南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

本科生毕业设计（论文）

题　　目： 面向 MOBA 的场景设计及验证

姓　　名： 赵 宇

学　　号： 11612908

系　　别： 计算机科学与工程系

专　　业： 计算机科学与技术

指导教师： 刘佳琳

2020 年　5　月　12　日

诚信承诺书

1.本人郑重承诺所呈交的毕业设计(论文),是在导师的指导下,独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。

2.除文中已经注明引用的内容外,本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。

3.本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4.在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

作者签名:赵宇

2020 年 5 月 12 日

面向 MOBA 的场景设计及验证

赵 宇

南方科技大学计算机科学与工程系 指导教师：刘佳琳

[摘要]：多人在线战术竞技游戏（multiplayer online battle arena，MOBA）有多样化的游戏场景以及可供玩家摸索的多种游戏策略，具有很高的游戏性，从而在游戏市场上表现出色。然而也因为这些场景的多样性和多玩家间的博弈，给训练 MOBA 游戏智体带来了许多困难和挑战。本项目针对增强学习智体在不同场景下的训练难度和学习能力问题，使用一个基于 OpenAI Gym 的 MOBA 测试平台，MOBA-Env，研究场景的设计和多智体的训练，并完成以下几个主要工作：（i）完善 MOBA-Env，在该环境中加入了地形的定义，并对其中的控制单位设计了新的最短路算法控制其移动攻击；（ii）根据实际 MOBA 游戏中的经典策略设计多种场景（如放风筝、不同难度的轮流抗血场景等）并添加到 MBA-Env，对于不同场景分别定义了获胜条件和奖励函数；（iii）调研了近端策略优化算法（proximal policy optimization， PPO），在设计的场景下训练 PPO 智体，并将理论奖励与实际奖励进行对比，分析 PPO 算法在不同难度场景下的表现。

[关键词]：MOBA 游戏，强化学习，游戏 AI，多场景

[Abstract]: Multiplayer online battle arena (MOBA) has a variety of game scenarios and a variety of game strategies that players can explore. It has a high degree of gameplay and thus performs well in the game market. However, because of the diversity of these scenes and the game between multiple players, it has brought many difficulties and challenges to the training of MOBA game intelligence. This project aims to enhance the training difficulty and learning ability of learning intelligence in different scenarios, using a MOBA test platform based on OpenAI Gym, MOBA-Env, research scene design and multi-intelligence training, and complete the following main Work: (i) Improve MOBA-Env, add the definition of terrain to the environment, and design a new shortest path algorithm for the control unit to control its mobile and attack; (ii) According to the classic strategy in the actual MOBA game, designed a variety of scenarios (such as kite flying, anti-blooding scenarios of different difficulty, etc.) are added them to MBA-Env, and define the winning conditions and reward functions for different scenarios; (iii) Investigate Proximal policy optimization algorithm(PPO), training the PPO agent in the designed scenarios, comparing the theoretical reward with the actual reward, and analyzing the performance of the PPO algorithm in different difficulty scenarios.
[Keywords]: MOBA, Reinforcement learning, Game AI, Multi-scenario.

目录

1. 引言

MOBA 是 Multiplayer Online Battle Arena 的简称，即多人在线战术竞技游戏。比较出名的 MOBA 游戏有英雄联盟、DOTA、王者荣耀，在这些游戏中，玩家需要控制一名英雄（以及这名英雄召唤的单位），通过团队合作打败对方，以摧毁敌方基地为获胜条件。

MOBA 游戏是大部分游戏爱好者很感兴趣的游戏类型，一般的 MOBA 游戏具有可供玩家摸索的游戏策略、多样化的游戏场景，这些特性使 MOBA 游戏具有很高的游戏性，也让 MOBA 游戏在游戏市场上表现出色。随着 MOBA 游戏的逐渐发展，关于 MOBA AI 的研究也逐渐发展起来。近期最为出名的 MOBA AI 是 OpenAI 的 OpenAI-Five[1] 在 2019 年的 8 月 15 日挑战 2019 年 DOTA2 世界冠军 OG 战队获得胜利，DOTA2 被认为是 MOBA 游戏中游戏机制最复杂、决策最多样的 MOBA 游戏。OpenAI 训练的 MOBA AI 战胜了 DOTA2 的世界冠军队伍也向人们展示了，在不注重操作更注重决策的 MOBA 类游戏里，AI 可以表现的比大多数人出色。

