**杭州电子科技大学**

**毕业设计（论文）外文文献翻译**

|  |  |
| --- | --- |
| **毕业设计（论文）题目** | **基于Godot引擎的冒险游戏设计与实现** |
| **翻译（1）题目** | **使用Unity引擎开发MOBA游戏** |
| **翻译（2）题目** | **星际争霸中的混合寻路** |
| **学 院** | **计算机学院** |
| **专 业** | **计算机科学与技术** |
| **姓 名** | **金嘉浩** |
| **班 级** | **14052313** |
| **学 号** | **14051616** |
| **指导教师** | **陆剑锋** |

译文一：

使用Unity引擎开发MOBA游戏[[1]](#footnote-1)

摘要：MOBA（多人在线竞技游戏）是目前最火爆的在线视屏游戏之一。这篇论文将讨论如何使用Unity5游戏引擎来实现一个Windows平台上的典型的MOBA游戏原型。论文会着重讲述利用Unity的内部组件来完成MOBA游戏，以及使用Unity的C#脚本API和网络引擎、3D模型、粒子系统等为Unity设计的或者是Unity资源商店中的第三方组件来完成附加行为。论文将展示一些有用的程序设计模式和Unity中使用的设计模式的综述。同时，论文会讨论指定网络引擎——Photon Unity Networking中可用的多重游戏状态同步机制，以及它们在多个客户端之间同步多种不同种类的游戏信息的具体方法。最重实现的游戏中包含了大部分现代MOBA游戏的主要特色，包括：不同玩法的英雄、技能、队伍间对战、资源收集和消费、多重不同的地图、防御设施等。最后，论文以对将Unity 5作为MOBA游戏开发和执行的环境的评论作为总结。

1. 介绍

MOBA是基于对抗的多人在线游戏。两队玩家以完成一个或者更多的游戏目标为最终目的，在一个封闭的游戏环境中（一般是一张地图）进行对抗。让MOBA游戏在其他游戏类型，比如FPS（第一人称射击游戏）中脱颖而出的是其从它原型RTS游戏（实施战略雷游戏）中所借鉴的高度策略化的游戏模式。这包括了资源收集和等轴的游戏视角。大多数情况下，MOBA游戏的首要目标是摧毁对方队伍的防御设施，不过随着更多的MOBA游戏被开发，很多其他的首要目标也随之出现。

目前，MOBA是最热门的在线游戏类型之一。 最热门的MOBA游戏：英雄联盟的发行公司宣布了英雄联盟在2016年有着超过百万的月活跃玩家。

这篇论文讨论使用Unity引擎实现一个MOBA游戏的原型。游戏引擎是游戏开发的软件和工具，它提供了游戏开发中常用组件的完整实现，比如物理引擎，渲染引擎，声音引擎，网络支持等。同时它们还允许游戏开发者在之前的功能模块上或者在之前的模块之中开发针对自己游戏特定的功能模块。

选择Unity引擎来开发是因为它免费，并且它有丰富的在线学习资源。同时Unity也是当前三个最热门的商业游戏引擎之一，其余两个引擎是Unreal（虚幻）引擎和CryEngine。Unity是丹麦的Unity科技开发的。

实现的游戏名称叫做Moghi Battles，它保留了绝大部分的MOBA游戏特色，第三部分中会提及。

1. 相关工作

除了一些在线视频教程和英雄联盟首席设设计师的演示录像以外，没有太多关于MOBA游戏开发的资料。一般的在线游戏研究包括了讨论在线游戏架构和算法的研究，到在线电子游戏的实现教程。比如Prabha的研究，讨论了Unity实现的在线游戏架构和登录方法。其研究也讨论了这篇论文中所使用到的Photon Unity网络引擎的验证方法的实现。Bharambe，Pang，Seshan和Colyseus的讨论了一个不同的分布式的在线游戏框架以及在该框架下同步游戏状态的方法。其他讨论在线游戏分布式架构的研究有Assiotis和Tzanov的研究，他们聚焦于有一个庞大世界的MMORPG（大型多人在线游戏）。Hsu，Ling，Li和Kuo将传统的客户端——服务端服务器架构与端对端的服务器架构、景象服务器架构进行比较，得出延时敏感网络游戏最适合的服务器架构是客户端——服务端服务器架构，同时它们还期待一个集群服务器架构来增加可扩展性。他们的研究也讨论了游戏状态同步的两个主要方法：保守方法和乐观方法。前者在多客户端中试图完全避免游戏状态冲突，后者则允许少量冲突。

关于Unity引擎自身的论文有许多。有一些论文讨论了使用Unity作为开发非游戏应用程序的工具。比如Craighead、 Burke和Murphy的论文，讨论了使用Unity实现一个机器人模拟器。Unity官方的网页也提供了非常多的教程来帮助新的Unity开发者入门和理解Unity的基本概念。最后，视频网站上还有很多热门的视频教程，比如YouTube上的教程会教你如何制作特定类型的游戏。用户quill18creates上传的视频系列就是这样一个非常有用的视频教程，它演示了如何用Unity引擎和Photon Unity网络框架制作一个FPS游戏。

我们还不知道是否有一个使用Unity引擎开发的MOBA游戏取得商业成功。

1. 基本MOBA概念

因为市面上的MOBA游戏题材非常多并且仍然在增加，所以精确定义MOBA游戏的样式非常难。不过大多数MOBA游戏都有的基本概念如下：英雄、不同英雄拥有不同技能、队伍、资源收集和消费、拥有防御设施的基地、小兵和道路。

**英雄**是玩家控制的单位。每个玩家一般只控制一个单位，那就是他们选择的英雄。这里没有RTS游戏中军队管理的概念。MOBA一般允许玩家在多个不同玩法、技能和背景的英雄中选择一个。英雄一般以生命值作为死亡衡量的标准，生命值耗尽时，英雄死亡。大多数MOBA  
游戏中，英雄可以在复活点复活。

