StarCraftII强化学习实验报告

贺子凡 1800013017
李典泽 1800012990
郑钦源 1800013003

目录

1. 实验介绍

实验环境

本实验基于deepmind开发的pysc2环境进行。pysc2是一个StarCraftII的学习环境，提供了用于机器学习的API，定义了各种环境参量和动作种类，可以通过调用其中的库函数实现与StarCraftII环境的交互以及得到反馈。deepmind/pysc2: StarCraft II Learning Environment (github.com)是关于pysc2的详细介绍。

本实验总共在7个minigames上进行了实验，分别为“MoveToBeacon"，"CollectMineralShards"，"FindAndDefeatZerglings"，"DefeatRoaches"，"DefeatZerglingsAndBanelings"，"CollectMineralsAndGas"以及"BuildMarines"。七个minigames将全局游戏分为更小单元，对agent在收集资源，简单战斗等特定方面分别进行测试。关于minigames的详细介绍请见第二部分。

本实验中主要使用了pysc2中与环境交互得到下一步observation的部分。得到的observation主要有四个属性：screen、minimap、player以及available actions，其中前三个作为模型网络的输入，而第四个则对选择动作有至关重要的作用。action在pysc2中被定义为函数，对应不同的function_id；其参数就是完成该动作需要的信息，对应type_id。

运行环境：

- pysc2 1.7.2
- pytorch 1.7.0
- ubuntu 18.04 or windows 10

分工情况

贺子凡同学主要负责A3C和PPO框架的搭建及模型7的训练

李典泽同学主要负责训练模型1、2、4及rule-based agent2、4、5代码

郑钦源同学主要负责训练模型3、5、6及rule-based agent6、7代码

代码介绍

实现了

- 用PPO，A2C两种算法进行强化学习训练以及相应的测试模型方法。
- 6张地图的Rule base算法以及用相应的rule base算法进行预训练。
- 搭建了多线程环境模型。
- 实现了两种神经网络（Atari net和Full net）进行训练，两种observation的预处理方法。

Rule Base：

运行我们实现的rule base算法：

```
$ python run_RuleBase.py --map <map_name>
```

对利用rule base对模型进行预训练：

```
$ python pretrain.py --map <map_name>
```

从效果上来看，预训练只有'CollectMineralShards'和'DefeatRoaches'有显著的提升

Reinforcement learning train：

- 对大部分地图，运行以下代码可以训练

```
$ python main.py --map <map_name>
```

- 需要改变其它参数时，具体请查看main.py

```
$ python main.py --map <map_name> --algorithm <a2c or ppo, default:a2c> --
load_model <pretrained model> --envs <num of env> --save_dir <root directory for
checkpoint storage>
```

由于训练地图'CollectMineralsAndGas'时，由于会出现迅速梯度爆炸的情况，所以需要对observation中的screen和minimap的较大的标量进行预处理，详细处理过程在build_input.py中，通过增加维数来解决梯度爆炸问题，其它地图也可以使用这个方法，但是训练速度和测试速度会慢很多。

- 将process_screen设置为True

```
$ python main.py --map CollectMineralsAndGas --algorithm <a2c or ppo,
default:a2c> --process_screen
```

Reinforcement learning test：

- 对于test，提供了两种方法，determined和sample，就是选择动作时是取argmax还是直接从probility中sample。一般来说，使用sample的方法就可以：

```
$ python test.py --map <map_name> --load_model <model to test>
```

- 对于地图'CollectMineralShards'，我们提供的模型用determined选取动作效果更好

```
$ python test.py --map CollectMineralShards --load_model
./save/CollectMineralShards_model.pkl --determined
```

- 对于地图'CollectMineralsAndGas'，由于训练加了observation预处理，所以test时需要加上预处理参数，同样，训练时有预处理参数，那么test都需要加上这个参数。

```
$ python test.py --map CollectMineralsAndGas  --load_model
./save/CollectMineralsAndGas_model.pkl --process_screen
```

实验方法

本实验参考deepmind论文《StarCraft II: A New Challenge for Reinforcement Learning》中提出的实验方法，对其中的Atari-net, FullyConv架构进行了复现和比较，并分别采用A3C，PPO两种方法进行训练。除此之外，我们还尝试了基于rule-based的模仿学习(imitation learning)方法进行预学习，在一些minigames上取得了比较好的结果。