OpenAI 训练 OpenAI-Five 使用的是 PPO 算法，我们也选用了 PPO 算法作为我们的训练算法，OpenAI 使用的 PPO 算法是策略梯度方法[5]的一种，在实际场景中，MOBA AI 需要根据场景的状态（敌方血量、我方血量、位置、防御塔等）做出下一步的决策（移动方向、移动距离、攻击目标等），而这些决策往往是连续的，PPO 算法可以用于这种连续决策情境下的决策过程。

MOBA 游戏的场景是多样的，防御塔、地形、视野、位置、野怪、蓝量、技能 CD 等都有可能成为影响决策的因素，而在不同场景下我们期望得到的结果也不同，比如一场小规模团战，对方随时可能支援，我们可能更希望尽快将对面击杀，而如果对方不可能来支援，我们往往希望己方不死且尽可能的掉更少的血。这需要我们合理设计奖励函数，并且使用大量场景对 AI 进行训练，确保 AI 能处理所有可能的游戏场景。

主要贡献和成果

- 根据 A*算法设计了适合 MOBA 游戏中单位移动攻击的 MOBA 最短路算法，该算法首先改变图中少量点的连边规则，再通过 A*算法进行求解，最后扣除路径中多余的节点并还原图。优化常数和具体目标小兵或英雄有关，约为 60 到 360。
- 设计了风筝场景、寻路场景、轮流抗血场景（易）和轮流抗血场景（难），根据每个场景的需要设计了英雄的血量、攻击力、移动速度等数据，并设置了这些场景的获胜条件（赢、己方单位全部存活等），分析了场景数据设计的合理性。
- 根据以上场景分别设计奖励函数，计算各个场景下的理论最优回报，使用 PPO 算法训练 AI 决策，并将最终收敛的决策值的回报与理论最佳值对比。

2. 背景介绍

2.1 MOBA 游戏

MOBA 游戏一般是在一张地图上的 5v5 游戏，一般分为选人阶段，对线期，发育期和后期。MOBA 游戏复杂的决策和多样的场景在这里就已经体现出来：选人阶段，决策的玩家需要考虑己方阵容的合理性、针对敌方阵容和防止被敌方后选针对、所选择英雄在当前版本下的强度等。对线阶段，决策的玩家需要计算小兵的血量，通过自己的攻击力和弹道判断什么时候出手补兵，对方的血量和技能，对方英雄可能的游走或者队友游走提前处理兵线进行配合等等。后期基本所有的 MOBA 游戏都在一直打团，这个时候的决策更加多样化，比如谁先手对方，先手谁，利用视野优势能否让对面减员，肉盾能否通过站位承受更多伤害，输出能否通过走位或者辅助的帮助使输出最大化，打赢了团是需要推进还是打龙等等。多样的场景和决策是 MOBA 游戏吸引玩家的一部分，同时也是 MOBA AI 面临的挑战。大部分的场景不可能靠单一英雄去完成，而是需要合作，通过合理的团队决策达到目标。单智能体的 AI 很难玩好 MOBA 游戏，MOBA 的合作性强调了 MOBA AI 必须是多智能体训练的结果。

2.2 MOBA AI 训练环境

本章介绍常见的 MOBA 训练环境，并分析其优缺点，并介绍本项目中使用的 MOBA 环境。

2.2.1 DOTA2

目前比较流行的 MOBA 游戏训练环境是 DOTA2，这主要是因为 DOTA2 提供了很多现成的接口供脚本调用，使用 DOTA2 进行 AI 训练是很方便的选择，但是相应的缺点也非常明显：一个可用的 AI 必须在对应游戏的任何可能场景下都能做出决策，DOTA2 的游戏逻辑非常复杂，所以用 DOTA2 进行 AI 训练需要消耗大量的训练资源，而对于算法的验证，并不需要如此多的游戏元素，所以使用 DOTA2 进行训练进而验证强化学习算法可能导致运算资源的浪费。

2.2.2 OpenAI Gym[3]

OpenAI 的 Gym 也可以用来训练 MOBA AI，如果使用 OpenAI Gym，MOBA 游戏的游戏逻辑需要自己实现，可以只实现自己需要的游戏逻辑。OpenAI Gym 的优点是它提供了很多用于强化学习的接口，这些接口是标准化的，这也就解决了定义上的微小差别可能对结果造成很大影响的问题。

