**中文摘要**

象棋起源于中国，最早可追溯到战国时期。随着时代的变迁，象棋的个数和种类以及玩法都有所变化，以至于演变成现在我们熟知的中国象棋。象棋的演变和创新可以让更多人参与到益智类竞技当中。

决战象棋是一款基于中国象棋行走规则的带有战斗力比较的游戏，当棋子攻击敌方棋子时会比较双方属性，胜者留，败者亡。结合战斗力比较以改变传统的中国象棋着法思维模式，让游戏带有不一样的趣味和挑战性，让双方玩家在游戏互动中得到更好的思维锻炼。

本系统利用Unity 3D游戏引擎开发，分为5个模块：计时模块、复盘模块、战斗模块、资源重用模块和界面。

关键词：战斗，复盘，资源重用， Unity 3D

**ABSTRACT**

Chess originate in China, it dates back to the warring states period. With the changing times, the number and type of chess and the playing method make a difference, which evolves Chinese Chess weknow.The evolution and innovation of chess can help more people participate in the competitive sports.

Battle chess, based on the rules of ChineseChess, is a game with comparison of combat power. It will compare the attributes of both sides when the chess attacks the enemy chess, winner survived, loser gone.Combiningcomparison of combat power to change traditional Chinese chess thinking mode, which makes the game fun and challenging, and makes both player get better mentai exercise in the game.

This system uses Unity 3D engine development, includeingfive modules: battle,time, chessboard reuming, resource reuse, and interface.

Key words: Battle,chessboardresuming,resource reuse,Unity3D

# 1.诸论

**1.1研究背景与意义**

近几十年来，随着计算机硬件和软件技术的不断发展，人们的娱乐方式有了巨大的变化，电子游戏就是其中之一。电子游戏在整个计算机产业的带动下不断地创新和发展。自从计算机发明，向各个领域发展，到成为我们现在每天工作和生活必不可少的一部分的这个过程中，电子游戏也逐步渗入到我们每个人的娱乐活动当中。而计算机已经普及的今天，对于可以用计算机进行程序编辑的人来说，开发属于自己的游戏，已经不再是梦想。

益智类游戏相较于其他类型的游戏更有益身心健康，现如今，象棋游戏比比皆是，但由不同玩法和规则建立的象棋游戏并不多，如果象棋游戏有新的创新和玩法，那么将会给益智类游戏贡献出新的力量，将会让玩家改变新的思维模式来面对这个游戏。

**1.2决战象棋设计研究方法**

本系统使用C#语言和Unity 3D游戏引擎开发。一方面，C#的事件处理机制对游戏开发非常友好。另一方面，Unity 3D不仅支持C#语言开发，而且提供的API和强大的UGUI可视化编辑系统使游戏开发更加方便，因此，整体游戏功能会得到更好的完善。

**1.3本文的主要工作**

本文的主要工作是将战斗属性结合到象棋中，建立一个全新的象棋计算机系统，研究工作主要集中在以下几个方面:

1.战斗属性和计算

结合战斗属性，赋予棋子新的生命力，通过战斗计算决出胜负。

2.行走规则

各兵种行走规则各异，都需要相应的算法实现。

3.计时

计时跟回合切换息息相关，而本系统的回合切换因加入战斗属性而变得更为复杂，因此计时也会作出相应的改变。

4.资源重用

对于游戏来说，资源的优化既减轻硬件负担，也有利于提高游戏体验的流畅度。

5.悔棋和复盘

悔棋和复盘都需要记录棋谱，而且也因加入战斗属性而变得不一样。

**1.4本文的结构**

第一章阐述了选题背景，课题的研究方式，课题的主要工作和文章的章节安排。

第二章是系统的分析和总体设计。

第三章是给出了实验环境和程序实现。

第四章是全文的总结及展望。

# 2.系统的分析和设计

**2.1需求分析**

**2.1.3功能需求**

一. 开始界面

开始游戏按钮：进入加载界面。

二. 加载界面

进度条追踪第三个界面加载进度。

进度条满后，自动进入游戏主界面

三.游戏主界面

(游戏进行时界面)

1. 局时：本局总剩余用时
2. 步时：当前这一步限定用时
3. 棋子：点击己方棋子时，在棋盘上会显示出可走点，根据行走规则，第二次点击有效可走点时会进行走步。第二次点击原棋子会取消选择。第二次点击其他己方棋子会更换选择。
4. 棋子属性：点击己方棋子时，右侧会显示本棋子具体属性，包括生命，攻击，防御，战力
5. 悔棋按钮
6. 求和按钮
7. 认输按钮
8. 返回按钮
9. 加生命属性按钮
10. 加攻击属性按钮
11. 加防御属性按钮

(复盘时界面)

1. 返回按钮
2. 开局按钮：使棋盘步数置为0的状态
3. 终局按钮：使棋盘步数置为最终步数的状态
4. 某棋损失生命显示文本：吃子时，胜方显示丢失血量
5. 某棋加属性显示文本：显示每回合双方加属性情况
6. 上一步按钮：使棋盘进入上一步数的状态
7. 下一步按钮：使棋盘进入下一步数的状态
8. 总步数显示文本

**2.1.1非功能性需求**

**2.1.1.1游戏规则**

1. 每人每回合操作有两步骤，分别是走步与加属性。
2. 各棋子走法遵循普通中国象棋的各棋子行走规则。
3. 每人走完步后，除自己的将帅之外，可以给其他棋子加一定属性。
4. 当完成走步和加属性操作后，计时才会停止，对方才开始计时。
5. 吃子行为属于战斗行为，胜败之分参考战斗计算方式。
6. 将帅无视战斗计算方式，可以无损吃子。
7. 棋局胜负判定遵循普通中国象棋规则。