预处理

首先对得到的observation进行预处理，具体包括：

1. 将observation中的screen、minimap、non_spatial数据进行处理，将screen和minimap中含有多个类别信息通道的feature layer分开成多个只含一个类别信息通道的feature layer(此时该layer为one-hot)，再对screen和minimap中数据信息通道的feature layer以及non_spatial进行对数变换防止出现过大值
2. 将一个batch的observation的screen、minimap、non_spatial和available actions分别连接起来

网络结构

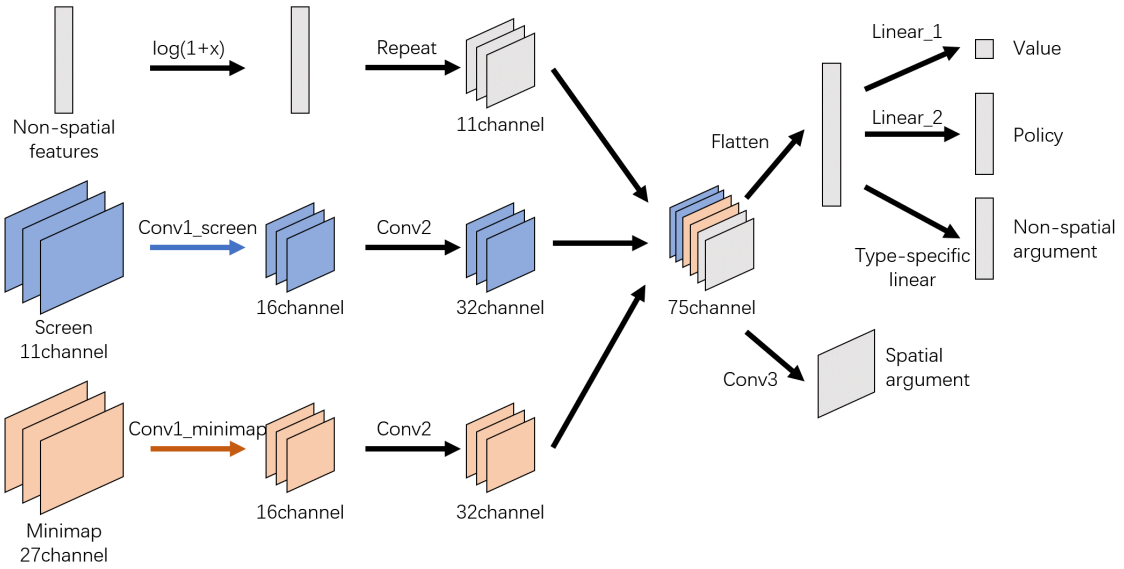
本实验采用Actor-Critic架构。Actor-Critic架构通过Actor网络输出当前状态下的policy，再通过Critic网络对该策略进行评估；Critic和Actor都基于TD(λ) 进行更新。不同于一般Acor-Critic，本实验通过一个网络同时输出actor和critic，实现部分参数共享，减少计算量，加快收敛。网络架构参考论文中采用的FullyConv网络，输入为多个rgb图像(screen, minimap)以及一些非空间(non_spatial)信息，输出为当前状态下的policy和value。网络分为以下几层：

1、输入图像特征提取及non_spatial信息的复制。用两个不同的卷积层大小都是filters=16，kernel_size=3*3的卷积层分别对screen和minimap进行卷积，再通过一个大小为filters=32，kernel_size=5*5的卷积层，提取screen和minimap的特征。由于non_spatial信息维数较低，因此需要进行复制扩充，从而与处理后的screen和minimap数据形状保持一致。将处理后的screen、minimap和non_spatial数据拼在一起，记为当前的状态state；

2、value部分：让state经过Flatten层降为一维，再通过一个全连接层输出当前预测的value；

3、policy部分：state经过Flatten，再通过一个全连接层(激活函数为softmax)输出function_id，再对每个type进行处理：若该type是空间相关的(screen等)，则将state通过一个卷积层同时softmax；若该type是非空间相关的(queue等)，则通过一个与type相关的全连接层并softmax

