五子棋项目设计

1.设计目的

收到雀混等3d类棋牌游戏启发，制作一款轻松有趣，用户体验良好的动态3D五子棋游戏

2.总体设计

**主要函数**流程图

最先执行，运行时对数据进行初始化

在Start后运行，且每一帧都会运行，直到游戏结束退出

3.详细设计

**类与函数及方法**功能说明

**Move类：**用于实现棋子的移动及判定胜负，同时包含悔棋，重置棋盘等功能，是程序中最主要的类

**void Start()**

在程序最开始运行，初始化游戏场景数据。

**void Update()**

在游戏的每一帧运行，Unity刷新率一般在30-100fps，使人看到动态效果。

**void** **Resources.Load(str)**

加载Unity工程文件，参数为文件名称，字符串形式。

**void Getcomponent<name>()**

访问物体属性，参数name为属性名称，无需字符串形式。

**bool Input.GetMouseButtonDown(0)**

检测鼠标左键是否按下。

**Ray** **Camera.main.ScreenPointToRay(Input.mousePosition)**

在鼠标在屏幕上的点击处生成一系列摄像机到棋盘的射线

**RaycastHit() Physics.Raycast(ray, out hit)**

获取射线打在棋盘上的坐标，返回一个存放坐标的数组

**void transform.Translate(x,y,z,Space.World)**

系统库函数，实现物体世界坐标的变化

**void Instantiate(GameObject, Vector3, Quaternion.identity)**

在指定三维坐标处实例化，即创建一个游戏物体

**void SetActive(true/false);**

激活或禁用游戏物体，即让其可见或不可见且脚本失效

**void Destroy(GameObject)**

在场景中删除游戏物体

**int result()**

判断胜负，黑棋胜返回1，白棋胜返回-1

**void regret()**

用于悔棋，每运行一次退回到上一步

**void GetKeyDown(KeyCode.R)**

用于获取键盘输入，当输入R时重置棋盘

**Changeview类：**用于实现转变视角

**void Clickme()**

用于禁用/激活副摄像机，实现多机位视角切换

4.调试与测试

调试过程中，要针对不同棋型进行测试调整，并观察结果是否符合，同时要测试悔棋，重置棋盘，切换视角等按钮的功能。由于本程序在Unity软件内部运行，且脚本捆绑在按钮上，测试调试都是可视化的，调整起来比较容易。

5.源程序清单和执行结果(Move.cs中result（）函数的算法可以进一步在很大程度上优化，result（）显得繁冗复杂，是本程序的败笔)

**Move.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Move : MonoBehaviour

{

    // Start is called before the first frame update

    public GameObject chessb,chessw,chessbwin,chesswwin,chessbc,chesswc,chesswturn,chessbturn;

    public Material wc, bc;

    public Renderer rend;

    public AudioSource chessau;

    private bool finish = true;

    private Vector3 chess;

    private Vector3 pos;//鼠标坐标,目的地

    enum turn {black,white};

    int[,] chessState; //二维数组存储棋盘位置上的落子状态

    int i,j,k,l;

    int re;

    turn chessturn;

    bool playstate = true;//是否结束

    void Start(){

        wc = Resources.Load("white")as Material;

        bc = Resources.Load("black")as Material;

        rend = GetComponent<Renderer>();

        chessturn = turn.black;

        chessState =new int[16,16];

    }

    void Update()

    {

        if(Input.GetMouseButtonDown(0)&&playstate){

            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

            RaycastHit hit = new RaycastHit();

            if (Physics.Raycast(ray, out hit)){

                if (hit.collider.name == "ChessBoard"){

                    pos = hit.point;

                    pos.y = 0.2f;

                    pos.x = Convert.ToInt64(pos.x);

                    pos.z = Convert.ToInt64(pos.z);

                    k = i = (int) (pos.x);

                    l = j = (int) (pos.z);

                    chessState[i,j] = chessturn == turn.black ? 1 : -1;

                    finish = false;

                }

            }

        }

        if(!finish&&playstate){

            float vilocity = 10 \* Time.deltaTime;

            if (transform.position.y <=2){

                transform.Translate(0,vilocity,0,Space.World);

            }

            if (transform.position.y >2){

                Vector3 direction=pos;

                direction.y=transform.position.y;

                Vector3 offset = direction - transform.position;

                transform.position += offset.normalized \* vilocity;

            }

            if(Vector3.Distance(pos, transform.position) <4){

                transform.position = pos;

                chess = transform.position;

                if (chessturn == turn.black){

                    GetComponent<Renderer>().material = bc;

                    GameObject b = Instantiate(chessb, pos, Quaternion.identity);

                    b.transform.parent = chessbc.transform;

                    chessau.Play();

                    chess.x=-3;

                    chess.z=15;

                    transform.position = chess;

