实、工业生产、以及很多科研中必不可少的工具，因为我们需要真实地模拟这个世界，从而创造出精致的作品，作出巧妙的决策。

在接下来的一段时间里，我们希望你能够开发一套自己的“物理引擎”！

这套引擎也许并不那么复杂，因为任务的时间只有短短的两星期，你们中的大部分在此之前对离散的计算机对现实世界的模拟也毫无经验。但是在进入任务的引导之前，我们需要强调以下几点要求：

1. 我们并不对你的编程语言作任何限制
2. 我们并不对你完成的具体功能作任何限制
3. 我们并不对你功能的表现形式作任何限制
4. 我们对你所使用的库有严格的限制——**你所使用的库不能含有任何内置现成物理引擎**(你如果使用类似Unity这样的开发库，那么这项有趣的任务还有什么意义呢？)

任务要求

1. 我们希望你能实现一个**简易的图形界面**
2. 我们希望你能尽量逼真地**实现物体之间的各种物理现象**，包括但不限于刚体的碰撞、重力场的实现、力的平衡、物体表面摩擦系数的加入、非刚体的实现等。
3. 你可以首先从2D平面进行尝试。（其实如果你足够强，我们并不限制你实现3D的物理引擎，只是用最最基础的图形库去在几星期内实现3D引擎实在太难）
4. 你可以首先从最基本的几何形状开始实现，圆形、方形、矩形、三角形再至不规则的形状。
5. 你可能在写完圆形之间的碰撞，再写方形、矩形等其他形状的带角度碰撞时会遇到极大的困难。不要灰心，尽力的去查阅资料，推导公式，看看别人是如何实现这些问题的。
6. 鼓励使用**面向对象的技术**进行开发（不过话说，这么大的工程为啥不面向对象）
7. 如果你擅长 **C++** 的话，图形界面的实现可以用 **OpenCV，EGE** 等带图形操作的库来实现；如果你擅长 **Python** 的话，可以用 **pygame** 中的图形操作来实现。以上只是我所知道的库，你们可以任意使用你们顺手的语言和库，但是一定确保不要带有内置的物理引擎。
8. 形成 **完整的任务报告**，其中尽可能详细的阐述你的任务完成情况。
9. **如果哪里有什么感觉没有表述清楚的地方，大可不必纠结，放飞自己的想象去实现，这是一个开放性任务，随意发挥你们的创造力，去开发自己的物理引擎吧！！！**

任务提交方式

请将全部的源代码（注意是源代码，不含编译后的各种文件）及任务报告

打包发送至邮箱 1328657938@qq.com

选题二、Eric和坦克大战

任务背景

2015 年的一天，Eric 同学从房间向窗外看去，忽然发现自己的邻居掉到了水箱里，正在求救，于是Eric同学突发奇想：为什么不写一个坦克大战呢！于是他花了 3 秒钟写出了一个工程，随后将邻居救了起来。Eric 写的工程是一个全平台可运行的坦克大战游戏，**C 语言编写**，游戏有**简单 AI 功能**，坦克可以上下左右移动，发射炮弹，判断是否击中和击毁，但该游戏仍存在一些问题，需要你找到修复并改进，这样 Eric 同学和他的邻居就可以享受这个游戏。

任务要求

首先是一些**基本要求**：你需要运行这个工程，不规定使用平台，可以改写成 Windows 版本（本项目不能直接在Windows 上运行，需要改动代码），或直接在 Linux 运行（Linux 使用虚拟机或双系统都可以），当然如果你和 Eric 同学一样是 Mac 爱好者和使用者，那么运行它将会变得更加简单。同时，请将你的过程全部记录下来，形成你的文档并提交。

具体来说，Eric 同学希望你帮助他完成以下几个 **任务**：

1. 将他的代码改写得更加规范。
2. 读懂这个工程的运行方式。
3. 找出游戏中尽可能多的bug并修复（可能涉及增、删、改等操作），使游戏能够完成坦克大战的基本任务。
4. 读懂并改进游戏中的AI功能（Eric 同学认为 NPC 越灵活越有趣）。
5. 你觉得游戏还有什么可以改进的吗？比如让界面更美观，引入新的游戏机制和规则，加入多人对战模式，通关模式等等，请你思考并实现。
6. 形成**完整的任务报告**，其中尽可能详细的阐述你的任务完成情况。