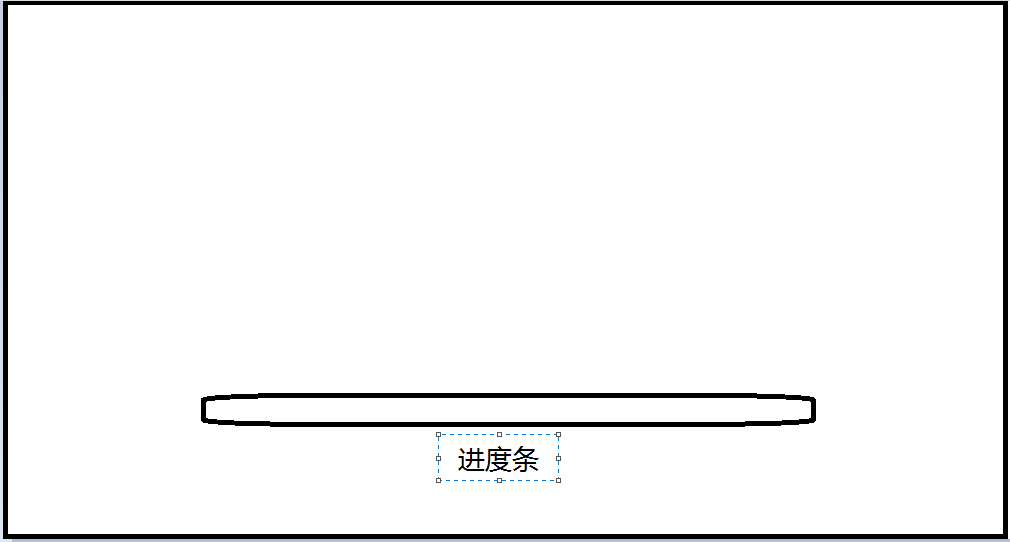
**2.1.1.2界面要求**

横屏，分3个界面，开始界面，加载界面，游戏主界面。

1. 开始界面



1. 加载界面

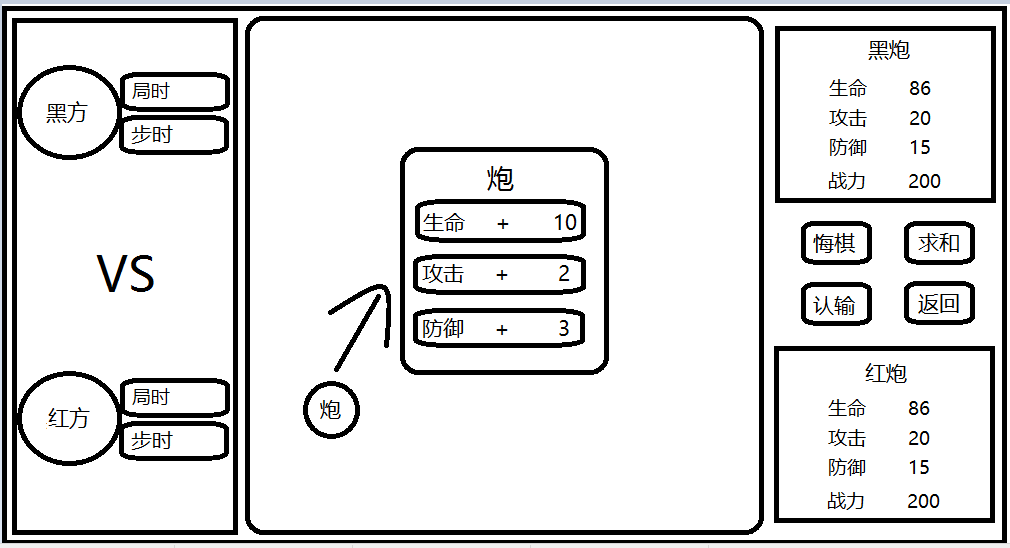


1. 游戏主界面

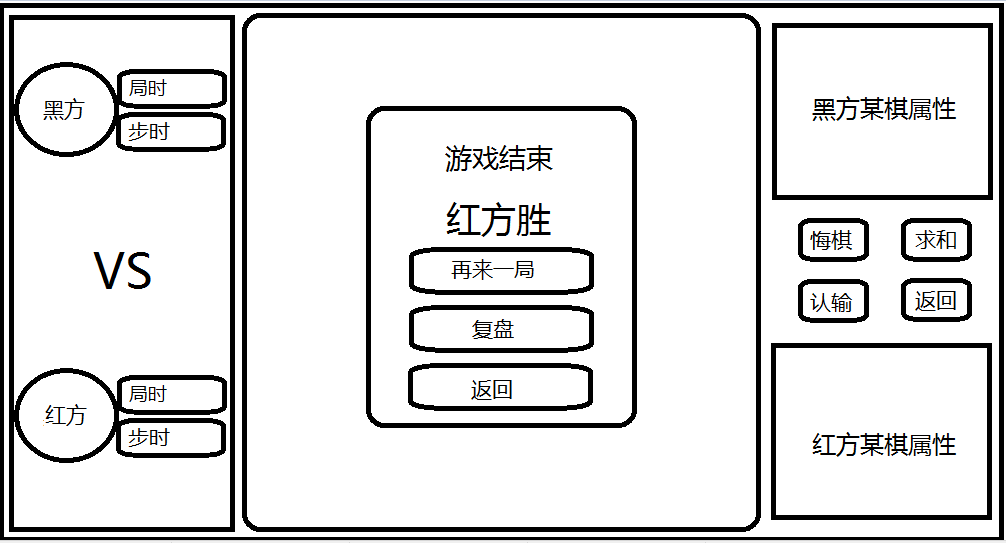
游戏进行时的界面1走棋



游戏进行时的界面2 选择加属性



游戏结束时的界面



复盘时的界面



**2.1.1.3状态要求**

1. 整体
2. 游戏未开始
3. 游戏进行中
4. 游戏暂停
5. 游戏复盘
6. 游戏结束
7. 游戏进行时
8. 轮到红方走棋
9. 轮到黑方走棋
10. 轮到红方加属性
11. 轮到黑方加属性
12. 棋子交互状态
13. 未被选择
14. 被选中
15. 正在移动
16. 棋子所处棋局状态
17. 安然无恙
18. 攻击状态（将军）
19. 被攻击（被将军）
20. 被称为目标，敌人可吃
21. 无路可走
22. 阵亡

**2.2开发所需条件和相关技术**

操作系统为Window10系统，游戏引擎为Unity3D，开发环境为Visual Studio 2017，开发语言为C#和Json。测试环境为Window10 PC端和Android移动端。

**2.3总体设计**

本系统设计遵循MVC架构。MVC架构分为三层，分别是：视图层（View）、控制层（Control）和数据层（Model）。

视图层（View）：这层为展现给玩家的逻辑层，包括UI和场景，View中的任何逻辑不对Model层的数据进行修改，它对Model层拥有只读权限。

控制层（Control）：对Model层拥有读写权限。一般功能包括：（1）判断玩家操作是否符合条件，若符合执行指令。（2）根据逻辑对Model层中的数据进行修改。（3）回调View层中的方法通知操作完成或者失败。