网络完整架构见下图。



训练过程

A3C

本实验用动作的advantage代替reward，实现了Advantage Actor-Critic(A2C)。除此之外，本实验还通过加入policy的entropy作为loss的一部分，从而增强agent的探索性。Loss函数完整计算公式如下：

$$Loss = \underbrace{(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{policy\ gradient} + \beta\underbrace{(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t)}_{value\ estimation\ gradient} + \eta\underbrace{\sum_a \pi_\theta(a|s_t)\log \pi_\theta(a|s_t)}_{entropy\ regularisation}$$

$v_\theta(s_t)$为状态评估value；θ为策略参数；β、η为超参数，对应value loss和entropy在Loss函数中占的比重；$G_t = \sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n v_\theta(s_{t+n})$，是我们希望maximize的对象，$G_t - v_\theta(s_t)$即该动作的advantage，

采用Adam优化器进行随机梯度下降。训练采用8个进程同时sample数据训练，实现了Asynchronous Actor-Critic(A3C)。

PPO

本组同样实现了**proximal policy optimization (PPO)**算法，与a3c不同，该算法基于off-policy，首先，我们将新旧策略之间的概率比表示为：

$$r(\theta_0) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

在没有限制θ_{old}和θ_{now}之间的距离的情况下，最大化policy loss会变得很不稳定，ppo强制把$r(\theta)$进行了限制

$$r(\theta) = min(r(\theta_0), clip(r(\theta_0), 1-\epsilon, 1+\epsilon))$$

与a2c类似，PPO有3个loss函数，主要的区别在于policy loss

$$policy\_loss = \underbrace{r(\theta)(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{policy\ gradient}$$

其余两个loss函数不变，所以总的loss函数为：

$$Loss = \underbrace{r(\theta)(G_t - v_\theta(s_t))\nabla_\theta \log \pi_\theta(a_t|s_t)}_{policy\ gradient} + \beta\underbrace{(G_t - v_\theta(s_t))\nabla_\theta v_\theta(s_t)}_{value\ estimation\ gradient} + \eta\underbrace{\sum_a \pi_\theta(a|s_t)\log \pi_\theta(a|s_t)}_{entropy\ regularisation}$$

由于PPO算法每次梯度下降都要经过若干个epoch，所以更新的速度相较于a2c更慢一些。

Supervised Rule-based learning

本实验具有动作空间和状态空间非常大的特点，不仅给了exploration很大的挑战，还会导致模型收敛极慢甚至不收敛。实验证明，若没有足够的算力则上述强化学习的方法很难在短时间内得到较好的结果。因此我们尝试加入人类先验知识对网络进行预训练从而加速模型的训练和收敛过程：

1、mask掉一些available actions。minigames的任务中很多情况下available actions有很多，但该任务只需要很少几个特定动作就能完成。通过mask掉不需要的动作，可以排除其对训练的影响，加快模型的收敛速度；

2、采用基于rule-based agent的模仿学习进行预学习。我们首先实现了除第三个任务的其余rule-based agent，之后让agent去模仿rule-based agent采取的动作，从而在较短时间内获得一定的效果。预学习完成后再将得到的模型通过PPO或A2C进行进一步训练，对于加快收敛速度有很大的帮助。本实验采用两个agent policy的交叉熵和value的均方差的加和作为损失函数。

