



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

本科生毕业设计（论文）

题 目： 完善 MOBA 游戏环境并在该环境作强化
 学习算法研究

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

完善 MOBA 环境并在该环境作 强化学习算法研究

赵 宇

南方科技大学计算机科学与工程系 指导教师：刘佳琳

腾讯 IEG 光子学习发展组

[摘要]：多人在线战术竞技游戏（Multiplayer online battle arena, 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，从而在游戏市场上表现出色。本毕业设计完善了一个基于 OpenAI Gym 的 MOBA 环境，在 MOBA 环境中加入了地形的定义，对其中的单位设计了 MOBA 最短路算法来控制其移动攻击。学习了 MOBA AI 训练的常用算法 PPO 算法，根据实际 MOBA 游戏中的经典策略设计了放风筝、寻路、和不同难度的轮流抗血场景，对于不同场景，定义了场景的获胜条件和奖励函数，通过 PPO 算法训练 MOBA AI 的策略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) games are a type of games that most game enthusiasts are very interested in. Generally, MOBA games have game strategies for players to explore and diversified game scenarios. These characteristics make MOBA games highly gameable, and thus play a role in the game market. Excellent performance. This graduation design improves a MOBA environment based on OpenAI Gym. The definition of terrain is added to the MOBA environment. The MOBA shortest path algorithm is designed for the units to control its mobile attacks. Learned the common algorithm PPO algorithm for MOBA AI training, and designed the anti blood scenes of kite flying, pathfinding, and different difficulty according to the classic strategy in actual MOBA games. Train the strategy of MOBA AI through PPO algorithm and compare the theoretical reward with the actual reward to analyze the performance of PPO algorithm in different difficulty scenarios.

[Keywords]: MOBA, Reinforcement learning, Game AI, Multi-scenario.

目录

1. 引言	1
1.1 MOBA 游戏介绍	1
1.2 基于 MOBA 游戏的 AI	1
1.3 多场景训练 MOBA AI 的重要性	2
1.4 PPO 算法简介	2
1.5 研究动机	2
1.6 主要贡献和成果	3
2. 背景介绍	3
2.1 MOBA 游戏	3
2.2 MOBA AI 训练环境	3
2.2.1 DOTA2	3
2.2.2 OpenAI Gym	4
2.2.3 环境选择	4
2.3 PPO 算法	4
2.3.1 PPO-Penalty	4
2.3.2 PPO-Clipped	4
2.4 单源最短路 A*算法	5
3. MOBA 最短路算法	6
3.1 问题描述	6
3.2 MOBA 最短路算法	6
3.3 算法正确性	7

3.4 优化效果	7
4. MOBA 游戏场景和对应奖励函数设计	7
4.1 风筝场景	7
4.1.1 场景参数	7
4.1.2 奖励函数设计	8
4.1.3 实验结果	8
4.2 寻路场景	8
4.2.1 场景参数	8
4.2.2 奖励函数设计	9
4.2.3 实验结果	9
4.3 轮流抗血场景（易）	9
4.3.1 场景参数	9
4.3.2 奖励函数设计	9
4.3.3 实验结果	10
4.4 轮流抗血场景（难）	10
4.4.1 场景参数	10
4.3.3 奖励函数设计	10
4.3.4 实验结果	11
5 总结	11
参考文献	12
致谢	13



1. 引言

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。目前的 MOBA AI 已经可以击败世界冠军队伍，幸运的是，这个强化学习算法是公开的，本文将完善一个 MOBA 环境，并基于这个 MOBA 环境做该强化学习算法在不同场景下的效果验证。

1.1 MOBA 游戏介绍

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。

1.2 基于 MOBA 游戏的 AI

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 使用的算法是 PPO 算法，我们也选用了 PPO 算法作为我们的训练算法，这一算法将在后面进行介绍。