数据层（Model）：主要是存放各模块的相关数据同时提供修改数据的方法。

根据本系统的开发需求，作出了以下整体的脚本规划：

1.视图层有3个UI场景，需要3个对应的UI脚本。另外，场景的异步加载状态信息监听需要1个脚本。

2.控制层包括对整体游戏的控制，物体的生成，资源的回收和时间的限制，需要4个对应的控制管理类脚本。

3.数据层负责存取数据的脚本和提供修改数据的方法，需要2个脚本。

4.动态和需要获取信息的游戏物体主要有棋子和棋盘可走点。棋子可以抽象出共有的属性和行为，需要1个父类脚本。接着，共有车、马、炮、士、象、兵、帅七个兵种，都拥有不同行走规则，需要7个子类脚本。棋盘可走点物体需要1个脚本来控制状态。

5.对游戏各模块的状态定义和事件声明，需要1个脚本。

6.对棋子的属性，要有一个属性载体和属性配置信息相关处理，需要2个脚本。

7.各棋子初始属性配置表，这个为txt文件即可，语法格式为Json。

**2.4数据结构**

数据结构是一个程序的骨架，选择一种好的数据结构可以使程序的运行效率大大提高。本系统需要存储的数据主要是棋子，棋子属性和棋盘点。这三种数据类型又有紧密的联系，具体表现为：给定棋子，可以访问到棋子的属性，也可以访问到它所处的位置；给定棋盘点，则可以得知点上是否有棋子，是什么棋子。因此，如果使用数组来存储以上的数据类型就很难取得关联信息了，而C#有一种叫字典（Dictionary）的键值对数据结构，能很好的处理棋子、棋子属性、棋盘点的关系。棋子和棋盘点建立双向键值对关系，方便互相访问。棋子和棋子属性则是单向键值对关系。

**2.5资源重用**

游戏性能的优劣取决于资源的消耗量，为减轻CPU运算和内存占用的负担，本系统资源重用主要应用在三个方面：

1.棋子阵亡时，回收到池而不是直接销毁物体对象。

2.再来一局时，从池里面提取所有棋子而不是重新生成棋子。

3.复盘和悔棋时，充池里面提取相应棋子而不是重新生成棋子。

**2.5战斗属性处理**

每颗棋子都有3种属性，分别为生命，攻击，防御。战斗力是对这3个属性的总体评价。

战斗力计算方式：

Combat为战斗力，Hp为生命，Attack 为攻击，Defence为防御。

Combat = Hp\*0.4 + Defence\*10 + Attack\*10

A,B双方决斗，生死计算如下：

假设：

a\_Hp = 100, a\_Attack = 4, a\_Defence = 2

b\_Hp = 120, b\_Attack = 6, b\_Defence = 1

按最小回合数为准，生命最先低于0的死亡，胜出方扣除最小回合数\*(对方伤害-自 身防御)的生命值

a对b造成的伤害a\_demage = a\_Attack-b\_Defence，若相减是负数，那么a\_demage=1。

b对a造成的伤害b\_demage =b\_Attack-a\_Defence，若相减是负数，那么b\_demage=1。

a攻击：b\_Hp/a\_demage = 40 （回合)

b攻击：a\_Hp/b\_demage = 25 （回合)

因此，b最先把a打死。

若a为进攻方，b为防守方，那么b用了25回合，a用了25回合，a阵亡。

minTimes = 25

b剩余血量：hp = b\_Hp - (a\_Attack-b\_Defence) \* minTimes = 120 - 75 = 45

若b为进攻方，a为防守方，那么b用了25回合，a用了24回合，a阵亡。

minTimes = 24

b剩余血量：hp = b\_Hp - (a\_Attack-b\_Defence) \* minTimes = 120 - 72 = 48

# 3.系统实现

接下来对每个模块都会详细介绍，按照脚本底层顺序和优先完善顺序，即先有此脚本，下一个脚本才好处理的原则分别介绍。

**3.1实验环境**

Window10 PC端和Android移动端。

**3.2状态和事件定义**

为了方便管理，游戏中各个模块用到的状态和事件定义都放在同一个文件里。

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

/// 游戏整体状态

/// </summary>

public enum GameStatus

{

NotBegin, //未开始

Going, //进行中

Pause, //暂停

Replay, //复盘

End, //结束

}

/// <summary>

/// 游戏时的各个状态

/// </summary>

public enum Playing

{

None,

OnRed, //到红方走

RedAdding, //到红方加属性

OnBlack, //到黑方走

BlackAdding, //到黑方加属性

}

/// <summary>

/// 棋子交互状态

/// </summary>

public enum ChessReciprocalState

{

unChoosed, //未被选择状态

beChoosed, //被选择中

moving, //移动中

}

/// <summary>

/// 棋子所处的情境状态

/// </summary>

public enum ChessSituationState

{

Idle, //安然无恙

Attacking, //将军状态

BeAttacked, //只有帅/将才拥有的被将军状态

BeTaget, //被成为目标状态，敌人可吃

NoWayOut, //无路可走状态

Death, //阵亡

}

public delegate void PushEventHandler(GameObject chess);

public delegate void TakeEventHandler(GameObject chess);

public delegate void RestoreEventHandler(GameObject chess);

/// <summary>

/// //选择本棋子事件，通知其他棋子为取消选择状态

/// </summary>

public delegate void ChooseEventHandler(GameObject chess);

/// <summary>

/// 提示可以吃子

/// </summary>

/// <param name="point">传入可吃子的位置信息</param>

public delegate void TipsKillEventHandler(Vector2 point);

/// <summary>

/// 设置攻击方事件，

/// </summary>

/// <param name="chess">传自身棋子即进攻方</param>

public delegate void SetAttackerEventHandler(GameObject chess);

/// <summary>

/// 设置防守方事件，

/// </summary>

/// <param name="chess">传自身棋子即被进攻方</param>

public delegate void SetDefenderEventHandler(GameObject chess);

/// <summary>

/// 移动，选中状态的棋子才会动，其他会过滤掉

/// </summary>

/// <param name="point">传入目标位置</param>

public delegate void MoveEventHandler(Vector2 point);

/// <summary>

/// 移动完成

/// </summary>

public delegate void MoveCompleteHandler();

/// <summary>

/// 阵亡者事件

/// </summary>

/// <param name="chess">阵亡者物体</param>

public delegate void KilledEventHandler(GameObject chess);