$$Loss = CrossEntropyLoss(\pi, target) + \alpha\sum(value - value_{target})^2$$

2. Minigame逐个分析

2.1 MoveToBeacon

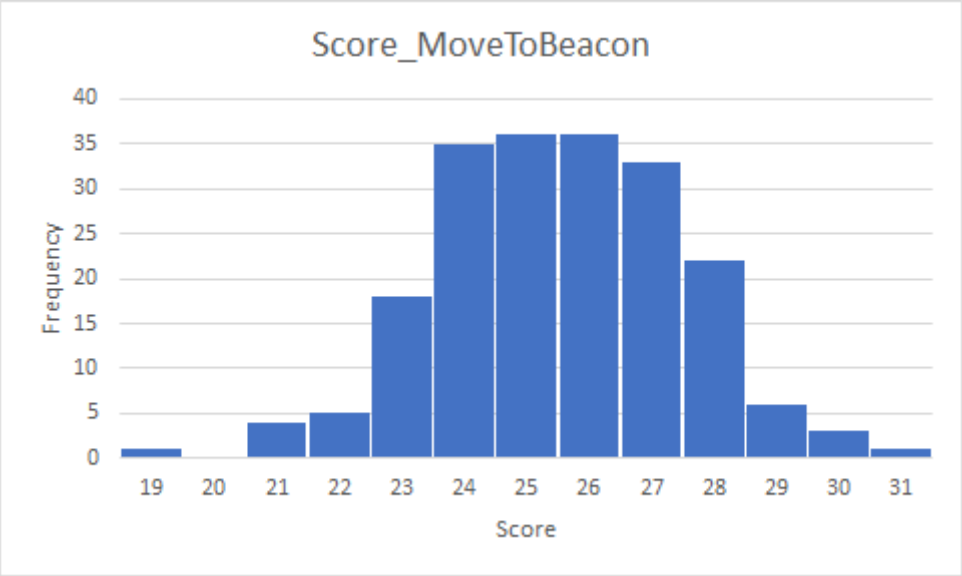
该minigame任务是操纵一个marine到达图中的beacon位置，到达后beacon获得1的reward，同时beacon位置会改变，一次游戏时间(120s)内到达beacon数量越多，得到的分数越高。该minigame中需要的actions为select_army(function_id=7)、move_screen(function_id=331)。该任务动作空间较简单，基本只需一次select_army，之后向正确的位置执行move_screen即可，因此我们未作过多处理，通过原始的A3C方法经过50个episode左右的训练模型便基本收敛。

该实验中有两点尤其需要注意的地方：

1、由于本实验主要任务是找到beacon的位置，因此对screen和minimap进行卷积从而实现特征提取的前几层卷积层非常重要，好的卷积层可以帮助模型迅速收敛，而不好的卷积层会提取不到好的特征，进而大幅影响模型的效果；

2、为惩罚已选定marine后的no_op和select_army操作，我们修改了reward，使其在未得到reward时获得-0.1的reward，实验证明该方法确实能够使agent更少执行这两个动作。

模型在50个episode时涨幅变得非常缓慢，100个episode基本完全收敛；对该模型进行测试，200个episode中max score=31，mean score=25.7。



2.2 CollectMineralShards

该minigame任务是操纵2个marine将图中的25个minerals全部收集起来，收集到1个mineral获得1的reward，而全部收集完后又会刷出25个新的minerals，一次游戏时间(120s)内收集的minerals越多，得到的分数越高。该minigame需要的动作为select_point(function_id=2)(select_army可替代select_point)，以及move_screen。该任务除了需要做到MoveToBeacon的找到目标位置外，还需要agent制定最优的行动顺序。agent需要有规律地采集minerals，这样才可以避免绕远路；同时agent还需要向同一个地方连续的执行move_screen，否则他就会出现左右徘徊而不前进的问题。以上两点使得其相比于第一个任务而言状态空间复杂了很多，大大增加了训练的难度。

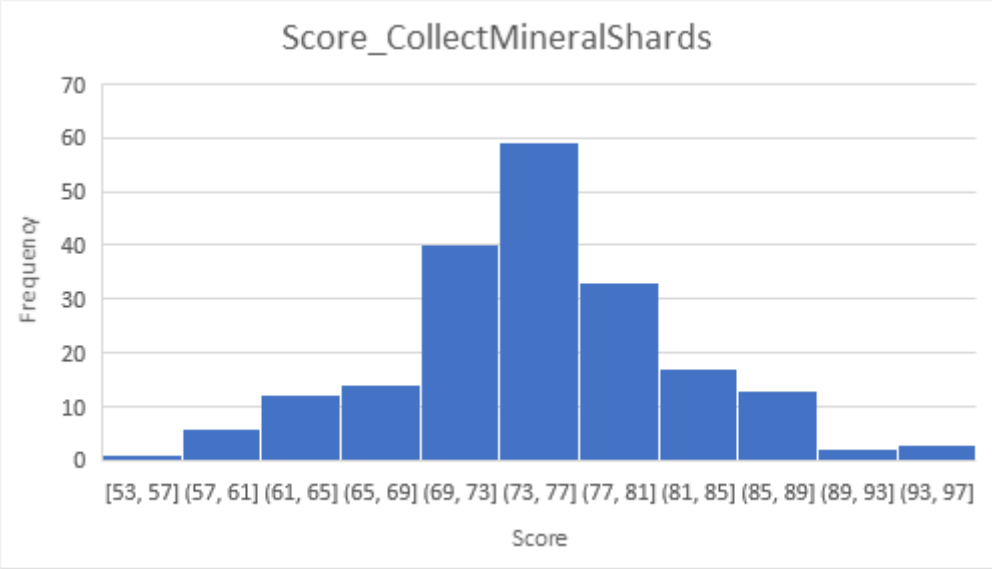
实验过程中我们发现，一些模型可以很精准的向mineral位置执行move_screen，说明其卷积层已经能够进行有效的特征提取，且全连接层得到的policy也有一定效果；但紧接着agent又向另一个位置的mineral执行move_screen，导致agent偏离了一开始选定的mineral，而多个上述情况接连发生则会导致marines左右徘徊而未收集到minerals。为解决上述问题，我们首先在训练的过程中人为的将每次move_screen操作的queued属性都设为1，即所有move_screen都不是立即执行的，而是排队等候队列中前面的动作执行完后再执行。实验结果并不理想，因为虽然marine不会再左右徘徊，但其经常选择一个非常远的mineral采集，而后又选择一个离一开始的位置很近，当前位置很远的mineral进行采集，绕远路的问题并未解决；除此之外，若某一次move_screen向一个错误位置移动，则marine必须移动到那