在MOBA中，**队伍**一般是由二到六个玩家组成的。一般有两个队伍进行对抗。两个队伍以达到首要目标为最重目的进行对抗，首要目标可能是摧毁对方的防御设施或者是得到最高的击杀数等。率先完成首要目标的队伍就是胜利的队伍。

**技能**一般是玩家可以与自己英雄进行交互的英雄特有的操作，是与敌人英雄进行交互和增强自己的唯一交互手段。在大多数MOBA中，技能可以用键盘上的快捷键进行释放。释放技能一般会消耗一些非生命值资源。同时，技能在释放过后有冷却时间，一般在技能释放后开始计时，在冷却时间中技能不能被再次释放。不同英雄的技能有着很大的不同，这些技能定义了这个英雄的玩法。

生命值以外的**资源**可以用来进行特定的游戏操作，比如释放一个技能，复活英雄或者是购买其他东西。资源一般来说会随着时间基类，也可以通过击败地方英雄或者攻破对方的防御设施获得。

**基地**是玩家比赛开始的地方，也是防御设施所在的地方。每个队伍有他们自己的基地。基地中的防御设施也拥有生命值，它们中的一些拥有攻击地方英雄的能力。MOBA游戏的主要目标大部分是进攻到地方基地，摧毁主要防御设施。

**路**是连接两方基地的道路，MOBA游戏可以有不同路数，不过一般都是三条：上路下路和中路。中路一般是在上路下路之间，是最短的一条路。

**小兵**是电脑AI控制的有生命值的单位。小兵一般出生在每一队的基地，向对面基地进攻。小兵可以对对方英雄和小兵造成伤害。

这篇论文中讨论的MOBA原型包含了以上的所有概念（除了小兵的AI）。

1. 架构实现

在网络游戏中客户端——服务端架构的基本组成部分包括：游戏客户端、会话服务器、用户数据库、游戏服务器和游戏数据库。

**游戏客户端**是运行在玩家电脑上的应用程序，玩家可以通过它获取游戏的状态。游戏状态可能会由游戏客户端模拟生成或者在其他地方模拟生成。

**会话服务器**（也叫大厅服务器）负责开启客户端和游戏服务器的会话。这个操作一般只在玩家登陆游戏客户端或者启动一个特殊应用程序时发送一个有效的验证凭证时，与储存在用户数据库中的验证凭证进行比较。

**游戏服务器**是模拟游戏和帮助客户端进行数据交换的服务器。商业MOBA游戏一般有多个游戏服务器，将比赛分配到不同服务器上以提高可扩展性。如果有永久存储数据的需要，游戏服务器可以有访问游戏数据库的权限（比如MMORPG中存储永久性的世界数据）。

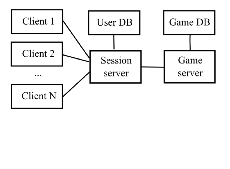


Figure 传统的客户端-服务器架构游戏服务器

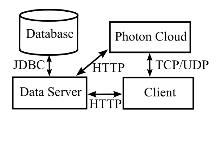


Figure 2 服务器架构实现

我们实现的游戏服务器架构与上面所说的客户端——服务器架构有略微不同。我们不会去实现游戏服务器和会话服务器，而是用Photon Cloud服务来代替。Photon Cloud服务在与PUN（Photon Unity网络）框架一起运行的时候可以提供会话服务器和游戏服务器的功能。我们会在第六部分讨论PUN的其他细节。此外，由于我们只实现一个简单的圆形，所以只会有一个存储用户数据和比赛数据的数据库。我们用Spring实现的JDBC（Java数据库连接）API来访问这个的数据库。Spring是一个很火的Java网络开发框架。有一部分英雄联盟的服务器栈也是用Spring实现的。数据服务器提供了新用户注册的HTTP端点，获取验证代号，返回结果。客户端和Photon Cloud服务都访问数据服务器。因为Photon Cloud服务是通用的、游戏不可知的，它不会自己去做验证工作，而是要把验证的工作委托给PUN接口。数据服务器用来验证的HTTP端点实现了这个接口，Photon Cloud可以使用它用做验证服务。当验证成功后，数据服务器将验证代号传给Photon Cloud，Photon Cloud再传给客户端。客户端与Photon Cloud使用TCP或者UDP协议进行通讯。第六部分将详细讨论在不同情况下使用不同的协议。因为数据服务器的实现比较简单，所以论文将着重讨论客户端的实现。

1. Unity基本概念

这个部分会讲解Unity中最重要的概念和特点。因为我们实现的游戏是一个3D游戏，所以这一部分会着重讲述Unity中3D开发的特点。Unity编辑器是基于Unity引擎的用来制作游戏的一个桌面应用程序。编辑器的用户界面由以下构成：在屏幕中间的场景视图，在屏幕底部的文件浏览器，在屏幕左侧的层级视图，在屏幕右边的游戏对象浏览器或是资源观察器，在屏幕顶部的菜单栏。

1. 项目

所有Unity里的开发文件都存在Unity项目中。当你创建一个新的项目，你的指定用户目录下也会同时创建一个与你项目名称相同的文件夹。新建的项目会包含Assets（资源）文件夹，其中包含了所有游戏中用药的数据，比如3D模型，声音，脚本等。资源文件夹一开始是空的，运行时所需要用到的资源都要放在资源文件夹下的Resources文件夹。

1. 场景和游戏实体

场景是一个包含游戏实体的2D或者3D的空间。场景可以是给玩家展示的任何东西：一个菜单、游戏关卡等。在新的项目创建时，初始场景被自动创建。通过菜单栏可以创建新的场景。场景文件有.unity形式的后缀名。