/// <summary>

/// 加属性操作完成

/// </summary>

public delegate void AddAttrCompleteEventHandler();

/// <summary>

/// 游戏结束

/// </summary>

public delegate void GameOverEventHandlerWithParam(string winer);

public delegate void GameOverEventHandler();

/// <summary>

/// 更新棋局信息完毕

/// </summary>

public delegate void UpdateGameDataCompleteEventHandler();

/// <summary>

/// 悔棋事件

/// </summary>

public delegate void UndoEventHandler();

/// <summary>

/// 复盘模式事件

/// </summary>

public delegate void ReplayModeEventHandler();

public class GameDefine{

}

**3.3资源重用实现**

棋子在一开始被生成时即加入到工作区，棋子阵亡时会被回收到回收区，当棋子再次出现在棋盘上时，会从回收期区提取出棋子并加入到工作区。工作区和回收区都使用GameObject类型的列表来表示。当棋子在这两个区发生变动时，有提供通知各模块的事件声明。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PoolManager : MonoBehaviour

{

public static event PushEventHandler PushEvent; //刚加入工作区事件(棋子刚被创建就订阅一堆事件)

public static event TakeEventHandler TakeEvent; //从回收区提取棋子事件(需要将被吃时取消订阅的事件全订阅回来)

public static event RestoreEventHandler RestoreEvent; //某个棋子回收事件(需要取消订阅一堆事件)

public static List<GameObject> work\_List; //工作区列表

public static List<GameObject> restore\_List; //回收区列表

public static Transform restoreChesses; //回收区父物体

public static Transform workChesses; //工作区父物体

void Awake()

{

restoreChesses = GameObject.Find("RestoreChesses").transform;

workChesses = GameObject.Find("WorkChesses").transform;

work\_List = new List<GameObject>();

restore\_List = new List<GameObject>();

}

/// <summary>

/// 将创建的棋子加入工作区

/// </summary>

/// <param name="chess"></param>

public static void Push(GameObject chess)

{

work\_List.Add(chess);

SetParentToWork(chess);

PushEvent(chess);

}

/// <summary>

/// 提取棋子

/// </summary>

/// <param name="chess"></param>

public static void Take(GameObject chess)

{

work\_List.Add(chess);

if (restore\_List.Contains(chess))

restore\_List.Remove(chess);

SetParentToWork(chess);

chess.SetActive(true);

TakeEvent(chess);//把取消订阅的事件全订阅回来

}

/// <summary>

/// 回收棋子

/// </summary>

/// <param name="chess"></param>

public static void Restore(GameObject chess)

{

restore\_List.Add(chess);

if (work\_List.Contains(chess))

work\_List.Remove(chess);

SetParentToRestore(chess);

chess.SetActive(false);

RestoreEvent(chess);

}

public static void SetParentToWork(GameObject chess)

{

chess.transform.SetParent(workChesses, true);

}

public static void SetParentToRestore(GameObject chess)

{

chess.transform.SetParent(restoreChesses, true);

}

public void ClearPool()

{

if (work\_List != null)

work\_List.Clear();

if (restore\_List != null)

restore\_List.Clear();

}

public void OnDestroy()

{

ClearPool();

}

}

**3.4数据层实现**

游戏中需要存取的数据分别为：场景中棋盘可走点物体与二维坐标的映射关系；当前棋局棋子与二维坐标的映射关系；当前棋局二维坐标与棋子的映射关系；棋局图谱即棋子与二维坐标的映射字典列表；棋子属性图谱即棋子与其属性的映射字典列表。棋子与其阵亡时对应步数的映射关系。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class GameCache

{

/// <summary>

/// 场景cell或棋子与平面直角坐标(x方向0-8，y方向0-9)之间的映射关系

/// </summary>

private static Dictionary<Vector3, Vector2> coords;

/// <summary>

/// 棋子与他现在二维坐标的映射

/// </summary>

private static Dictionary<GameObject, Vector2> chess2Vector;

/// <summary>

/// 棋子二维坐标与自身的映射

/// </summary>

private static Dictionary<Vector2, GameObject> vector2Chess;

/// <summary>

/// 记录每回合的所有棋局图谱信息

/// </summary>

public static List<Dictionary<GameObject, Vector2>> maps;

/// <summary>

/// 记录每回合的所有棋子对应的属性信息

/// </summary>

public static List<Dictionary<GameObject, string>> attrMaps;

/// <summary>

/// 记录阵亡者与对应步数的映射

/// </summary>

private static Dictionary<GameObject, int> loserStepDic;

public static void SetCoords(GameObject[,] cells)

{

if (coords == null)

{

coords = new Dictionary<Vector3, Vector2>();

for (int x = 0; x <= 8; x++)

{

for (int y = 0; y <= 9; y++)

{

int a = (int)cells[x, y].transform.position.x; //强制为整形，消除微弱差别的影响

int b = (int)cells[x, y].transform.position.y;

int c = (int)cells[x, y].transform.position.z;

coords.Add(new Vector3(a, b, c), new Vector2(x, y));

}

}

}

}

public static Dictionary<Vector3, Vector2> Coords

{

get { return coords; }

}

public static void SetChessVectorDic(List<GameObject> chessList)

{

chess2Vector = new Dictionary<GameObject, Vector2>();

vector2Chess = new Dictionary<Vector2, GameObject>();

for (int i = 0; i < chessList.Count; i++)

{

Vector3 pos = chessList[i].transform.position;

Vector3 v3 = new Vector3((int)pos.x, (int)pos.y, (int)pos.z);//强制为整形，消除微弱差别的影响

chess2Vector.Add(chessList[i], coords[v3]);

vector2Chess.Add(coords[v3], chessList[i]);

}

}

public static void SetLoserStepDic(GameObject loser, int step)

{

if (loserStepDic == null)

loserStepDic = new Dictionary<GameObject, int>();

loserStepDic.Add(loser, step);

}

public static Dictionary<GameObject, Vector2> Chess2Vector

{

get { return chess2Vector; }

}

public static Dictionary<Vector2, GameObject> Vector2Chess

{

get { return vector2Chess; }

}

public static Dictionary<GameObject, int> LoserStepDic

{

get { return loserStepDic; }

}

public static void ClearChessVectorDic()

{

if(Chess2Vector != null || Vector2Chess != null)

{

chess2Vector.Clear();

vector2Chess.Clear();

}

}

/// <summary>

/// 更新棋局信息

/// </summary>

/// <returns></returns>

public static void UpdateChessData()