个位置后才能继续移动，极大浪费了时间；同时，虽然agent能够比较正确的找到mineral的位置并进行移动，但其向同一个位置移动时相邻几次移动的坐标并不是完全一致，而是相距非常近，且这几次move_screen都存储到了队列中，这导致marines在采集到一个mineral后会在附近执行几次距离非常小的move_screen，也对最终分数产生了较大影响。总而言之，仅通过A3C或PPO很难实现该任务的学习。

最终我们采取的方法是首先通过模仿学习进行预学习，再利用PPO进行进一步学习得到最终模型。我们首先采用的rule-based agent采取的策略是同时移动两个marines，向距其最近的mineral移动，这种rule-base agent可以达到mean score=80的水平。预学习50个episode后进行测试，发现上述两个问题得到了有效解决，agent既能够向靠近的mineral移动，同时也不会进行无谓的左右徘徊；之后降低学习率和η(熵系数)，通过PPO进一步训练，200个episode后模型基本稳定。之后我们尝试了更复杂的rule-based agent，即分别操控单个marine收集最近的mineral，这种agent可以达到mean score=113分的水平，但实际模仿学习后我们发现效果并不理想，经常会出现一直执行select_point而从不移动的问题。经过分析，我们认为这与rule-based agent采取的动作有关。第二种rule-based agent以四步作为一个周期，每个周期中执行select_point(marine0)，move_screen(mineral0)，select_point(marine1)，move_screen(mineral1)，这种频繁的select_point操作可能使得agent只学会了模仿select_point，进而导致了上述结果。除此之外，rule-based agent相比于我们的agent获得了更多诸如是否选中marine等信息，这些信息对rule-based agent决策的选择是至关重要的，但我们的agent无法获得这些信息，因此难以进行模仿。基于次我们得出结论：若模仿的对象采取的动作过于复杂，且其得到了额外的信息，则模仿学习很可能难以取得效果。

尽管我们在预学习后通过PPO进行了一定的探索和训练，但我们的agent依然未学会分别操控2个marine。我们认为这其中的原因有两点：一是经过预学习后policy中select_army的概率已经远大于select_point的概率，因此即使有探索性保证，也很难采样到单独操控marine的情况；二是即使采样到了单独操控marine的情况，很多时候在一个trajectory中它得到的分数并不会比一起操控的分数高很多，相反除非采样的动作序列按照select_point和move_screen交替进行，否则其得到的分数可能还会更低，进一步降低了采取select_point的概率。以上两点原因使得该任务的完美策略训练几位困难。

测试预学习后PPO继续训练200episode的模型，得到max score=97，average score=75.64。



2.3 FindAndDefeatZerglings

该minigame任务是操纵三个marine探索被战争迷雾笼罩的未知地图区域，寻找尚未击败的虫族，与之交战并获取胜利。游戏开始之初会在地图中心产生三个已被选择了的marine，并在其视野范围内随机生成两个虫族，视野外随机生成23个虫族，合计25个敌对虫族；当地图上虫族全被消灭后，新的25个虫族会在地图上较为均匀的随机生成。每击败一个虫族获得+1reward，每死亡一名战士获得-1reward，士兵全部死亡或到达时限后重启游戏。