腾讯的“绝悟”AI^[2]也在 1v1 王者荣耀单挑比赛中打赢了顶级职业选手，但是腾讯 AI Lab 并未公开使用的算法，故不在此作更多介绍。

1.3 多场景训练 MOBA AI 的重要性

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，比如两个英雄至少有一个要死，我们更希望辅助死掉而不是输出死掉。这需要合理设计奖励函数，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。

1.4 PPO 算法^[4]简介

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。

策略梯度方法是一种不需要估算状态-动作值函数来推断最优策略，而是直接对策略进行优化的方法，但是这种方法在调节步长上有些不足，如果步长太小，那么更新速度会很慢，训练简单任务也许会消耗大量时间。而如果步长太大，噪声等因素的影响可能使得结果越学越差或者导致策略不会收敛。

TRPO^[6]算法可以在更新策略的同时保证奖励函数不减少，但是 TRPO 的数学推导过于复杂，与其他算法的兼容性也不好。

PPO 算法很好的平衡了实现难度和算法效率，通过创新的 clip 设计保证了步长总是在合理区间内。它采取了 TRPO 的长处，并将算法变得更加易于实现、普适性更强。目前 PPO 算法已经是 OpenAI 的强化学习的默认算法，OpenAI Five 也是使用 PPO 算法进行训练的。

1.5 研究动机

我在 2019 年看到 OpenAI 打赢世界冠军 OG 之后，和我的几个感兴趣的朋友组队挑战 OpenAI，我们的 DOTA2 天梯排名都在前 1000，但是面对这些从对线开始就很凶残的 AI，我们输得非常惨烈。从那之后我对 MOBA AI，尤其是 OpenAI five，产生了很大的兴趣。后来我也得到了自己训练 AI 的机会，通过自己设计的场景，对 OpenAI 训练 AI 的算法进行验证：在一些场景下（比如轮流抗血），AI 能否表现得像人类一样出色。

1.6 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。通过这个算法我将直接正向使用 A*算法求解优化了 $2\pi A$ 的常数，其中 A 是英雄的攻击距离。

- 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



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，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 TRPO 算法那样复杂。PPO 算法有两种实现，一种是 PPO-Penalty, 另一种是 PPO-clipped。

2.3.1 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 $\hat{E}_t \left[\frac{\pi_{\theta}(\alpha_t | s_t)}{\pi_{\theta_{old}}(\alpha_t | s_t)} \hat{A}_t - \beta KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)] \right]$ ，其中 θ 是策略参数， π 是策略概率， β 表示自适应罚分系数， s_t 代表状态， α_t 代表行为， \hat{A}_t 是时间 t 的估计优势。通过计算 $d = \hat{E}_t [KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)]]$ ，更新自适应罚分系数 β 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（论文中 $\beta/2$ ），当 d 大与一个设定值，表示策略更新太快，此时加大 β ($\beta*2$)， 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。PPO-Penalty 更像 TRPO，在具体实验中，效果不如 PPO-Clipped 优秀。

2.3.2 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 $clip(\frac{\pi_{\theta}(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)}, 1 - \epsilon, 1 + \epsilon)$ ， 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) + ETP
    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 $(v_x - A_x)^2 + (v_y - A_y)^2 \leq (S/F)^2$ ， $v \in V$ 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，这种算法的伪代码如下：

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

EndVertex := all points v with $\text{distance}(v, T) = A$

currentShortestLength := INF

currentRoad := NULL

for every endpoint in EndVertex:

 road := AStar($S, \text{endpoint}, G$)

 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 (因为即使是逆向求解，最终移动的还是 A，所以取 A 的速度)，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 $(v_x - B_x)^2 + (v_y - B_y)^2 \leq (\text{AttackRange}_A)^2, v \in V$ ，都是图中和 B 相连的点，其他点的连边规则不变。

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

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

$E_{\text{origin}} := G(E)$

```
remove edges from G whose start point is T
add edges between T to all points v whose distance (v, T) = A, set edge
length = 0
road := AStar(T, S, G)
remove T from road
G(E) = Eorigin
return road
```