{

ClearChessVectorDic();

SetChessVectorDic(PoolManager.work\_List);

}

/// <summary>

/// 添加棋谱

/// </summary>

public static void SetMaps()

{

if (maps == null)

maps = new List<Dictionary<GameObject, Vector2>>();

Dictionary<GameObject, Vector2> temp = new Dictionary<GameObject, Vector2>();

foreach (KeyValuePair<GameObject, Vector2> kvp in Chess2Vector)

{

//一定要这样遍历赋值的，否则如果直接temp=CalculateUtil.chess2Vector的话

//就相当于引用了这个静态字典，每次添加到棋谱里就是同一个temp数据

temp.Add(kvp.Key, kvp.Value);

}

maps.Add(temp); //添加棋谱

}

/// <summary>

/// 添加每回合全部棋子对应属性信息图谱

/// </summary>

public static void SetAttrMaps()

{

if (attrMaps == null)

attrMaps = new List<Dictionary<GameObject, string>>();

Dictionary<GameObject, string> dic = new Dictionary<GameObject, string>();

foreach (GameObject chess in PoolManager.work\_List)

{

AttrBox chessAttrs = GameUtil.GetChessAttrList(chess);

string attrStr = chessAttrs.Hp + "\_" + chessAttrs.Attack + "\_" + chessAttrs.Defence;

dic.Add(chess, attrStr);

}

attrMaps.Add(dic);

}

public static void ClearCache()

{

if (maps != null)

maps.Clear();

if (attrMaps != null)

attrMaps.Clear();

if (loserStepDic != null)

loserStepDic.Clear();

ClearChessVectorDic();

}

}

**3.5界面实现**

场景1UI界面摆放如图：



场景1界面比较简单，只有一个开始游戏按钮，点击就进入场景2。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Scene1\_UI : MonoBehaviour

{

public void onBeginClick()

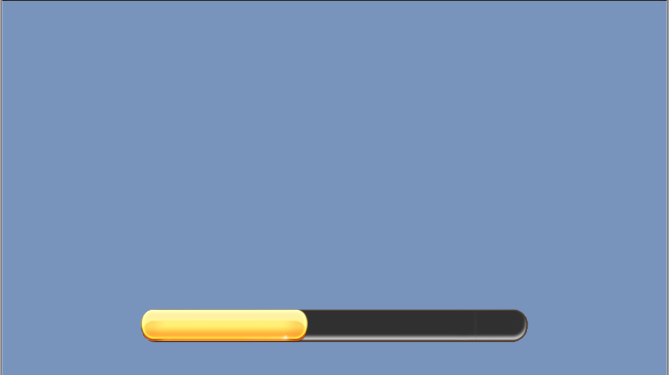
{

SceneManager.LoadScene("scene2");

}

}

场景2UI界面摆放如图：



场景2是加载界面，由场景1跳转进入，等待场景3物体加载完毕后自动跳转到场景3界面。这个过程需要异步加载。进度条显示场景3的加载进度。

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class LoadingByAsync : MonoBehaviour

{

AsyncOperation async; //异步对象

public static LoadingByAsync instance;

public Slider loadingSlider;

void Awake()

{

if (instance == null)

instance = this;

loadingSlider = GameObject.Find("LoadingSlider").GetComponent<Slider>();

}

void Start()

{

StartCoroutine(DelayLoading());

}

void Update()

{

if (async == null)

return;

int toProcess;

//async.progress 的取值范围在0.1 - 1之间， 但是它不会等于1，加载完也就等于0.9

if (async.progress < 0.9f)

{

toProcess = (int)async.progress \* 100;

}

else

{

toProcess = 100;

}

if (loadingSlider.value < toProcess)

{

loadingSlider.value = toProcess;

}

if (toProcess == 100)

{

async.allowSceneActivation = true;

}

}

public IEnumerator DelayLoading()

{

yield return new WaitForSeconds(0.1f);

yield return Load\_Scene("scene3(Main)");

}

//注意这里返回值一定是 IEnumerator

public IEnumerator Load\_Scene(string sceneName)

{

//异步读取场景。

async = SceneManager.LoadSceneAsync(sceneName);

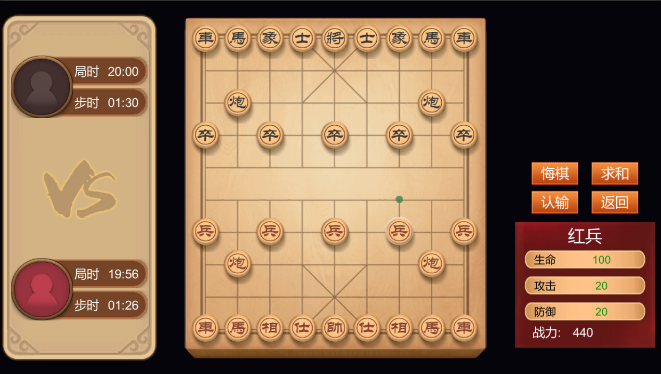
async.allowSceneActivation = false;

yield return async;