2.2.3 环境选择

为了验证 PPO 算法在不同场景下的表现，我们需要设计很多独立场景进行训练，这些场景只需要部分地图和部分英雄就可以实现。从验证算法的角度，使用 DOTA2 或者 OpenAI Gym 都可以对算法进行验证，而 OpenAI Gym 不需要统一各个算法之间的微小定义差别。从计算资源的角度，一个场景只是游戏的一部分，场景外的数据是多余的，比如下路的团战，计算上路的野怪、兵线对 AI 来说就是多余的。综合使用的需求，我们基于 OpenAI Gym 开发了自己的 MOBA 游戏环境 MOBA-Env。MOBA-env 使用 OpenAI Gym 作为 MOBA AI 的训练环境，使用 Go 语言实现了简单的 MOBA 游戏，使用 Python 语言建立 OpenAI Gym 与游戏的交互，使用 TensorFlow[7]作为强化学习框架。

2.3 PPO 算法

PPO 算法是一种优化的策略梯度方法(Policy Gradient Methods, PG)，他解决了策略梯度方法的步长很难选取的问题，同时数学形式不像**信赖域策略优化**

(Trust region policy optimization, TRPO)那样复杂。PPO 算法有两种实现，一种是 PPO-Penalty,另一种是 PPO-clipped。

2.3.1 策略梯度方法与信赖域策略优化

PG 首先计算策略梯度的估计，常用计算策略梯度估计的方式是计算$\hat{\mathrm{E}}_t\left[\nabla_\theta log\pi_\theta(\alpha_t|s_t)\hat{\mathrm{A}}_t\right]$，其中其中$\theta$是策略参数，$\pi$是随机策略，$\hat{\mathrm{A}}_t$是时间 t 的估计优势，$s_t$代表状态，$\alpha_t$代表行为，$\hat{\mathrm{E}}_t$表示在有限次样本下的经验估计期望。基于这个策略梯度的损失函数是

$$L^{PG}(\theta) = \hat{\mathrm{E}}_t\left[log\pi_\theta(\alpha_t|s_t)\hat{\mathrm{A}}_t\right]$$

根据这个损失函数更新策略的缺点是他并没有对每一步更新的幅度(步长)进行限定，如果步长太小，那么策略的更新将会非常缓慢，如果步长太大，可能会忽略一些很好的行为或者导致策略并不会收敛（在比较大范围内波动）。

而 TRPO 利用更新后策略的比来作目标函数的一部分：

$$\underset{\theta}{maximizes}\ \hat{\mathrm{E}}_t\left[\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}\hat{\mathrm{A}}_t\right]$$

当作要优化的目标函数，每一步要让这个值在满足$\hat{\mathrm{E}}_t\left[KL\left[\pi_{\theta_{old}}(\cdot|s_t),\pi_\theta(\cdot|s_t)\right]\right] < \delta$的情况下尽可能的大，其中 KL 是指 KL 散度系数[9]。TRPO 一定程度上解决了 PG 的步长问题，他约束了步长更新的上限，并让步长的更新尽可能更快。但同时他的缺点是：数学形式复杂、实现难度大、在不同情境下需要更换约束条件，不能自适应的更换步长的约束。

2.3.2 PPO-Penalty

PPO-Penalty 使用自适应 KL 散度系数，使用 KL 散度调整罚分系数。KL 罚分的目标函数是

$$L^{PPO-Penalty}(\theta) = \hat{\mathrm{E}}_t\left[\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}\hat{\mathrm{A}}_t - \beta KL[\pi_{\theta_{old}}(\cdot|s_t),\pi_\theta(\cdot|s_t)]\right]$$

其中β表示自适应罚分系数。通过计算$d = \hat{\mathrm{E}}_t[KL\left[\pi_{\theta_{old}}(\cdot|s_t),\pi_\theta(\cdot|s_t)\right]]$,更新自适应罚分系数$\beta$的值，当 d 小于一个设定值，表明策略更新的太慢，则β适当减小（变为$\beta/2$），当 d 大与一个设定值，表示策略更新太快，此时加大β(变为$\beta*2$)，β的初始取值对结果影响非常小，因为算法中β的自适应性使结果对β的