游戏实体是在场景中存在的实体。当一个新的场景创建时，它就已经包含了两个实体：镜头和光源。为了将场景呈现给玩家，场景至少拥有一个镜头。玩家可以通过层级视图、主菜单或者是拖拽实体的方式为场景添加新的实体。一个游戏实体可以作为另一个游戏实体的父节点。

1. 组件

游戏实体并不能进行任何操作，只有将组件附加到游戏实体上时实体才有功能。这就表示Unity在设计上使用了组件模式。组件模式是一个解耦设计模式，它用于以下的情况：当一个类与多个领域互相影响，这个类有必要将每个领域特定的功能解耦；或者是不同的实体共享多个相同的、不严格意义上继承的功能。Unity提供了与内部组件进行交互的简单方法：使用编辑器或者是程序。

在Unity中，游戏实体可以被激活或者设置为无效，组件也可以被设置为启用或者禁用。附着在无效实体上的组件都是禁用状态。这提供了实现状态设计模式的简单方法。状态设计模式是游戏中常用的设计模式，它的存在使实现像面向对象编程范式中的状态机成为可能（在状态机中的每一个状态都作为一个实体抽象出来）。在Unity中，这个模式可以被这样实现：激活一个实体表示当前状态，同时将其他状态实体设置为无效。



Figure 3 Unity编辑器

1. 脚本API

尽管Unity中包含了许多内置的组件，大部分游戏仍然需要额外的特定功能。这些功能可以用Unity的脚本API实现，使用方法是在一个特殊的组件：脚本中用C#，JavaScript或者Boo

编写。在我们的实现中，所有脚本都用C#编写。脚本API是一个IoC（反转控制）框架，开发者的脚本不会控制游戏的执行。开发者需要在脚本中实现具体的方法，让Unity在特定条件下去调用。脚本中最常见的方法是Start和Update方法。Update方法在每一帧渲染前被调用，所以大部分的游戏罗技都在Update方法中执行。Start方法在第一次Update调用前被调用。

1. Unity资源商店

Unity资源商店是一个用户可以买卖自定义资源的网上商城。在Unity编辑器中就可以打开Unity资源商店。在Unity资源商店购买的资源可以直接导入到Unity编辑器中。新下载的资源会被放在Asset文件夹下。像3D模型这样的资源一般都被附加到游戏实体上，而且会有一些简单的脚本或者组件附加在上面。其他资源也与Unity支持的格式匹配。

1. Photon Unity网络（PUN）

PUN是一个为了在Unity中开发在线游戏设计的框架。在Unity资源商店中可以找到PUN的两个版本：免费版本和付费的升级版本。我们的游戏Moghi Battles将使用免费版本。PUN通常非常易于使用，因为大多数框架的函数都是以一个类的静态成员的方式公开的，这个类既是外观模式中的一个外观，也是单例模式中的一个单例。

在PUN中有两个主要的概念：房间和大厅。连接到大厅在概念上就等于连接到会话服务器，而连接到房间就等于连接到游戏服务器。这说明，玩家必须在连接到游戏服务器之前首先连接到会话服务器。当验证成功后，玩家进入大厅，在大厅中玩家可以新建游戏房间。Photon服务器不模拟游戏，他们只负责提供一个方法发送游戏服务端模拟的游戏数据到其它客户端，然后在它们之中使用PUN同步游戏状态。这种交换数据的方式是使用以下同步机制实现的：实体同步，PUN RPC（远程过程调用）和自定义房间属性。

实体同步是用来同步简单的游戏实体状态（比如当前的生命值状态或者位置）和通过网络实例化游戏实体的。这可以通过为一个游戏实体添加一个特别的脚本来实现。PUN每隔一段时间就在各个客户端之间同步一次数据。PUN可以通过配置实现以UDP或TCP协议同步游戏实体数据。一般会使用UDP协议，因为同步性往往比一些细节重要，比如实体的具体位置。对于Moghi Battles来说也是一样。

PUN RPC（远程过程调用）允许在其他客户端调用脚本中的方法。这为客户端间的事件同步提供了一个简单的方法。PUN RPC一般用TCP协议来保证事件被正确地同步。PUN RPC可以是缓冲的或者非缓冲的。缓冲的RPC会被所有进入房间的客户端和之后进入房间的客户端调用。非缓冲的RPC只在进入房间的客户端上调用。

自定义房间属性是用来同步数值的键值对，它不属于任何实体。

1. 地图

在实现的游戏中，玩家可以在两张不同的地图中选择一张。地图用Unity场景来实现。这些场景包含以下三种概念的游戏实体：控制器、活动实体和被动实体。控制器是包含游戏特定逻辑脚本的空游戏实体，它不是任何其他实体行为的一部分。活跃实体和被动实体是包含组件的实体，这些组件使它们可以与场景中的其它实体进行交互，控制其他实体是否对玩家显示。在这个实例中，所有与物理有关的组件都是Unity内置组件，除了地形是用内置地形编辑器制作的以外，其他3D模型、纹理、例子效果都是从Unity资源商店下载的第三方资源。活跃实体包含实现游戏相关的行为的脚本。例如活跃实体可以是摄像机或是防御设施，被动实体是地形和不可破坏的环境物件。两张地图中的背景音乐也是从Unity商店中获得的第三方资源。

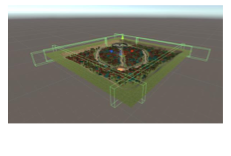


Figure 4 神圣森林地图编辑器内的场景视图，四周是高亮的不可见墙壁

玩家被不可见的墙限制在地图范围内。不可见的墙是没有纹理和组件只有碰撞属性（形状4）的游戏实体。玩家只能加入在等待状态或者是预备状态的比赛。每个新建的比赛都初始化为等待状态，创建的玩家自动进入一个新的房间，并且将当前的服务器时间设置为房间的自定义属性，当玩家数量增加时会对所有客户端发出一个缓存的RPC指令。当其他玩家进入地图，他们进入同一个游戏房间，并且通过缓存的RPC指令增加玩家数量。当每个队伍的玩家数量都达到最小标准时，比赛进入为时1分钟预备状态。当预备状态结束，比赛进入战斗状态，队伍开始对抗。当战斗状态时一个英雄死亡，它会在基地复活。