}

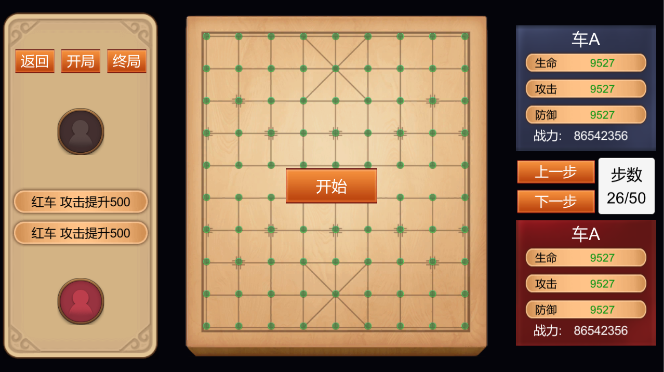
}

场景3游戏进行时界面如图：

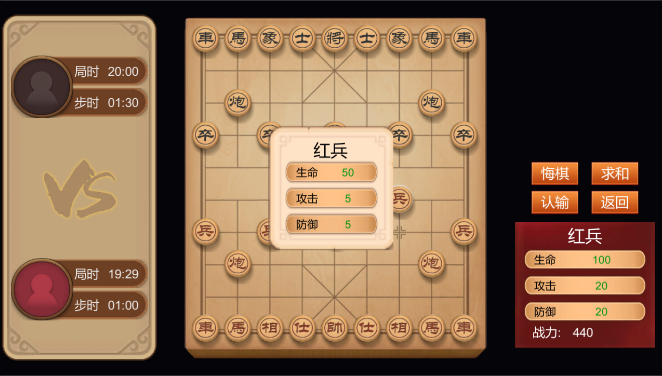


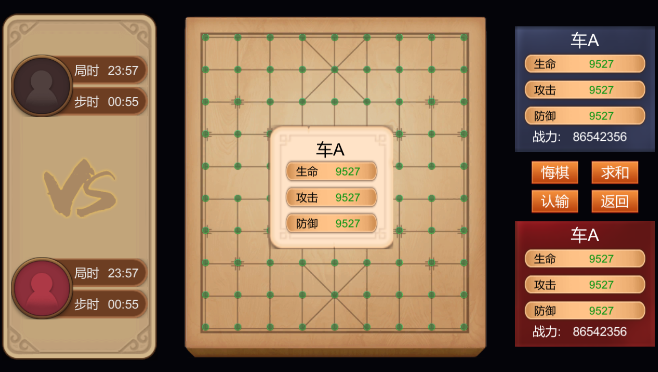


场景3复盘时界面如图：



场景3游戏进行时加属性如图:





场景3的UI元素比较多，需要分情况处理。具体如下：

按游戏模式划分：

1.游戏进行时，左侧显示双方局时和步时，右侧中部显示四个供玩家操作的按钮，分别是悔棋、求和、认输和返回按钮。右侧相应位置显示当前行走方选中棋子的属性面板。

2.游戏复盘时，左侧显示双方当前步数棋子属性发生修改的情况。左侧上方显示三个供玩家操作的按钮，分别是返回、开局和终局按钮。右侧中部显示步数情况和两个供玩家操作的按钮，分别是上一步和下一步按钮。

按游戏进展状态划分：

1.当一方走步完成后，再次点击棋子，中间显示出加属性面板，其中有三个供玩家操作的按钮，分别是加生命，加攻击和加防御按钮。

2.当游戏结束时，中间显示出游戏结束面板，其中有三个供玩家操作的按钮，分别是再来一局，复盘和返回按钮。游戏结束时关闭其余面板并且禁止其他操作。

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

using UnityEngine.UI;

public class Scene3\_UI : MonoBehaviour

{

private static Scene3\_UI instance = null;

public static Scene3\_UI Instance { get { return instance; } }

#region 左

private GameObject leftGameMode;

private GameObject leftReplayMode;

private Text blackAllTime;

private Text blackStepTime;

private Text redAllTime;

private Text redStepTime;

private Text lostHp;

private Text addAttr;

#endregion

#region 中

private Transform GridsTrans;

public static GameObject[,] cells;

private GameObject beginBtn;

private GameObject addAtrrPanel; //加属性面板

private Text addChessName;

private Text addHpValue;

private Text addAttackValue;

private Text addDefenceValue;

private GameObject endPanel; //结束面板

private Text winer;

#endregion

#region 右

private GameObject rightGameMode;

private GameObject rightReplayMode;

private GameObject blackDetailPanel;

private Text b\_Name;

private Text b\_Hp;

private Text b\_Attack;

private Text b\_Defence;

private Text b\_Combat;

private GameObject redDetailPanel;

private Text r\_Name;

private Text r\_Hp;

private Text r\_Attack;

private Text r\_Defence;

private Text r\_Combat;

private Text stepsLabel;

#endregion

private GameObject curAddChess;

private AttrBox curAttr;

public static event AddAttrCompleteEventHandler AddAttrCompleteEvent;

public static event UndoEventHandler UndoEvent;

public static event ReplayModeEventHandler ReplayModeEvent;

public static event GameOverEventHandler GiveUpEvent;

public static event GameOverEventHandler SuePeaceEvent;

void Awake()

{

if (instance == null)

instance = this;

leftGameMode = GameObject.Find("Canvas/Left/GameMode");

leftReplayMode = GameObject.Find("Canvas/Left/ReplayMode");

blackAllTime = GameObject.Find("Canvas/Left/GameMode/Black/AllTime/Value").GetComponent<Text>();

blackStepTime = GameObject.Find("Canvas/Left/GameMode/Black/StepTime/Value").GetComponent<Text>();

redAllTime = GameObject.Find("Canvas/Left/GameMode/Red/AllTime/Value").GetComponent<Text>();

redStepTime = GameObject.Find("Canvas/Left/GameMode/Red/StepTime/Value").GetComponent<Text>();

lostHp = GameObject.Find("Canvas/Left/ReplayMode/LostLife/Label").GetComponent<Text>();

addAttr = GameObject.Find("Canvas/Left/ReplayMode/AddAttr/Label").GetComponent<Text>();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*中\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

GridsTrans = GameObject.Find("Grids").transform;

cells = new GameObject[9, 10];

for (int y = 0; y <= 9; y++)

{

Transform row = GridsTrans.FindChild("row" + y);

for (int x = 0; x <= 8; x++)

{

cells[x, y] = row.FindChild("cell" + x).gameObject;

cells[x, y].GetComponent<Image>().enabled = false;

}

}

GameCache.SetCoords(cells); //将场景找到的cell作为参数，处理写入映射缓存

beginBtn = GameObject.Find("Canvas/Middle/BeginBtn");

addAtrrPanel = GameObject.Find("Canvas/Middle/AddAttrPanel");

addChessName = GameObject.Find("Canvas/Middle/AddAttrPanel/ChessName").GetComponent<Text>();

addHpValue = GameObject.Find("Canvas/Middle/AddAttrPanel/Grid/Hp/Value").GetComponent<Text>();

addAttackValue = GameObject.Find("Canvas/Middle/AddAttrPanel/Grid/Attack/Value").GetComponent<Text>();

addDefenceValue = GameObject.Find("Canvas/Middle/AddAttrPanel/Grid/Defence/Value").GetComponent<Text>();

endPanel = GameObject.Find("Canvas/Middle/EndPanel");

winer = GameObject.Find("Canvas/Middle/EndPanel/ResultLabel").GetComponent<Text>();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*右\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

rightGameMode = GameObject.Find("Canvas/Right/GameMode");

rightReplayMode = GameObject.Find("Canvas/Right/ReplayMode");

blackDetailPanel = GameObject.Find("Canvas/Right/BlackAttrDetail");