                    GetComponent<Renderer>().material = wc;

                    chessbturn.SetActive(false);

                    chesswturn.SetActive(true);

                }

                if (chessturn == turn.white){

                    GameObject w = Instantiate(chessw, pos, Quaternion.identity);

                    w.transform.parent = chesswc.transform;

                    chessau.Play();

                    chess.x=-5;

                    chess.z=15;

                    transform.position = chess;

                    GetComponent<Renderer>().material = bc;

                    chesswturn.SetActive(false);

                    chessbturn.SetActive(true);

                }

                finish = true;

                chessturn = chessturn == turn.black ? turn.white : turn.black;

            }

        }

        re = result();

        if (re == 1)

        {

            chessbwin.SetActive(true);

            playstate = false;

        }

        else if(re == -1)

        {

            chesswwin.SetActive(true);

            playstate = false;

        }

        else if(re == 0)

        {

            chessbwin.SetActive(false);

            chesswwin.SetActive(false);

        }

        if (Input.GetKeyDown(KeyCode.R))

        {

            for (int i = 0; i < 16; i++)

            {

                for (int j = 0; j < 16; j++)

                {

                    chessState[i, j] = 0;

                }

            }

            for (int i=0; i < chessbc.transform.childCount; i++){

                Destroy (chessbc.transform.GetChild(i).gameObject);

            }

            for (int i=0; i < chesswc.transform.childCount; i++){

                Destroy (chesswc.transform.GetChild(i).gameObject);

            }

            chessbwin.SetActive(false);

            chesswwin.SetActive(false);

            chess.x=-5;

            chess.z=15;

            transform.position = chess;

            GetComponent<Renderer>().material = bc;

            playstate = true;

            chessturn = turn.black;

        }

        int result(){

            int flag = 0;

            //如果当前该白棋落子，标定黑棋刚刚下完一步，此时应该判断黑棋是否获胜

            if(chessturn == turn.white)

            {

                for (int i = 0; i < 12; i++)

                {

                    for (int j = 0; j < 16; j++)

                    {

                        if (j < 4)

                        {

                            //横向

                            if (chessState[i, j] == 1 && chessState[i, j + 1] == 1 && chessState[i, j + 2] == 1 && chessState[i, j + 3] == 1 && chessState[i, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == 1 && chessState[i + 1, j] == 1 && chessState[i + 2, j] == 1 && chessState[i + 3, j] == 1 && chessState[i + 4, j] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //右斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j + 1] == 1 && chessState[i + 2, j + 2] == 1 && chessState[i + 3, j + 3] == 1 && chessState[i + 4, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j - 1] == 1 && chessState[i + 2, j - 2] == 1 && chessState[i + 3, j - 3] == 1 && chessState[i + 4, j - 4] == 1)

                            {

                                flag = 1;

                               return flag;

                            }

                        }

                        else if (j >= 4 && j < 12)

                        {

                            //横向

                            if (chessState[i, j] == 1 && chessState[i, j + 1] == 1 && chessState[i, j + 2] == 1 && chessState[i, j + 3] == 1 && chessState[i, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == 1 && chessState[i + 1, j] == 1 && chessState[i + 2, j] == 1 && chessState[i + 3, j] == 1 && chessState[i + 4, j] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //右斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j + 1] == 1 && chessState[i + 2, j + 2] == 1 && chessState[i + 3, j + 3] == 1 && chessState[i + 4, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j - 1] == 1 && chessState[i + 2, j - 2] == 1 && chessState[i + 3, j - 3] == 1 && chessState[i + 4, j - 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                        }

                        else

                        {

                            //横向

                            if (chessState[i, j] == 1 && chessState[i, j + 1] == 1 && chessState[i, j + 2] == 1 && chessState[i, j + 3] == 1 && chessState[i, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == 1 && chessState[i + 1, j] == 1 && chessState[i + 2, j] == 1 && chessState[i + 3, j] == 1 && chessState[i + 4, j] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //右斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j + 1] == 1 && chessState[i + 2, j + 2] == 1 && chessState[i + 3, j + 3] == 1 && chessState[i + 4, j + 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == 1 && chessState[i + 1, j - 1] == 1 && chessState[i + 2, j - 2] == 1 && chessState[i + 3, j - 3] == 1 && chessState[i + 4, j - 4] == 1)

                            {

                                flag = 1;

                                return flag;

                            }

                        }

                    }

                }

                for (int i = 12; i < 16; i++)

                {

                    for (int j = 0; j < 12; j++)

                    {

                        //只需要判断横向

                        if (chessState[i, j] == 1 && chessState[i, j + 1] == 1 && chessState[i, j + 2] == 1 && chessState[i, j + 3] == 1 && chessState[i, j + 4] == 1)

                        {

                            flag = 1;

                            return flag;

                        }

                    }

                }

            }

            //如果当前该黑棋落子，标定白棋刚刚下完一步，此时应该判断白棋是否获胜

            else if(chessturn == turn.black)