每张地图的最大玩家数量是3，最小玩家数量是1。当前的玩家数量和当前游戏状态会显示在玩家的屏幕上方。在同一队伍中的玩家可以通过发消息的方式进行聊天，聊天功能在左下角，也使用RPC的方式同步数据。

1. 神圣森林地图

神圣森林是MOBA中常见的森林主题的地图，它有三条路。与其他MOBA不同的是在这张地图中两个队伍的首要目标不相同。进攻方队伍的首要目标是在10分钟内摧毁防御方的建筑。需要摧毁的防御方建筑位于防御方基地的腹地，防御方的首要目标是抵挡住进攻方的攻势。防御方有包围基地的围墙和旋转的能攻击附近进攻方单位的火焰塔帮助防御。火焰塔的火焰伤害效果用旋转的不可碰撞子实体实现，它会用RPC机制让进攻方的玩家受到伤害。与另一张地图比较，这张地图包含了更多游戏实体（一共823个游戏实体，大部分是人工放置的）。

1. 沙漠地图

沙漠地图包含了沙漠的设定和特点，它只有一条路。两个队伍的主要目标是得到更高的杀敌数。一个英雄对另一个英雄造成足够伤害能使其死亡，这算作控制造成伤害英雄的玩家队伍的一个击杀。这张地图也包括了一些可以被捡起来的提升能力的物品散布在地图中。这些提升能力的物品也是有触发模式的碰撞体组件的游戏实体。这张地图比另一张地图小很多，提供了更令人激动的游戏内容但是欠缺战术选择。

1. 英雄和技能

在Moghi Battles中，玩家可以选择两个英雄中的一个。这两个英雄有不同的属性和技能树，这使它们有着完全不同的玩法。英雄属性是：初始生命值、攻击、防御和速度。英雄由主要游戏实体和两种子实体构成。第一种表现静态英雄外观（3D模型和纹理）；第二种定义了区域和技能效果。3D模型、纹理和动画都是从Unity资源商店中获取的第三方资源。在主游戏实体和第二种子实体中都包含了内置物理组件。主对象还包含了同步所需的PUN脚本和用来控制动画转换的内置组件。一开始，组件不包含任何转换，游戏开发者需要为组件定义转换。其他主实体上的脚本实现了游戏逻辑：如移动、资源、属性追踪和技能相关的逻辑。所有技能图标、技能例子效果和英雄音效都是Unity资源商店中的第三方资源。

英雄可以通过按键W、A、S、D来进行八个方向的移动。其他游戏也常用这个键位来移动。按下相应方向的键后，英雄主实体会执行一个方向向量。如果超过一个按键被按下，这些方向向量将被正交相加。最终的方向向量将会是将总正交方向向量与速度属性相乘 的结果。

释放技能会花费金币。金币是Moghi Battles中随着时间自动生成的可以消费的资源。通过摧毁敌方建筑和英雄，玩家可以获得额外的金币。每个英雄有三个技能，两个伤害技能和一个增益技能。一个伤害技能的冷却时间很短，金币花费也很少，伤害也很低。另一个伤害技能有着更强的伤害，不过冷却时间和金币花费也相应增加。每个英雄的增益技能都非常强大，它有这最长的冷却时间和最贵的金币花费。每个技能都以一个继承了抽象类的脚本代表，这个抽象类实现了所有这些技能所需要的接口，比如等待玩家输入和管理例子效果。派生出来的类只需要实现技能生命周期中三个阶段调用的三个不同方法。这也是模板方法设计模式的实现。技能条显示在玩家的屏幕下方，技能条会为玩家显示哪个技能可以释放，哪个技能在冷却中，哪个技能没有足够的金币进行释放。当前玩家的金币数量显示在右上角。

伤害技能用来击败其他玩家，摧毁敌方的防御设施。在这些技能开始的时候，一连串伤害效果会被定义。开始时间被定义为技能释放时的时间差。每个伤害效果有一个与之对应的范围效果。当时间经过了一个伤害效果的开始时间，游戏会寻找所有在效果范围中的可破坏实体，用RPC机制通知这些实体它们将会受到伤害。每个可伤害的实体都有生命条，显示它们已经受到的伤害和还可以受到的伤害。

增益技能释放时会改变玩家的属性。到增益结束，属性被还原。在Moghi Battles中，这个行为是按照命令设计模式实现的。

1. 与其他玩家测试

游戏经过了一组四个玩家的测试。两个玩家有MOBA游戏的经验，另外两个没有。一共包含两次进行整个比赛的测试。

1. 第一次测试

在第一次测试当中出现了一些问题。有时游戏不能正确的将玩家放入队伍中，预备状态会在正常的预备情景之前被启动。在客户端中同步玩家数量的代码存在错误导致了这个问题。另一个问题是其中一个英雄与另一个英雄比太过于强大。

1. 第二次测试

在第一次测试的问题被修复之后我们进行了第二次测试。没有发现其他重大问题，只有一些优化不足导致的小的表现上的问题。

1. 结论

尽管Unity有很多在线教程，其脚本API也有很好的文档，但是总的印象是，Unity的学习曲线非常陡峭，这意味着初学者必须学习许多概念才能展开更高级的项目。性能方面，即便没有优化，使用Unity实现的游戏也运行的相当好。在开启V同步的情况下观察到了一些运动不连续，关闭V同步之后问题被解决。尽管脚本API是一个反转控制框架且限制了可以实现哪种算法，但是它非常直观且易于使用。让Unity更值得选择的原因之一是它的资源商店，拥有非常多的第三方组件，可供开发者随时使用。一共创建了7个场景（2张地图和5个菜单），59个类。