b\_Name = blackDetailPanel.transform.FindChild("ChessName").GetComponent<Text>();

b\_Hp = blackDetailPanel.transform.FindChild("Grid/Hp/Value").GetComponent<Text>();

b\_Attack = blackDetailPanel.transform.FindChild("Grid/Attack/Value").GetComponent<Text>();

b\_Defence = blackDetailPanel.transform.FindChild("Grid/Defence/Value").GetComponent<Text>();

b\_Combat = blackDetailPanel.transform.FindChild("Combat/Value").GetComponent<Text>();

redDetailPanel = GameObject.Find("Canvas/Right/RedAttrDetail");

r\_Name = redDetailPanel.transform.FindChild("ChessName").GetComponent<Text>();

r\_Hp = redDetailPanel.transform.FindChild("Grid/Hp/Value").GetComponent<Text>();

r\_Attack = redDetailPanel.transform.FindChild("Grid/Attack/Value").GetComponent<Text>();

r\_Defence = redDetailPanel.transform.FindChild("Grid/Defence/Value").GetComponent<Text>();

r\_Combat = redDetailPanel.transform.FindChild("Combat/Value").GetComponent<Text>();

stepsLabel = rightReplayMode.transform.FindChild("Step/StepsLabel").GetComponent<Text>();

AddAttrCompleteEvent += HideAddAttrPanel;

AddAttrCompleteEvent += HideAttrPanel;

ReplayModeEvent += ResetChessBoardPoints;

ReplayModeEvent += UpdateReplayModeText;

UndoEvent += ResetChessBoardPoints;

UndoEvent += UpdateAttrPanel;

TimeManager.TimeUpEventWithParam += ShowEndPanel;

TimeManager.TimeUpEvent += ResetChessBoardPoints;

TimeManager.TimeUpEvent += HideAddAttrPanel;

Chess\_Boss.BossKilledEventWithParam += ShowEndPanel;

Chess\_Boss.BossKilledEvent += ResetChessBoardPoints;

Chess\_Boss.BossKilledEvent += HideAddAttrPanel;

}

public List<GameObject> chessList;

public CreateManager createManager;

void Start()

{

blackAllTime.text = "00:00";

blackStepTime.text = "00:00";

redAllTime.text = "00:00";

redStepTime.text = "00:00";

lostHp.text = "";

addAttr.text = "";

addAtrrPanel.SetActive(false);

addHpValue.text = "50";

addAttackValue.text = "5";

addDefenceValue.text = "5";

blackDetailPanel.SetActive(false);

redDetailPanel.SetActive(false);

SetMode(true);

}

void Update()

{

if (GameController.gameStatus == GameStatus.Going)

{

blackAllTime.text = GameUtil.TimeToStr(TimeManager.GetAllTime("Black"));

redAllTime.text = GameUtil.TimeToStr(TimeManager.GetAllTime("Red"));

blackStepTime.text = GameUtil.TimeToStr(TimeManager.GetStepTime("Black"));

redStepTime.text = GameUtil.TimeToStr(TimeManager.GetStepTime("Red"));

}

}

/// <summary>

/// 设置模式，游戏模式还是复盘模式

/// </summary>

public void SetMode(bool isGameMode)

{

leftGameMode.SetActive(isGameMode);

rightGameMode.SetActive(isGameMode);

leftReplayMode.SetActive(!isGameMode);

rightReplayMode.SetActive(!isGameMode);

}

/// <summary>

/// 开始按钮点击事件

/// </summary>

public void OnBeginClick()

{

TimeManager.ResetAllTime();

GameController.BeginGame();

beginBtn.SetActive(false);

}

/// <summary>

/// 重置棋盘所有点状态

/// 1.开始移动时

/// 2.取消自身选中时

/// 3.切换选中时

/// 4.悔棋时

/// </summary>

public static void ResetChessBoardPoints()

{

foreach (GameObject cell in cells)

{

cell.GetComponent<Image>().enabled = false;

}

}

/// <summary>

/// 监听棋子被选中事件

/// </summary>

/// <param name="chess"></param>

public void OnChoose(GameObject chess)

{

curAddChess = chess;

curAttr = GameUtil.GetChessAttrList(chess);

UpdateAttrPanel(chess);

bool isAdding = GameController.playing == Playing.RedAdding || GameController.playing == Playing.BlackAdding;

addAtrrPanel.SetActive(isAdding);

if (isAdding)

{

addChessName.text = GameUtil.GetChineseChessName(chess);

}

else

addChessName.text = "";

}

public void UpdateAttrPanel()

{

if (curAddChess != null && curAttr != null)

{

SetAttrTexts(curAttr, curAddChess);

}

}

public void UpdateAttrPanel(GameObject chess)

{

AttrBox attrBox = GameUtil.GetChessAttrList(chess);

SetAttrTexts(attrBox, chess);

}

public void UpdateReplayModeText()

{

string[] arr = GameUtil.CompareStepAttr(GameController.step);

lostHp.text = arr[0];

addAttr.text = arr[1];

stepsLabel.text = GameController.step + "/" + (GameCache.maps.Count - 1);

}

public void OnHpClick()

{

curAttr.Hp += 50;

AddAttrCompleteEvent();

}

public void OnAttackClick()

{

curAttr.Attack += 5;

AddAttrCompleteEvent();

}

public void OnDefenceClick()

{

curAttr.Defence += 5;

AddAttrCompleteEvent();

}

public void HideAddAttrPanel()

{

addAtrrPanel.SetActive(false);

}

public void HideAttrPanel()

{

redDetailPanel.SetActive(false);

blackDetailPanel.SetActive(false);

}

/// <summary>

/// 设置属性面板

/// </summary>

/// <param name="attrBox">属性</param>

/// <param name="tag">棋子阵营</param>

public void SetAttrTexts(AttrBox attrBox, GameObject chess)

{

if (attrBox.Hp == 0)

{

redDetailPanel.SetActive(false);

blackDetailPanel.SetActive(false);

return;

}

redDetailPanel.SetActive(chess.tag == "Red");

blackDetailPanel.SetActive(chess.tag == "Black");

if (chess.tag == "Red")

{

r\_Name.text = GameUtil.GetChineseChessName(curAddChess);

r\_Hp.text = attrBox.Hp.ToString();

r\_Attack.text = attrBox.Attack.ToString();

r\_Defence.text = attrBox.Defence.ToString();

r\_Combat.text = attrBox.Combat.ToString();