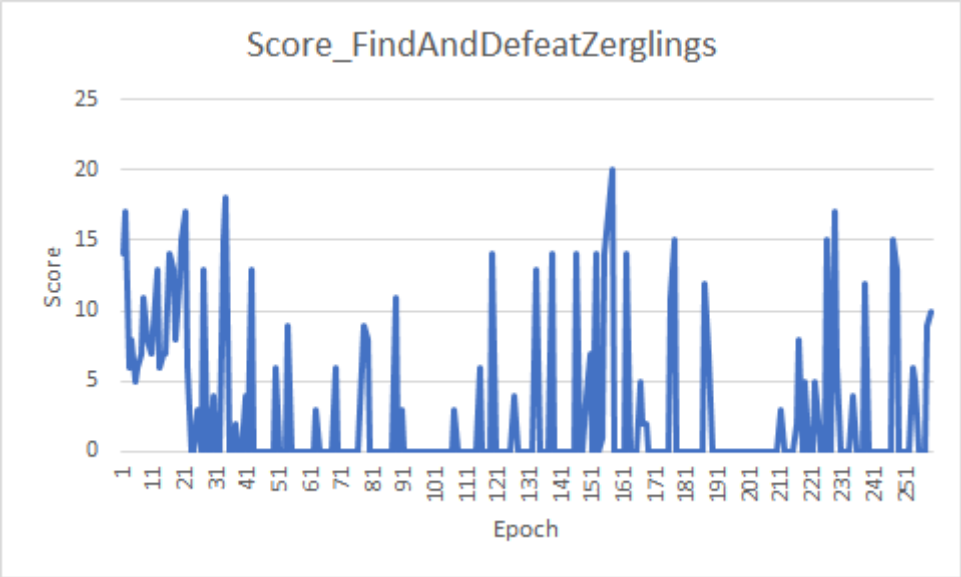
该minigame中需要的actions为move_camera(function_id=1)、attack_screen(function_id=12)。虽然看似动作空间很小，但这个小游戏的逻辑其实一点也不简单：

1. 与其他六个minigame不同的是，FindAndDefeatZerglings是唯一一个有战争迷雾加入的小游戏。战争迷雾掩盖了敌对虫族的存在信息，使得screen和minimap这两种输入中的信息含量受到了极大限制，大幅度加大了agent的学习难度
2. 该minigame还有另外一个独特之处，它是唯一一个需要移动视野来获取信息、下达指令的小游戏。其他小游戏在当前视野范围内就能够获取到所有需要的信息，执行所有需要的功能与任务，但在这个minigame中agent要学会通过移动视野来获取信息、下达指令，这一层隐性的逻辑判断也同样增大了agent的训练复杂度
3. 除此之外，agent还需要能够规划出一条能够满足两种性质的探索/战斗路径：一是高效性，这与第二个小游戏CollectMineralShards类似，需要尽可能快的探索遍历整个地图，避免出现绕远、来回徘徊不定的问题；二是安全性，由于该minigame中获得胜利并不会得到兵力的补充，因此每个战士能否持续存活也十分重要，agent应当学会避免让战士单打独斗或是走进多个虫族的包围圈中。

在训练过程中，我们遇到的最严重的问题就是agent经过开头短短二十余个episode训练后，策略反而迅速恶化，甚至劣于随机策略，呈现出来的效果是agent一直选择乱移视野而从来不选择让士兵移动攻击。经过分析，我们猜测其原因可能如下：在收获+1reward时，其真正的原因往往是先前某一步采取的攻击移动指令，而非来自于当前帧所采用的指令；由于训练时采用了小于一的衰减因子，agent会错误的将本次reward主要归功于当前动作，而对之前真正有价值的动作归功不足。更令人头疼的是，若某一个动作A在一次更新后偶然成为了选中可能性更高的动作，那么之后每一次收获的reward都更有可能将主要功劳分给动作A，而非真正的有价值动作，于是形成了一个错误的正反馈循环，最终导致了agent在训练过程中迅速崩掉。