Figure 第二次测试Moghi Battles的游戏内截图

译文二：

星际争霸中基于势场的寻路机制[[2]](#footnote-2)

摘要：即时战略（RTS）游戏是通常发生在战争环境中的战略游戏的子类型。即时战略游戏为人类玩家和计算机玩家（机器人）提供了丰富的挑战。每个玩家都有一些可以采集资源和建造建筑的工人，训练更多的工人，建造作战单位，研究科技解锁更强大的单位和能力。游戏的目标是集结一支强大的军队，摧毁敌人的基地。军队通常由大量能够在游戏中移动的单位组成。即时战略游戏中的高度动态性和实时性使得寻路成为电脑的一项艰巨的任务。一般来说会使用A\*等寻路算法进行处理，但是如果不经过特殊适配，这些算法不能很好地适应动态世界。在这篇论文中，我们将展示一个星际争霸的AI将如何使用A\*和势场的组合来更好的处理游戏中的动态地形。

1. 简介

即时战略游戏为玩家和电脑控制的玩家（bot）提供了非常多的挑战。在游戏一开始，玩家会拥有一个指挥中心和一些农民。农民需要从资源点采集一种或者多种资源，然后把资源放回指挥中心。资源可以用来建造建筑，扩张你的基地，或者是用来建造可以进攻敌人、防守自己基地的作战单位。在即时战略游戏中，科技树也很常见。玩家可以在科技上投资资源，升级单位和建筑。当一个玩家用实力和操作摧毁另一个玩家的所有建筑时，游戏结束。

在即时战略游戏中，单位的移动一般是用一些寻路算法来实现的，最常见的寻路算法就是A\*算法。不过A\*算法不能很好地在动态的世界中运行。算法找到的路径是静态的，但是单位沿着这条路径移动需要时间。如果这条路被其他移动的物件挡住，之前的这条路径就会被废弃，代理必须重新计算整个部分。已经有大量的工作优化A\*算法，使其能够在高度动态的世界中更好的运行。Silver提出寻路图额外的时间维度，使单位能够在特定时间预留节点。Olsson的工作解决了由建造建筑或者建筑被摧毁导致的寻路图变化的问题。Koenig和Likachev在实时A\*上的工作也对这个领域做出了贡献。

势场是一个来自机器人领域的概念。它是由Khatib首次提出的，用于机器人和移动机器人的实时避障。它通过在虚拟世界中的重要位置放置吸引或者排斥电荷来工作。在目标点放置一个吸引电荷，在障碍物的位置放置排斥电荷。每个电荷都会生成特定大小的电场。障碍物位置附近的排斥电荷的电场通常很小，而在目标点的吸引电荷的电场必须覆盖大部分的地图。这些不同的电场被加权聚集在一起形成一个集合电场。集合电场通过让单位移动到它附近引力最强的点的方式来进行导航。许多关于势场的研究与空间导航和避障有关，例如Borenstein和Massari的研究。Alexander介绍了游戏Blood Wake和NHL Rivals中使用场进行避障的应用。Johnson描述了游戏The Thing用场来进行避障的方法。

在Hagelback和Johansson的一系列论文中，他们讨论了在开源即时战略游戏ORTS中使用势场来移动。他们建议为游戏中每一个有趣的对象，友善的、中立的或者敌对的都根据当前的代理类型和影响的对象建立一个势场。

如果自身单位需要移动到一个目标点，一个引力场也将被放置在目标点。这个场在目标点引力最强，之后线性或者指数性减弱至0。

在如其他自己的单位之类的障碍物附近的场比较小，并且在一定的距离排斥自身单位经过障碍物附近。

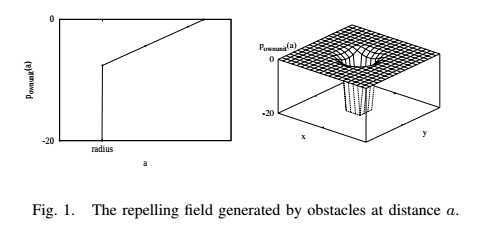


图 1 一定距离上障碍物产生的斥力场

由敌方单位生成的场的引力是用一个分段的线性函数或者用一个指数函数来生成的，并且在某个非零距离上值最高。这样的原因是我们希望让自身单位与敌方单位保持一定的距离，一般来说保持的距离是自身单位的最大攻击距离。这样会让我们的部队在最大射击距离包围敌军，因为这是引力场最大的地方。图2展示了一个地方单位产生的分段线性场。MDR表示最大视野范围。在这个范围之外自身单位不会被敌方单位吸引。

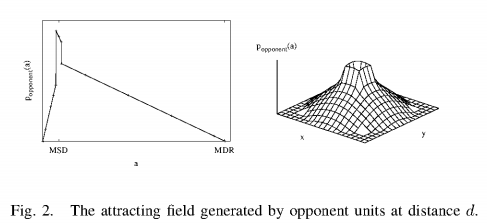


图 2 在一定距离上敌方单位产生的引力场

图3展示了一张星际争霸的截屏，一些人族攻城坦克用多段线性势场（用蓝色的线表示）在最大攻击距离包围虫族的基地。

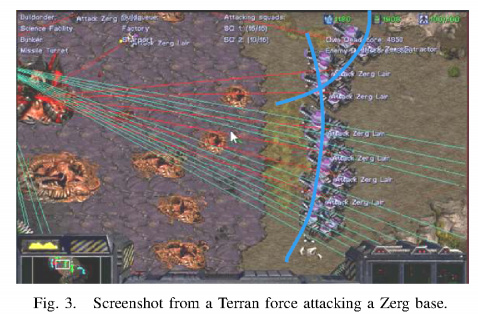


图 3 人族部队攻击虫族基地的截屏

当用势场作为寻路方式时，总场使用等式1计算，N是影响位置的物体总数，是第个子场的权重。自身单位检测在它自身一定范围内的电势。检测的范围通常是单位可以在一帧中可以移动的距离。随后，自身单位向电势最高处移动。注意，这篇论文中更高的电势引力更大，但是在一些其他文章中，更高的电势会有更低的吸引力。