}

else

{

b\_Name.text = GameUtil.GetChineseChessName(curAddChess);

b\_Hp.text = attrBox.Hp.ToString();

b\_Attack.text = attrBox.Attack.ToString();

b\_Defence.text = attrBox.Defence.ToString();

b\_Combat.text = attrBox.Combat.ToString();

}

}

public void ShowEndPanel(string loserStr)

{

endPanel.SetActive(true);

if (loserStr == "Red")

{

winer.text = "黑方胜";

}

else if (loserStr == "Black")

{

winer.text = "红方胜";

}

else

{

winer.text = "和棋";

}

}

public void HideEndPanel()

{

endPanel.SetActive(false);

}

/// <summary>

/// 再来一局

/// </summary>

public void OnPlayAgainClick()

{

HideEndPanel();

GameController.ResetGame();

OnBeginClick();

}

/// <summary>

/// 复盘模式

/// </summary>

public void OnReplayModeClick()

{

SetMode(false);

HideEndPanel();

GameController.ReplayModeGame();

ReplayModeEvent();

}

/// <summary>

/// 点击返回事件

/// </summary>

public void OnBackClick()

{

SceneManager.LoadScene("scene1");

}

/// <summary>

/// 悔棋点击事件

/// </summary>

public void OnUndoClick()

{

if (GameCache.maps.Count >= 3)

{

//回合状态回退

if (GameController.playing == Playing.OnRed || GameController.playing == Playing.RedAdding)

GameController.playing = Playing.OnRed;

else

GameController.playing = Playing.OnBlack;

//棋谱回退 并移除回退前两步信息

GameUtil.SetChessBoardByMaps(GameCache.maps.Count - 1 - 2);

GameCache.maps.RemoveRange(GameCache.maps.Count - 2, 2);

GameCache.attrMaps.RemoveRange(GameCache.attrMaps.Count - 2, 2);

GameCache.UpdateChessData(); //更新映射关系，但不计入图谱

UndoEvent();

}

else

{

//GameObject.Find("UndoButton").GetComponent<Button>().enabled = false;

}

}

/// <summary>

/// 认输

/// </summary>

public void OnGiveUpClick()

{

GiveUpEvent();

if (GameController.playing == Playing.OnRed || GameController.playing == Playing.RedAdding)

ShowEndPanel("Red");

else

ShowEndPanel("Black");

ResetChessBoardPoints();

HideAddAttrPanel();

}

/// <summary>

/// 求和

/// </summary>

public void OnSuePeaceClick()

{

SuePeaceEvent();

ShowEndPanel("Peace");

ResetChessBoardPoints();

HideAddAttrPanel();

}

/// <summary>

/// 上一步

/// </summary>

public void OnPreStepClick()

{

if (GameController.gameStatus == GameStatus.Replay)

{

GameController.step--;

if (GameController.step < 0)

{

GameController.step = 0;

return;

}

GameUtil.SetChessBoardByMaps(GameController.step);

UpdateReplayModeText();

}

}

/// <summary>

/// 下一步

/// </summary>

public void OnNextStepClick()

{

if (GameController.gameStatus == GameStatus.Replay)

{

GameController.step++;

if (GameController.step > GameCache.maps.Count - 1)

{

GameController.step = GameCache.maps.Count - 1;

return;

}

GameUtil.SetChessBoardByMaps(GameController.step);

UpdateReplayModeText();

}

}

/// <summary>

/// 开局

/// </summary>

public void OnBeginStepClick()

{

if (GameController.gameStatus == GameStatus.Replay)

{

GameController.step = 0;

GameUtil.SetChessBoardByMaps(0);

UpdateReplayModeText();

}

}

/// <summary>

/// 终局

/// </summary>

public void OnEndStepClick()

{

if (GameController.gameStatus == GameStatus.Replay)

{

GameController.step = GameCache.maps.Count - 1;

GameUtil.SetChessBoardByMaps(GameCache.maps.Count - 1);

UpdateReplayModeText();

}

}

public void OnDestroy()

{

ReplayModeEvent -= ResetChessBoardPoints;

ReplayModeEvent -= UpdateReplayModeText;

UndoEvent -= UpdateAttrPanel;

UndoEvent -= ResetChessBoardPoints;

AddAttrCompleteEvent -= HideAddAttrPanel;

AddAttrCompleteEvent -= HideAttrPanel;

TimeManager.TimeUpEventWithParam -= ShowEndPanel;

TimeManager.TimeUpEvent -= ResetChessBoardPoints;

TimeManager.TimeUpEvent -= HideAddAttrPanel;

Chess\_Boss.BossKilledEventWithParam -= ShowEndPanel;

Chess\_Boss.BossKilledEvent -= ResetChessBoardPoints;

Chess\_Boss.BossKilledEvent -= HideAddAttrPanel;

}

# 参考文献

[1].unity3d配置Android环境，打包发布Apk流程详解.

https://www.jianshu.com/p/3c67fbfbb67c.2016.4.2

[2].[整理]unity3d优化总结篇.

http://www.cnblogs.com/harlan1009/p/3970584.html.2014.9.14

[3]. unity5.3新API使用进度条,异步加载场景的代码.

https://www.jianshu.com/p/bf4ec64cb454.2014.9.14

[4]. C#中委托、事件和回调函数的理解.

https://blog.csdn.net/hit\_why/article/details/54604908.2017.1.18

[5].计算机专业毕业论文(象棋游戏).

https://wenku.baidu.com/view/782c1d1910a6f524ccbf85c2.html.2010.12.29

[6].中国象棋游戏设计与实现毕业设计毕业论文.

https://wenku.baidu.com/view/037039115b8102d276a20029bd64783e09127d69.html?rec\_flag=default&mark\_pay\_doc=1&mark\_rec\_page=1&mark\_rec\_position=5&mark\_rec=view\_r\_1&clear\_uda\_param=1.2017.10.9

[7].Unity - Scripting API

https://docs.unity3d.com/560/Documentation/ScriptReference/index.html.2017.7.12

[8].Unity基础 -- Git版本的源代码管理工具使用.

https://www.jianshu.com/p/fa06a5004d06.2016.12.15

[9].iTween for Unity by Bob Berkebile.

http://www.pixelplacement.com/itween/documentation.php.2017.1.18

[10].百度百科.

https://baike.baidu.com/item/%E4%B8%AD%E5%9B%BD%E8%B1%A1%E6%A3%8B/278314?fr=aladdin.2017.1.18