初始值并不敏感。数学形式上来看，PPO-Penalty 更像 TRPO，可以看到二者的目标函数是非常相似的，PPO-Penalty 定义了自适应罚分系数的更新规则。效果来看，在具体实验中，效果不如 PPO-Clipped 优秀。

2.3.3 PPO-Clipped

PPO-Clipped 没有使用 KL 散度去对策略的更新进行任何约束，而是使用了目标函数中的一个比值$\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}$，通过对比值的约束$clip(\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}, 1-\varepsilon, 1+\varepsilon)$，目标函数为：

$$L^{PPO-clipp}(\theta) = \hat{\mathrm{E}}_t\left[min(\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}\hat{\mathrm{A}}_t, clip(\frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}, 1-\varepsilon, 1+\varepsilon))\right]$$

ε是论文中提到的参数，为 0.2。这个约束可以防止每一步的步长离上一步太远同时又不会让策略的更新速度太慢。PPO 的这一种实现非常简单，因为省去了很多值的计算，但是一般表现反而比 PPO-Penalty 更加优秀。

我们使用的基线系统提供了二者的算法实现，但是实际使用中并不会直接使用基线系统中的 PPO-Clipped，因为即使约束新旧策略的比值，但是仍然有可能出现新策略和旧策略相差很大的情况，所以实际实现中，会固定两个策略的 KL 散度值的上限（不是使用 KL 散度做罚分系数），超过这个散度值的策略将会被舍弃。

2.4 单源最短路 A*算法[8]

单源最短路 A*算法对于一张无向图 G，起始点 S 和终点 E，伪代码如下：

Algorithm 1 AStar(S,E,G)

OPEN: 已经生成但是未拓展的节点,使用优先队列
CLOSE: 拓展过的节点
g(n): 从起点到 n 的路径长度
h(n): 启发式函数，这里使用两点间的曼哈顿距离作为启发式函数
hfunction(x, y): 节点 x 到 y 的曼哈顿距离
f(n) = g(n) + h(n)
E_{ab}: a 到 b 的边的长度
E_n:和 n 相连的所有边
Sv: start node
Ev: end node
put A into OPEN, set f(A) = h(A) + 0
while OPEN is not empty:

```
    Find the node T in OPEN which has the lowest f(T)
    if T is B:
        return g(B)
    remove T from OPEN
    add T to CLOSE
    for each P in E_T
        if P in OPEN:
            cost = g(T) + E_TP
        if P in OPEN and cost<g(P)
            remove n from OPEN
        else if P in CLOSE and cost<g(P)
            remove n from CLOSE
        else if P not in OPEN and P not in CLOSE
            g(P) = cost
            h(P) = hfunction(P,B)
            f(P) = g(P) + h(P)
            add P to OPEN
    end for
end while
return NULL(No path)
```

3. MOBA 最短路算法

本章介绍了我设计的 MOBA 游戏中的一种最短路算法，可以适用于小兵、野怪和英雄等需要移动和攻击的单位。MOBA 地图可以被看做是一张无向图，其中有一些障碍点（地形）不可以移动穿过，但是可以跨地形攻击（只要有视野）。当单位 A 想要攻击单位 B，这并不是一个简单的两点最短路问题，A 可以到达能攻击到 B 的圆内的任意一点，这需要我们对传统的最短路算法进行改进来求解。本章将介绍如何对 A*算法进行改进，使其更适合求解 MOBA 地图最短路问题。

3.1 问题描述

在 MOBA 游戏中，有一些单位如小兵、野怪和按照默认逻辑攻击、移动的英雄单位，这些单位在攻击目标时，首先沿着最短路到达能攻击到目标的点，也就是目标为圆心，攻击者攻击距离为半径的圆上任意一点，而后发起攻击。我们定义 MOBA 游戏地图是一张无向图 G(V, E)和障碍点集合 Z，其中 V 是不包含地形之外的网格，E 是从当前网格可以到达的网格。首先按照英雄的移动速度建图，设游戏每秒帧数是 F 英雄移动速度是 S，则英雄 A 在一帧内可以到达的节点 v 满足 $(v_x - A_x)^2 + (v_y - A_y)^2 <= (S/F)^2$，v ∈ V,且 Z 中任意元素 z 都不在 v 和 A 的