（等式1）

这个解法的一个问题是自身单位有可能在一个局部最优解上卡住。这意味着电势做大的地方是自身单位当前的位置，但是这个位置不是最终的目的地。这个问题一个可能的解法是当自身单位卡主时让它在一个随机方向上移动一段距离，但是这不能保证，因为自身单位有可能移动回原来的位置再次卡住。一个更鲁棒的解法是将一个小斥力场放在自身单位之前所在的位置，类似于蚂蚁使用的费洛蒙路径。其效果是尾路径会推动自身单位向前，并且阻止它从来的方向回来。这个解法在ORTS中成功地使用，其中使用了20个位置的尾路径。

图4中上侧的图片展示了如果总势场宗师被计算为所有子场加权和可能发生的另一个效应。如果几个敌方单位在同一地区，像即时战略游戏中经常发生的那样，总场中电势最高的地方一般是在地方军队的中心。这会让我们的自身单位移动到敌方兵力的最强点。

建议的解决办法是用等式2计算总场。等式的第一部分与等式1相同，只是这里的N表示除了敌方单位和建筑以外其他的引力物体。第二部分表示所有敌方单位和建筑产生的场的最大值M，而不是他们的加权和。这样的算法在图4底部的图片进行展示。这个改变很好地提高了ORTS中电脑的表现。

（等式2）

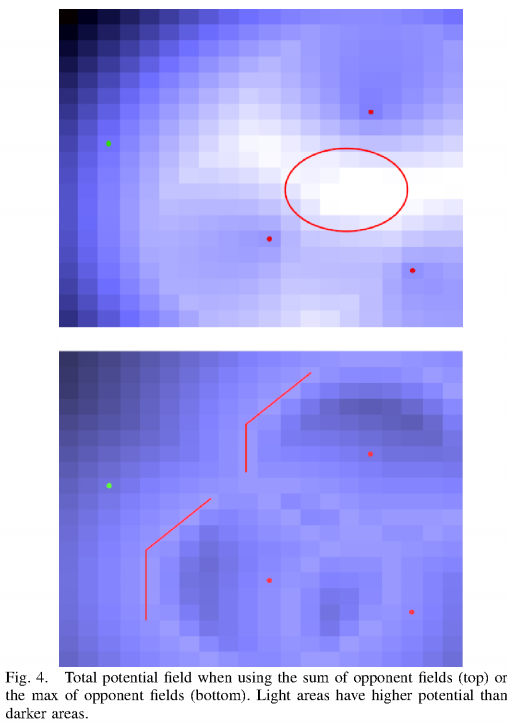


图 4 当使用敌方总势场时的总势场图(上图)使用敌方最大势场时的总势场图(下图)量的敌方比按得敌方势更高

另一种建议的改进是不仅让产生的场受自身单位和引力单位影响，而且让自身单位的内部状态也影响。在图5的上图中自身单位处于攻击状态，敌方单位产生的场在自身单位的最大攻击距离有着最大的电势。在图5的下图中自身单位在撤退状态中，电势最高的地方是在地方单位的最大攻击距离之外的地方。

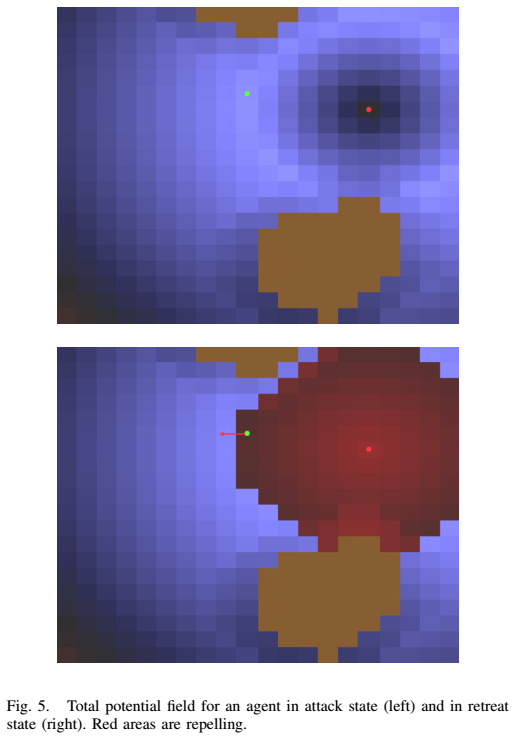


图 5 代理单位在攻击状态的总势场(上图)在撤退状态的总势场(下图)红色区域表示排斥

1. 星际争霸中的势场

我们已经实现了一个著名即时战略游戏星际争霸：母巢之战的电脑AI，它使用BWAPI框架与星际争霸引擎进行交流。它使用多种寻路方法：A\*（星际争霸内置的寻路方法），势场或者是两者结合。这个电脑AI已经开源并且可以下载。

在之前的工作中，基于势场的寻路系统是实现在ORTS引擎上的，并且在Tankbattle这个场景中测试过。在这个场景中，只有一种单位（坦克）和一种建筑（指挥中心）。星际争霸是一个有着非常多不同属性的单位和建筑的复杂的游戏。比如星际争霸中有地面单位和空中单位。有些单位只能攻击空中目标，有些只能攻击地面目标。也有单位即可以攻击空中目标也可以攻击地面目标，但是可能有不同的最大射击距离。星际争霸中也有不能直接攻击敌人的单位，比如神族的告诫圣堂武士或者人族的科技球。这些单位有一些辅助的功能（科技球可以反隐）或者用消耗能量的特殊技能进行攻击（高阶圣堂武士的灵能风暴）。