我组尝试采用了A3C、PPO等不同的学习方法，也尝试了手动去除无用动作，甚至直接用rule-based agent进行模仿学习，但都难逃网络震荡或是梯度爆炸的结局。对于后续研究工作，我组认为采用LSTM相关方法对该minigame进行学习，应该就能够对这种时效性问题作出不小的进步，不过要想学习时间维上的信息，是必须要大量的算力作为代价。

多次训练过程均呈现出下图趋势，随机初始化agent表现一般，超过数个epoch后迅速震荡或是梯度爆炸。



2.4 DefeatRoaches

该minigame任务是操纵九个marine击败四个虫族，双方最初生成在地图两个对立的区域。每击败一个虫族获得+10reward，每死亡一名战士获得-1reward。当虫族被全部消灭后，地图会重新刷出四个虫族，剩余战士保留当前血量，并额外奖励五个满血战士，双方重新设置在地图两个对立的区域。士兵全部死亡或到达时限后重启游戏。

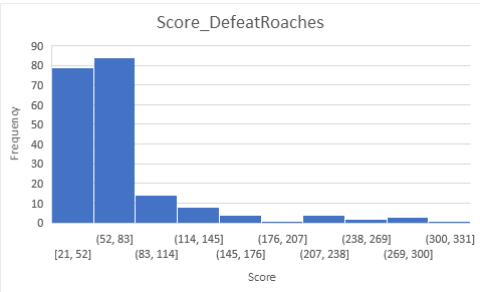
该minigame中需要的actions为select_army(function_id=7)、attack_screen(function_id=12)，与第三个小游戏相同，但因为并没有战争迷雾的干扰且不需要移动视野或者探索地图，所以这个小游戏会比第三个小游戏简单许多。该minigame所需要学到的最重要的策略就是让所有战士集火攻击同一个敌人，从而达到快速消灭敌对战力的效果；如果火力被分散了，每个虫族平均的存活时间也就更长，累积伤害输出也就越多，这是我们不希望看到的。

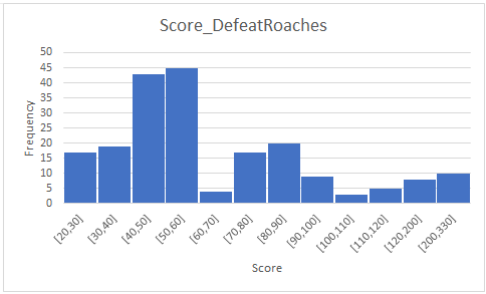
也许是因为算力不够，或者是学习率衰减得过快，我们最开始未进行模仿学习，全靠A3C、PPO算法探索学习的agent在得到一个一般的分数后就似乎停止了探索，有一定集火意识却又迟迟不能收敛至稳定，常常会在多个敌人附近随机点击。我们猜想其原因是agent在决策时是以动作可能性进行sample的，这导致输出动作必定会带有一定的不确定性，且由于无论集火攻击哪个虫族都几乎是等价的，这导致输出策略的分布较为均匀，不能很确定的决定集火攻击哪一个虫族。

为了得到更优秀的结果，我们参照上一个小游戏的训练过程引入了rule-based agent进行模仿学习预训练。rule-based的策略是选择y坐标最大的虫族进行集火攻击，虽然未必会比集火最近的虫族等等策略更优，但这种只与虫族坐标相关、与战士坐标无关的策略显然会更容易学到。实验结果表明，经过预训练后的agent输出稳定性得到了大幅度提升，不会再几个虫族间来回低效点击了。

测试预学习后PPO继续训练200episode的模型，得到max score=323，average score=74.68。实验结果的分布呈现出了极大的方差，且除了在平均分附近集中，在高分段也出现不少结果；这一特性其实并非来自于agent，而是源于该minigame本身含有一个正反馈放大机制，即获胜之后会获得战力补充，连胜两场就能保存不少额外战力，而后续只会越磊越多，战斗更加摧枯拉朽，故得分在高分段也出现了小型聚集。