任务提交方式

请将全部的源代码（注意是源代码，不含编译后的各种文件）及任务报告

打包发送至邮箱 1652772592@qq.com

附录A：关于选题一的小 demo

C++，OpenCV 版本

（这是51p在大一时候写的一小段代码，为了试一试他刚学的C++ OOP，现在看来那时候他的代码还很青涩，以下只是做一个小小的引导，仅供参考，并不保证正确性）

首先定义了Object类，作为之后所有几何物体的父类。定义了速度、加速度、中心点、旋转角度、角速度、角加速度等属性，还有一系列的函数，见名知意。

```
class Object
{
public:
    Object() = default;
    Object(Vec2D c, Vec2D s,  double W, double ang, Mat* i, Scalar p,
std::vector<forcemoment> F) :core(c), speed(s), angelW(W), angel(ang), image(i),
color(p) { forceSum = F; };
    virtual ~Object() = default;
    virtual void update() = 0;
    void clear();
    virtual void show() = 0;
```

```
    virtual void checkCollision() = 0;

    Vec2D getCore() { return core; }
    Vec2D getSpeed() { return speed; }
    Vec2D getAcceleration() { return acceleration; }
    double getAngelW() { return angelW; }
    double getAngel() { return angel; }


    void setCore(Vec2D x) { core = x; }
    void setSpeed(Vec2D x) { speed = x; }
    void setAcceleration(Vec2D x) { acceleration = x; }
    void setAngelW(double x) { angelW = x; }
    void setAngel(double x) { angel = x; }

    Mat* image;
    Scalar color;

    vector<forcemoment> forceSum;

    Vec2D core;
    Vec2D speed=Vec2D(0,0);
    Vec2D acceleration=Vec2D(0,0);


    double angelAcceleration=0;
    double angelW;
    double angel;


};
```

之后定义了他的子类们（具体的几何形状），这是Square类的头文件。定义了长、宽的、质量的属性，还实现了父类中的部分虚函数。

```
class Square :public Object
{
public:
    Square(Vec2D c, Vec2D s,  double W, Mat* i, Scalar p,vector<forcemoment> F,
double ang, int w, int h) :Object(c, s, W, ang, i, p,F), width(w), height(h) {
mass = w * h; }
    void show() override;
    void update() override;

    void checkCollision() override;

    int width;
    int height;
    double mass;
};
```

下面是部分Square中的非核心代码：

```
void Square::update()//进行对象物理状态的更新
{
    acceleration = Vec2D(0,0);
```

```
    angelAcceleration = 0;
    for (auto i = forceSum.begin(); i != forceSum.end(); i++)
    {
        acceleration += i->force / mass;
        angelAcceleration += (你的公式)
    }
    speed += acceleration;
    core += speed;
    angelW += angelAcceleration;
    angel += angelAcceleration;
}

void Square::show()//进行对象在图形界面上的展示
{
    Point points[1][4];
    double cosa = cos(getAngel());
    double sina = sin(getAngel());
    points[0][0] = Point(getCore().getX() + width * cosa / 2 - height * sina /
2, getCore().getY() + width * sina / 2 + height * cosa / 2);
    points[0][1] = Point(getCore().getX() + width * cosa / 2 + height * sina /
2, getCore().getY() + width * sina / 2 - height * cosa / 2);
    points[0][2]= Point(getCore().getX() - width * cosa / 2 + height * sina / 2,
getCore().getY() - width * sina / 2 - height * cosa / 2);
    points[0][3] = Point(getCore().getX() - width * cosa / 2 - height * sina /
2, getCore().getY() - width * sina / 2 + height * cosa / 2);

    const Point* pts[] = { points[0] };
    int npts[] = { 4 };

    fillPoly(*image,pts,npts,1,color);

}
```

主函数中你可以依次对你当前所有的对象进行下述处理来完成更新：

```
#include <iostream>
#include "Object.h"
#include "Square.h"
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>

using namespace cv;
using namespace std;

int WindowWidth = 1800;
int WindowHeight = 1000;
int FPS = 60;
vector<shared_ptr<Object>> objects;
Mat background = Mat(Size(WindowWidth, WindowHeight), CV_8UC3);

int main()
{
    Mat white;
    background.copyTo(white);

    namedWindow("6");
```

```
    imshow("6", background);

    while (int key = waitKey(10))//图形界面主循环
    {
        if (key == 27) break;

        white.copyTo(background);

        for (int i = 0; i < objects.size(); i++)
        {
            objects[i]->checkCollision();
            objects[i]->update();
            objects[i]->show();
        }

        imshow("6", background);
    }
}
```