障碍物产生的势场遵循简介中的产生法则。每个障碍物产生一个小的斥力场，其大小为自身单位加上障碍物的搬家加上相互通过的最小距离，这在我们的视线中是人族陆战队员大小的一半。唯一例外的是隐形单位神族的黑暗圣堂武士和虫族的潜伏者。除非对手有一个反隐单位在附近，否则潜伏者和黑暗圣堂武士不会被敌人看到。他们仍然可以被一些范围性伤害单位攻击到，比如人族的攻城坦克。为了避免受到不必要的伤害，黑暗圣堂武士和潜伏者需要在他们自己和其他单位之前保持工程坦克最小伤害溅射范围半径的距离。

敌方单位产生的势场由以下单位属性影响：

* 自身单位是否能攻击对手
* 对手是否能攻击自身单位
* 自身单位在进攻或者是防守的状态
* 自身武器的射程是否能攻击到敌方
* 敌方武器的射程是否能攻击到自身单位
* 自身单位的视野半径，即自身单位是否能看到敌方单位

根据这些属性值的不同，敌方单位将产生不同的场。这些场可以是吸引或者排斥的，有不同的大小，在距离敌方单位不同的距离处有最高的势场。表I展示了单位属性和产生的场的不同关联。注意，当自身单位不能攻击到敌方单位或者敌方单位不能攻击到自身单位时是一种特殊情况。如果自身单位能够用特殊技能攻击，会产生一个场，否则不会生成场。在我的实现中在一个场中最高的势设为浮点数200。在实际应用中，这个值可以是任何值，不过它应该足够高，来产生一个覆盖整个游戏世界逐渐减弱的场，并且在两个可调整的位置之间有着逐渐升高的势场值（即，它取决于游戏世界的大小，和实现中用整数，浮点数，双精度浮点数值来代表这个值）。

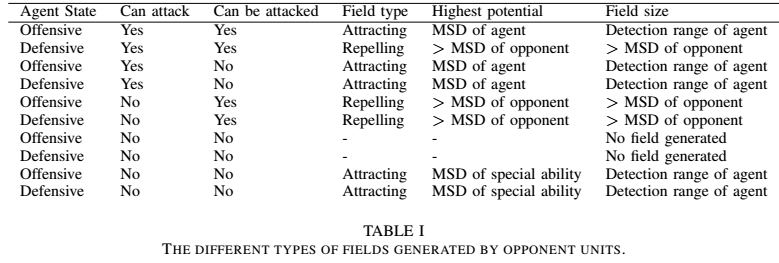


表 1 敌方单位生成的不同种类的场

就像之前在势场寻路系统上的研究一样，用激发费洛蒙路径的方式避免在局部最优解中卡主。总势场用等式2进行计算避免在敌军中心出现引力场最强的情况。

在之前的研究中使用ORTS引擎根据一个随机数种子随机生成地形。最终生成的地形一般都有很大的空旷地形和一些很窄的通路，一般可以不用绕太多的路就可以避开。这对基于势场的寻路系统非常理想，因为只有很少的区域会造成局部最优解问题，并且其中的一部分可以通过填充窄路和地形凹陷的方法避免。

在一张典型的星际争霸地图中，地形会更加复杂。一般只有一条狭小的路通向基地，而防守住这条路就是取得胜利的关键。空军单位没有任何限制，可以移动到地图的任何位置。大量使用的阻塞点和小路会让基于势场的寻路系统产生很多局部最优解。

我们建议用A\*和势场结合的方式来1）当A\*适用时，可以用它，以减少用势场方式造成的局部最优解问题。2）用势场的方式更好的适应动态的游戏世界。使用结合的寻路方法时，当没有敌方单位被自身单位看到，则使用A\*算法。这提供了长距离中有效的寻路机制。当敌方单位或者建筑被发现，寻路方式切换为使用势场的方式。这样，大部分的局部最优解问题将被避免，而且仍然能够发挥势场方法的优势，比如能够包围敌方单位。

图6展示了星际争霸中的截图，人族工程坦克的势场值被显示。图7显示了人族陆战队员视野中同样的情形。注意引力场最强的地方（亮色区域）在距离敌方单位不同的位置，因为攻城坦克和陆战队员的最大攻击距离不同。

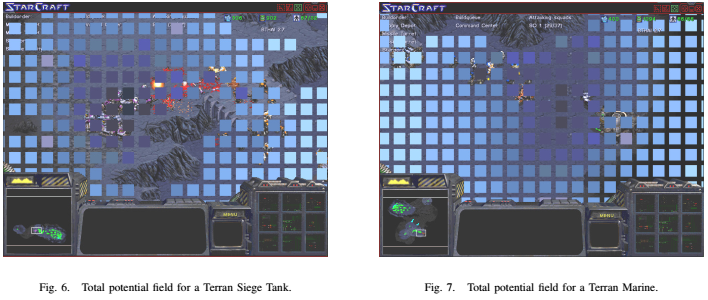


图 6 人族攻城坦克的总势场 图 7人族陆战队员的总势场

1. 试验

我们用两种电脑AI进行了一系列试验，我们的电脑用人族对抗星际争霸内置的人族、神族、虫族AI，使用A\*与市场结合的算法来寻路。每个电脑AI在两张地图上对三族打两场比赛。

选用的地图是：

* 终点1.1。一张两人地图，是AIIDE 2011 星际争霸电脑AI竞标赛地图库中的一张地图。
* Fading Realm。一张两人图，有着不同高度的很多高地。

测试的结果在表II中显示。结果显示内置的AI都不是我们两个版本的电脑AI的对手，赢下12场中的11场，输的那场都是被神族的速狂热者一波打死。两个电脑AI之间的分数差距也非常小。为了更好地比较，我们做了第二个试验，让两个电脑AI版本之间互相对抗。这个试验的结果在表III中显示。结果显示，将A\*和势场方式结合是更好的方案，它赢下了6盘中的5盘。