连线上。其中 v_x，v_y是 v 的 x 坐标和 y 坐标。E 是每个点向其单帧数时间内可达点的连边集，边权是两点间的距离。在假设移动单位体积近似为点的前提下，移动问题变成了多个终点的单源最短路问题。

3.2 MOBA 最短路算法

直接使用 A*算法求解是可以达到目标的，设发起攻击单位坐标位 S，结束坐标为 T，图为 G(V,E),V 代表节点集合，E 代表边的集合，AR 代表发起攻击单位的攻击距离，则伪代码如下:

Algorithm 2 Original Shortest Path(S,T,G(V,E),AR)

```
EndVertex := all points v with distance(v, T) = AR
currentShortestLength := INF
currentRoad := NULL
for every endpoint in EndVertex:
    road := AStar(S, endpoint, G)
    if length of road < currShortestLength:
        currShortestLength = length of road
        currRoad = road
return road
```

多终点最短路会消耗大量运算时间，尤其是当攻击单位攻击距离很长的时候，可以到达的终点非常多。而考虑反向求解的过程。对于攻击者 A 和被攻击者 B，首先求出 AStar(B,A)，而后在最短路上减掉 A 的攻击距离的长度就是 A 要攻击 B 需要走的路径。但是这样做是不完全正确的，在 MOBA 游戏中，只要有视野，子弹和技能都可以穿过地形。前面提到的对于 B，可以到达的节点 v 是 V_A/F，F 是每秒的帧数，其中 v 和 B 之间不存在地形。而在现在的问题中，我们最后一步的子弹可以穿墙，而且使用的是攻击距离，所以应该移除不存在地形的限制，同时将第一帧间移动的距离改为单位 A 的攻击距离。即满足$(v_x - B_x)^2 + (v_y - B_y)^2 \leq (AR_A)^2, v \in V$，$AR_A$表示单位 A 的攻击距离，其他点连边不变。

所以最终算法伪代码如下（S 代表起点，T 代表终点，AR 代表移动单位的攻击距离）：

Algorithm 3 MOBA Shortest Path(S,T,G(V,E),AR)

E_{origin} := G(E)

remove edges from G whose start point is T

add edges between T to all points v whose distance (v, T) = AR, set edge length = 0

road := Astar(T, S, G)

remove T from road

G(E) = E_{origin}

return road

3.3 算法正确性

算法的正确性通过证明这种算法考虑了所有的可能成为最短路的路径并证明得到的路径是这些路径中最短的路径。首先证明其包含了所有可能成为最短路径的路径，假设存在最短路径$P(E)$，其终点不在位于 B 为圆心，AR 为半径的圆 O 上，由于 MOBA 地图的连续性，最短路径$P(E)$一定与 O 至少存在一个交点V',则以V'为终点，其他点顺序不变的路径$P'(E)$是$P(E)$的子集，即存在$P'(E)$是合法路径且比 P(E)更短，即 P(E)不是最短路，假设错误。所以这个算法考虑到了所有可能是最短路的路径，在这些路径下的最短路径一定是正确的最短路。最后证明求得的路径是最短的路径，参考 A*算法最优性的证明[8]，可得$f(n)$沿任何路径的单调不减性，算法求得的最短路径一定是最短路的一条。

3.4 优化效果

通过这个算法，我们的 MOBA 游戏允许攻击单位接近目标，在目标出现在攻击距离内的时候进行攻击。通过比较容易发现，算法的效率比直接使用 A*寻找优化了$2\pi A$的常数系数（重新建图时间忽略不计），其中 A 是英雄的攻击距离。

算法的实际效果是在 OpenAI Gym v0.10.5 中计算两个寻路场景的时间间隔得到的，在测试场景中，被测单位需要攻击 10 个目标点（先移动到可以攻击目标然后攻击），被测单位速度是 1500（MOBA 地图的长度是 1000，因为优化的是

复杂度的系数，所以要尽量去除常数的影响，速度设置为 1500 可以忽略单位在路程中消耗的时间，从而只需要关心运行时间的比值是否接近理论值），设置五个攻击距离:10,20,30,40,50，每个攻击距离运行十次测试，在每次测试中，对照组是实现了原始 A*算法的组，实验组是实现了 MOBA 最短路算法的组，十次结果取平均值，算法测试平台是 MOBA-Env，结果如下：