3.3 算法正确性

对于这个算法的正确性，只需要证明任意最短路终点一定位于终点为圆心，A 为半径的圆上即可，因为 MOBA 最短路算法终点向这些点各连了一条边权为 0 的边。任意最短路终点不可能从圆内，因为 MOBA 地图是一张类网格图，必须连续移动，圆内到圆上还有一定距离，所以以圆上点为最短路终点得到的解不会比以圆内为终点差（因为移动速度问题，所以两个解消耗的时间可能相等，即单位在一帧的时间内即可以跑到圆上也可以跑入圆内）。任意最短路起点不可能从圆外，因为不符合攻击距离的需求，不是合法的最短路。所以算法寻找的最短路一定在最短路解的集合里。

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^[7]，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 放风筝场景

放风筝策略是指在 MOBA 游戏中，移动速度快的一方（或者攻击距离远的一方），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。放风筝场景的设计目标是训练 AI 学会使用放风筝策略。

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	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。

4.1.2 奖励函数设计

这一任务比较简单，在不考虑获胜时间的前提下，可以记我方英雄 1 对敌方英雄 1 造成的伤害的回报 $\text{positive reward} = (\text{敌方最大血量} - \text{敌方目前血量}) * 0.001$ ，而我們希望自己被攻击的尽量少， $\text{harm reward} = (\text{我方最大血量} - \text{我方目前血量}) * 0.003$ ， $\text{reward} = \text{positive reward} - \text{harm reward}$ ，在这个场景下， $\text{reward} > 0$ 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 $\text{reward} > 0$ 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 $\text{reward} = 3$ 。

4.1.3 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
3000	2.8066	3.0000

$\text{reward} > 0$ 即获胜，算法在 500 代就有了可以获胜的策略，平均 $\text{reward} > 0$ 则是 1000 代之后，而我們说的最佳获胜状态，也就是一滴血不掉，是在 1500-2000 代之间出现的，而 3000 代，平均 reward 计算可得我方英雄掉血平均不足 100，这可能是出生地点随机导致的，可以认为到了 3000 代，AI 已经学会了出色的放风筝策略。

4.2 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。

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	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。

4.2.2 奖励函数设计

考虑使用时间作为 harm reward, 游戏每运行一帧, hard reward 增加 0.005, 这个设计也可以避免 AI 原地不动, 预处理出终点到所有点的最短路长度, $positive\ reward = (上一帧距离 - 当前帧距离) * 0.001$, 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 $950 * 0.001 - (23 + 1) * 0.005 = 0.83$ 。

4.2.3 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。

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。

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

定义我方英雄 1 死亡, harm reward 增加 0.5, 我方英雄 2 死亡, harm reward 增加 1, 其余情况下 $hard\ reward = (我方英雄 1\ 最大生命值 - 我方英雄 1\ 当$

前生命值) * 0.001 + (我方英雄 2 最大生命值 - 我方英雄 2 当前生命值) * 0.005, 即输出英雄掉血惩罚更高。Positive reward = (敌方英雄 1 最大生命值 - 敌方英雄 1 当前生命值) * 0.001, 最好情况下, 开局我方英雄 1 先进一步, 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。

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, 也就是该场景下输出英雄和承受伤害英雄死亡都算没有达到场景目标, 理想情况下, 承受伤害英雄剩下的血量只足以承受两次攻击后就选择后退, 可以保证都存活且输出英雄掉血最少。场景难度如果太难(不能有任何一步多余), 就难以随机到正确结果, 很容易上万代中也没有达到最优解的策略。所以场景设计时仍然允许有冗余操作。

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

定义任何我方英雄死亡, harm reward 增加 1(目标是所有我方英雄都不死), 其余情况下 hard reward = (我方英雄 1 最大生命值 - 我方英雄 1 当前生命值) * 0.001 + (我方英雄 2 最大生命值 - 我方英雄 2 当前生命值) * 0.005, 即

输出英雄掉血惩罚更高。该场景的最佳 $\text{reward} = 4.5 - 3.8 - 0.2 = 0.5$ ，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。

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。

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.

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。