注意，两个版本都用的是同一套战术，同一套建造流程，兵种组合等。唯一的区别是当单位靠近敌人或者敌方建筑时是否使用势场寻路方式。

我们也进行了一些游戏只用势场的方式进行寻路。这些游戏都以平局收场。电脑AI在防守方面非常强，但是地图中的很多阻塞点就算用20位置的尾路径解决局部最优也也还是太复杂。电脑AI无法将足够的军队移动到敌人基地去取得游戏的胜利。所以想要在一张正常的星际争霸地图中应用势场的方法，需要一个更有效的局部最优解的解法。

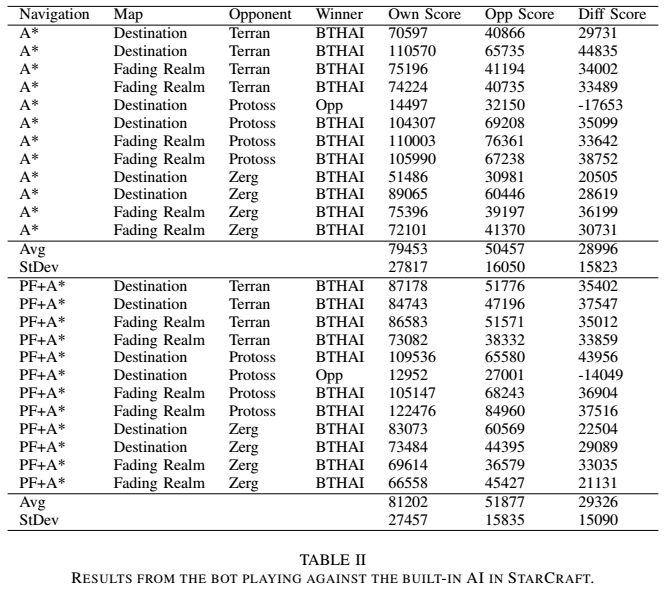


表 2 电脑AI与星际争霸内置AI对战的结果

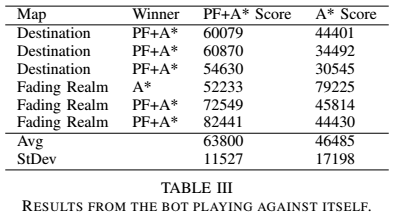


表 3 电脑AI与自己对战的结果

1. 讨论和未来的工作

这两个试验的结果显示了A\*和势场结合的寻路系统比只用A\*寻路表现的更加优秀。就像论文之前提到的这样，基于势场的方式可以用多段线性或者指数性子场包围地方单位。当用A\*和势场结合的方法时，很重要的一点是不要太晚切换到势场方法，因为这样会让自身单位没有时间去移动到包围地方单位的位置。同样的，也不要太早切换，因为这样会有陷入局部最优解的可能性。在几乎所有情况下，星际争霸中单位的视野比他们的最大射击距离长一点（除了攻城模式下的攻城坦克），可以用它们的视野作为寻路方式转换的标线。

仅基于势场的寻路系统不能正常运行，这是由于星际争霸地图的复杂性和无数的局部最优解问题。表8展示了对于只用势场寻路非常不友好的一张地图。截图显示了这张地图的上半部分，显示了其中一个玩家的开始点。另一个玩家的开始点在下方。黄色的圈显示开始点的唯一出入口。如果给自身单位发布一个向对方开始点移动的命令，最高的引力势点会在自己出生点的南方，自身单位会错过北方的出口。这个可以用更好的局部最优解的解法来解决，但是这已经超出了本论文的讨论范围。

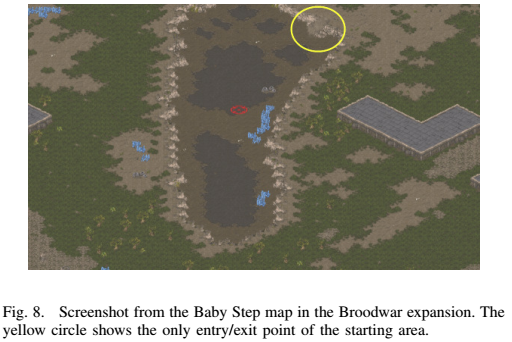


图 8 母巢之战中Baby Step地图的截图。黄色圈中显示了这个地区的唯一出入口

未来的工作一个可能的方向是检测阻塞点，比如用A\*算法，在阻塞点放置引力子目标。这样有可能解决大量的在使用势场方式遇到的局部最优解问题。

在试验中使用的电脑AI中的一个是2011星际争霸AI比赛的早期版本。这个电脑AI玩虫族，使用比较大胆的地刺一波战术，在比赛中拿到了13名中的第10名。在那之后修复了很多小问题（这个AI有13%的可能性会崩溃）和一些较小的改进。看现在这个电脑AI参加比赛会非常有趣，我们准备参加2012星际争霸电脑AI比赛。

**外文翻译考核表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指导教师对外文翻译的评语：**  **指导教师 （签名）**    **2017年 3月 12 日** | | | | |
|  |  |  | **建议成绩** |  |
| **评阅小组或评阅人对外文翻译的评语：**  **评阅小组负责人或评阅人 （签名）**    **2017 年 3 月 12 日** | | | | |
|  |  |  | **建议成绩** |  |

1. D. Polančeć, I. Mekterović. Developing MOBA games using the Unity game engine[C]// International Convention on Information and Communication Technology, Electronics and Microelectronics. IEEE, 2017:1510-1515. [↑](#footnote-ref-1)
2. Hagelback J. Potential-field based navigation in StarCraft[C]// Computational Intelligence and Games. IEEE, 2012:388-393. [↑](#footnote-ref-2)