攻击距离	10	20	30	40	50
MOBA shortest Path(ms)	89.82	175.60	269.92	353.61	431.50
Original Shortest Path(ms)	5641.59	21950.02	50531.73	88154.96	134412.20
比值 (Original/MOBA)	62.81	125.00	187.21	249.30	311.51
理论比值 (Original/MOBA)	62.83	125.67	188.50	251.33	314.16

实际比值总是小于理论比值，这是由于 MOBA 最短路算法需要重新建图导致的，MOBA 最短路算法在 MOBA-Env 的实际表现符合理论预期。

4. MOBA 游戏场景和对应奖励函数设计

多场景训练对于验证 MOBA AI 的训练算法（PPO 算法）来说是必须的，在本文的实验环节，我设计了简单的寻路场景、放风筝场景以及不同难度的轮流抗血场景来验证 PPO 算法的效果。所有场景都在基于 ubuntu16.04 操作系统的 OpenAI Gym v0.10.5 中运行，使用的 python 版本是 python3.7，Tensorflow 版本是 1.14.0。服务器 CPU: 8,MEMORY: 10000Mi,训练的部分重要参数如下：

```
LEARNING_RATE: 0.0008
BATCH_SIZE = 64 * 8
TIMESTEPS_PER_ACTOR_BATCH = 2048 * BATCH_SCALE
EPSILON = 0.2
```

PPO 算法的实现是基于 OpenAI 的 baseline[4]实现的。

4.1 寻路场景

寻路场景是用来验证 AI 能否找到没有障碍的到达终点的一条路径的场景。这个策略是最简单的 MOBA 策略之一，但因为衡量标准只有时间一项，所以可以通过这个策略验证 PPO 算法是否可以训练出尽快完成任务的策略。

4.1.1 场景参数

	血量	攻击力	攻击范围	移动速度
敌方英雄 1	1000	1000	40	0
我方英雄 1	2000	1000	80	40

设计一个不会动的敌方目标当“靶子”，移动到可以攻击到靶子的距离就可以获得胜利。

4.1.2 奖励函数设计

寻路场景奖励函数(我方英雄 1 为 state 中的 self_hero0,敌方英雄为 state 中的 oppo_hero0)：

Reward function 1:

```
function get_reward(curr_state, last_state)
    distance_last := distance between self_hero0 and oppo_hero0 at last_state
    distance_curr := distance between self_hero0 and oppo_hero0 at curr_state
    positive_reward := (distance_curr - distance_last) * 0.001
    harm_reward := (time from start_time) * 0.005
return positive_reward - harm_reward
```

实际场景构建的图两点之间最短路径长度为 1000，理论需要 23 步进入攻击距离一步进行攻击，所以 reward 最大为 950 * 0.001 - （23 + 1）* 0.005 = 0.83。

4.1.3 实验结果

timesteps	Reward(平均)	Reward(最大)
500	0.3922	0.525
1000	0.7114	0.815
1500	0.8203	0.830
2000	0.8295	0.830

500 代时，平均比最优解多走了 88 步,效率相当于最优解的 21%，1000 代比最优解多走了 24 步，效率相当于最优解的 50%，而 1500 代之后，平均只比最优解多走了 2 步，效率相当于最优解的 93%，2000 代时，平均比最优解多走 0 步，已经无限趋近最优解。所以 PPO 算法训练的 AI 可以学会尽快完成任务的策略。

4.2 放风筝场景

放风筝策略是指在 MOBA 游戏中，移动速度高的一方(或者攻击距离远的一放)，通过交替进行移动和攻击动作，以消耗最少的血量为目的来击杀移动速度慢的一方。

放风筝策略是 MOBA 策略中较为简单的策略，而对于这种简单的策略，很容易在 1000 代以内就随机出比较优秀的解（可能不是最优解），而如果策略更新过快，那么可能陷入这种局部最优解中，最终策略无法到达最优。我想通过对放风筝场景的测试，验证 PPO 算法是否会在简单任务（容易随机到好策略所以步长比较大）下陷入局部最优解。