            {

                for (int i = 0; i < 12; i++)

                {

                    for (int j = 0; j < 16; j++)

                    {

                        if (j < 4)

                        {

                            //横向

                            if (chessState[i, j] == -1 && chessState[i, j + 1] == -1 && chessState[i, j + 2] == -1 && chessState[i, j + 3] == -1 && chessState[i, j + 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == -1 && chessState[i + 1, j] == -1 && chessState[i + 2, j] == -1 && chessState[i + 3, j] == -1 && chessState[i + 4, j] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //右斜线

                            if (chessState[i, j] == -1 && chessState[i + 1, j + 1] == -1 && chessState[i + 2, j + 2] == -1 && chessState[i + 3, j + 3] == -1 && chessState[i + 4, j + 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == -1 && chessState[i + 1, j - 1] == -1 && chessState[i + 2, j - 2] == -1 && chessState[i + 3, j - 3] == -1 && chessState[i + 4, j - 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                        }

                        else if (j >= 4 && j < 12)

                        {

                            //横向

                            if (chessState[i, j] == -1 && chessState[i, j + 1] == -1 && chessState[i, j + 2] == -1 && chessState[i, j + 3] == -1 && chessState[i, j + 4] ==- 1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == -1 && chessState[i + 1, j] == -1 && chessState[i + 2, j] == -1 && chessState[i + 3, j] == -1 && chessState[i + 4, j] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //右斜线

                            if (chessState[i, j] == -1 && chessState[i + 1, j + 1] == -1 && chessState[i + 2, j + 2] == -1 && chessState[i + 3, j + 3] == -1 && chessState[i + 4, j + 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == -1 && chessState[i + 1, j - 1] == -1 && chessState[i + 2, j - 2] == -1 && chessState[i + 3, j - 3] == -1 && chessState[i + 4, j - 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                        }

                        else

                        {

                            //横向

                            if (chessState[i, j] == -1 && chessState[i, j + 1] ==- 1 && chessState[i, j + 2] == -1 && chessState[i, j + 3] == -1 && chessState[i, j + 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //纵向

                            if (chessState[i, j] == -1 && chessState[i + 1, j] ==- 1 && chessState[i + 2, j] ==- 1 && chessState[i + 3, j] ==- 1 && chessState[i + 4, j] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                            //左斜线

                            if (chessState[i, j] == -1 && chessState[i + 1, j - 1] == -1 && chessState[i + 2, j - 2] == -1 && chessState[i + 3, j - 3] == -1 && chessState[i + 4, j - 4] == -1)

                            {

                                flag = -1;

                                return flag;

                            }

                        }

                    }

                }

                for (int i = 12; i < 16; i++)

                {

                    for (int j = 0; j < 12; j++)

                    {

                        //只需要判断横向

                        if (chessState[i, j] == -1 && chessState[i, j + 1] == -1 && chessState[i, j + 2] == -1 && chessState[i, j + 3] == -1 && chessState[i, j + 4] == -1)

                        {

                            flag = -1;

                            return flag;

                        }

                    }

                }

            }

            return flag;

        }

    }

    public void regret(){

        if (chessturn==turn.black)

        {

            chessbwin.SetActive(false);

            chesswwin.SetActive(false);

            Destroy (chesswc.transform.GetChild(chesswc.transform.childCount-1).gameObject);

            chess.x=-3;

            chess.z=15;

            transform.position = chess;

            GetComponent<Renderer>().material = wc;

            chessturn = turn.white;

            chessbturn.SetActive(false);

            chesswturn.SetActive(true);

            chessState[k,l] = 0;

        }

        else

        {

            chessbwin.SetActive(false);

            chesswwin.SetActive(false);

            Destroy (chessbc.transform.GetChild(chessbc.transform.childCount-1).gameObject);

            chess.x=-5;

            chess.z=15;

            transform.position = chess;

            GetComponent<Renderer>().material = bc;

            chessturn = turn.black;

            chessbturn.SetActive(true);

            chesswturn.SetActive(false);

            chessState[k,l] = 0;

        }

    }

}

**Changeview.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

using TMPro;

public class Changeview : MonoBehaviour

{

    public Camera cam;

    public GameObject chessmove;

    public TMP\_Text t;

    public GameObject chessbwin,chesswwin;

    public void Clickme()

    {

        if (cam.GetComponent<Camera>().enabled == false)

        {

            cam.GetComponent<Camera>().enabled = true;

            chessmove.GetComponent<Move>().enabled = false;

            chessbwin.SetActive(false);

            chesswwin.SetActive(false);

            t.text = "下棋模式";

        }

        else

        {

            cam.GetComponent<Camera>().enabled = false;

            chessmove.GetComponent<Move>().enabled = true;

            t.text = "鉴赏模式";

        }

    }

}