4.2.1 场景参数

	血量	攻击力	攻击范围	移动速度
敌方英雄 1	3000	100	40	40
我方英雄 1	1000	100	80	60

场景参数上可以明显看出我方英雄需要利用移动速度和攻击距离的优势来获取胜利。

4.2.2 奖励函数设计

放风筝场景奖励函数设计)：

Reward function 2:

```
function get_reward(state)
   positive_reward :=  (oppo_hero0 max health  -  oppo_hero0 current health) *0.01
   harm_reward := (self_hero0 max health  -  self_hero0 current health)*0.03
return positive_reward  -  harm_reward
```

reward > 0 即表示我方目前血量 > 敌方目前血量，而游戏终止，表明敌方血量 = 0，故 reward > 0 在这个场景下即可获胜。而最佳获胜状态是依靠走位优势不掉血，也就是 reward = 3。

4.2.3 实验结果

Timesteps	Reward(平均)	Reward(最大)
500	-1.5625	0.0300
1000	0.0522	2.1000
1500	1.2034	2.7000
2000	1.9021	3.0000
2500	2.8066	3.0000
3000	2.9690	3.0000

reward > 0 即获胜，算法在 500 代就有了可以获胜的策略，平均 reward > 0 则是 1000 代之后，而我们说的最佳获胜状态，也就是一滴血不掉，是在 1500-2000 代之间出现的，而 3000 代，结果已经收敛，从平均结果来看，平均 harm reward=0.0310，也就是在一百次取样中只有一次被击中了一次，可以认为 AI 没有陷入局部最优解，学会了出色的放风筝策略。

4.3 轮流抗血场景(易)

轮流抗血场景是 MOBA 游戏中的重要场景，当面对 BOSS（比如英雄联盟的大龙）时，英雄间需要合理的团队策略才能取胜，在简单场景下，我们只要求两个英雄间有一个在承受伤害即可。该场景有两个我方英雄，目的是验证 PPO 算法能否能训练出用于多智能体的策略。

4.3.1 场景参数

	血量	攻击力	攻击距离	移动速度
敌方英雄 1	4500	100	40	50
我方英雄 1	3000	10	40	50
我方英雄 2	300	150	40	50

因为敌方英雄的攻击是优先攻击距离最近的目标，所以初始化英雄位置不可以随机，而是应该令我方英雄 1 和我方英雄 2 到敌方英雄的距离相等，而在这个场景中，我方英雄 1 只要一直被攻击，且我方英雄 1 和我方英雄 2 可以一直在这个时间内攻击敌方英雄 1 即可取胜。

4.3.2 奖励函数设计

轮流抗血(易)场景奖励函数设计(self_hero0 为我方英雄 1，self_hero1 为我方英雄 2)：

Reward function 3:

```
function get_reward(state)
    harm_reward := 0
    if self_hero0 health is 0:
        harm_reward = harm_reward + 0.5
    end if
    if self_hero1 health is 0:
        harm_reward = harm_reward + 1
    end if
    harm_reward = harm_reward + (self_hero0 max health  -  self_hero0
    current health)*0.001
    harm_reward = harm_reward + (self_hero1 max health  -  self_hero1
    current health)*0.005
    positive_reward := oppo_hero0 max health  -  oppo_hero0 current
    health) *0.001
return positive_reward  -  harm_reward
```

最好情况下，开局我方英雄 1 先前进一步，剩下的时间通过合作继续 28 步打死敌方英雄 1，这种情况下我方英雄 1 被攻击了 29 次，reward = 4.5 - 2.9 = 1.6。而场景设计允许我方英雄 1 死亡后我方英雄 2 继续攻击 2 次，也就是获胜策略有三次机会偏离最优动作。

4.3.3 实验结果

Timesteps	Reward（平均）	Reward(最大)
500	-1.6039	-1.050
1000	-0.2351	0.560
2000	0.7623	1.020
3000	1.0003	1.500
4000	1.2952	1.600

4999	1.4037	1.600

平均 reward ＞ 1.1，表示两个英雄在大多数情况下都存活，也就是简单的轮流抗血在 4000 代左右训练出的策略到达获胜目标。证明 PPO 算法训练的策略可以用于多智能体。

4.4 轮流抗血场景(难)

在实际游戏中，为了避免使对方击杀我方英雄获得金钱和经验，输出可以适当的承受一部分伤害，也就是该场景下，我方两个英雄需要都存活。这个场景的困难之处在于数据设计和训练，数据设计方面，考虑到真实的 MOBA 环境，输出英雄承受伤害的能力不会很高，所以两个英雄都不可以有冗余操作，并且承受伤害英雄转身时间最好是“恰好”无法承受下一次伤害，否则会导致 reward 小于理论最优解。

4.4.1 场景参数

	血量	攻击力	攻击距离	移动速度
敌方英雄 1	4000	200	40	50
我方英雄 1	4000	50	40	50
我方英雄 2	1000	150	40	50

4.4.2 奖励函数设计

轮流抗血(难)场景奖励函数设计(self_hero0 为我方英雄 1，self_hero1 为我方英雄 2)：

Reward function 4:

```
function get_reward(state)
    harm_reward := 0
    if self_hero0 health is 0:
        harm_reward = harm_reward + 1
    end if
    if self_hero1 health is 0:
        harm_reward = harm_reward + 1
    end if
    harm_reward = harm_reward + (self_hero0 max health - self_hero0
    current health)*0.001
```

```
    harm_reward = harm_reward + (self_hero1 max health - self_hero1
    current health)*0.005
    positive_reward := oppo_hero0 max health - oppo_hero0 current
    health) *0.001
return positive_reward - harm_reward
```

该场景的最佳 reward = 4.5 - 3.8 - 0.2 = 0.5，但是由于场景太难，我最终的训练结果也没有任何一种策略的 reward 到了 0.5。

4.4.3 实验结果

Timesteps	Reward(平均)	Reward（最大）
1000	-2.378	-2.010
2000	-1.703	-1.330
3000	-1.135	-0.860
4000	-0.853	-0.310
5000	-0.628	-0.109
6000	-0.409	-0.030
7000	-0.053	0.150
8000	-0.052	0.150

6000 代之后大多数情况下两个我方英雄都可以存活，但是却无法到达理论最优解,猜想是因为我方英雄 1 只有一个时间转身才能达到最优解(剩余 400 血，转身再被打一下，剩余 200 血)，其他情况下，转身太晚会导致死亡，太早会使输出英雄承受更多伤害，场景设计允许输出英雄多承受三次伤害，显然大部分策略我方英雄 1 转身过早。在太难随机到最优解的情况，PPO 算法可能会在一个比较优秀的解收敛。

5. 总结

本毕业设计完成了 MOBA 环境的完善和 PPO 算法在不同场景下的效果验证，在原有 MOBA 环境的基础上增加了地形设计，并基于地形设计了 MOBA 最短路算法，通过 MOBA 游戏的需求降低了最短路算法在 MOBA 地图上的复杂度常数系数，并实现了原始算法，测试了实际优化效果。根据 MOBA 游戏的常用策略设计了风筝、寻路和不同难度的轮流抗血场景，定义了获胜条件和奖励函数。学习了 PPO 算法并使用 PPO 算法在设计的场景下训练 AI 的决策，观察 AI 学会可获胜策略的时间并比较最终收敛的策略奖励值和设计场景时计算的最佳状态的理论值。通过

寻路场景，我验证了 PPO 算法可以用于训练与时间有关的策略；通过放风筝场景，我验证了 PPO 算法在可能导致步长过大的简单任务下不会陷入局部最优解；通过简单轮流抗血场景，我验证了 PPO 算法可以用于多智能体的训练(团队决策)；通过复杂轮流抗血场景的收敛情况，我得出了 PPO 算法在很难随机到最佳策略时仍然会陷入局部最优解。

参考文献

[1] G.Brockman and V.Cheung, “Openai five defeats dota 2 world champions,” https://openai.com/blog/openai-five-defeats-dota-2-world-champions/, April 15, 2019.

[2] Ye, D., Liu, Z., Sun, M., Shi, B., Zhao, P., Wu, H., ... and Chen, Q. (2019). Mastering Complex Control in MOBA Games with Deep Reinforcement Learning. arXiv preprint arXiv:1912.09729.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” 2016.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms. arxiv 2017,” arXiv preprint arXiv:1707.06347.

[5] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, “Emergence of locomotion behaviours in rich environments,” arXiv preprint arXiv:1707.02286, 2017.

[6] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in International conference on machine learning, 2015, pp. 1889－1897.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics. pp. 100－107.

[9] S. Kakade and J. Langford. “Approximately optimal approximate reinforcement learning”. In: ICML. Vol. 2. 2002, pp. 267–274.

致谢

感谢我的导师 Dr.刘佳琳对本次毕业设计的全方面指导，感谢腾讯 IEG 光子学习与发展组提供测试平台、训练环境还有理论指导。