注：为了能体现出均分附近的详细分布情况，直方图在最后两个频数的统计时扩大了范畴整合数据。





2.5 DefeatZerglingsAndBanelings

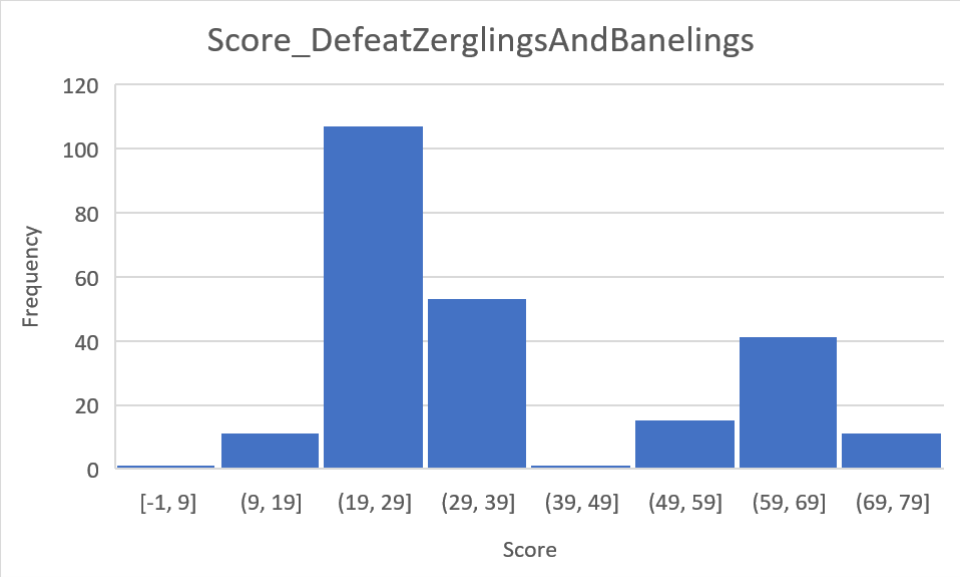
该minigame任务是操纵九个marine击败六个跳虫和四个毒爆虫，双方最初生成在地图两个对立的区域。每击败一个虫族获得+5reward，每死亡一名战士获得-1reward。当虫族被全部消灭后，地图会重新刷出十个虫族，剩余战士保留当前血量，并额外奖励四个满血战士，双方重新设置在地图两个对立的区域。士兵全部死亡或到达时限后重启游戏。

该minigame中需要的actions为select_point(function_id=2)、attack_screen(function_id=12)，与之前两个战斗类小游戏的不同之处在于用select_point取代了select_army，因为后者会选中全部战士一同出击，在碰到毒爆虫(Banelings)时往往会伤亡惨重甚至全军覆没。为了应对毒爆虫的一次性范围高伤害攻击方式，星际玩家们已经总结出了一套应对策略，即“风筝”战术：这是一种针对于毒爆虫所采取的一种诱饵战术，在本游戏中可实践为牺牲一名士兵独自上前引爆毒爆虫，随后剩余士兵再上前消灭剩余虫族。

由于这种战术思维层级较高，我组认为纯靠A3C、PPO算法几乎不可能习得这样的战术操作，于是在一开始就引入了rule-based agent进行模仿学习，然而得到的实验成果却非常不理想，agent最后总是一直执行select_point尝试选人，却完全放弃了attack_screen进行战斗，被“打怕了”。我组对此进行分析，认为原因在于“风筝”战术在执行到诱饵阶段时会暂时收获到一个-1reward，这种负反馈的存在大大干扰了模型对于动作价值的评估过程，造成了agent规避战斗的问题。事实上，这种对rule-based agent进行的模仿学习很难处理稍微迂回一点、带有一定矛盾属性的策略，本实验中我们需要牺牲暂时利益来获取长远收益，但在实践时agent往往卡在第一个阶段前就放弃执行了。

为了鼓励进行战斗，我们重新调整了reward的赋予方式，让agent在环境未返回任何reward的时候收获少量负数reward，形成一种持续“掉血”的效果强迫agent进行战斗，但实验结果迅速变为了全部战士一同出击，虽然不会再停住不动了，得分却依旧并不理想。我组认为其难点可能与第二个minigame非常相似，即需要单独点选某个战士执行某种操作。第二个minigame还只是想让两个战士分开来探索地图，尚且无法实现；本实验需要单独命令一个战士作为诱饵去赴死，其负反馈无疑更加大了学习难度。

我组最后放弃了对rule-based agent进行模仿学习预训练的方法，采用了原始的PPO算法进行了训练与测试，让agent尝试自己探索出合理的策略。测试PPO继续训练200episode的模型，得到max score=77，average score=36.675。



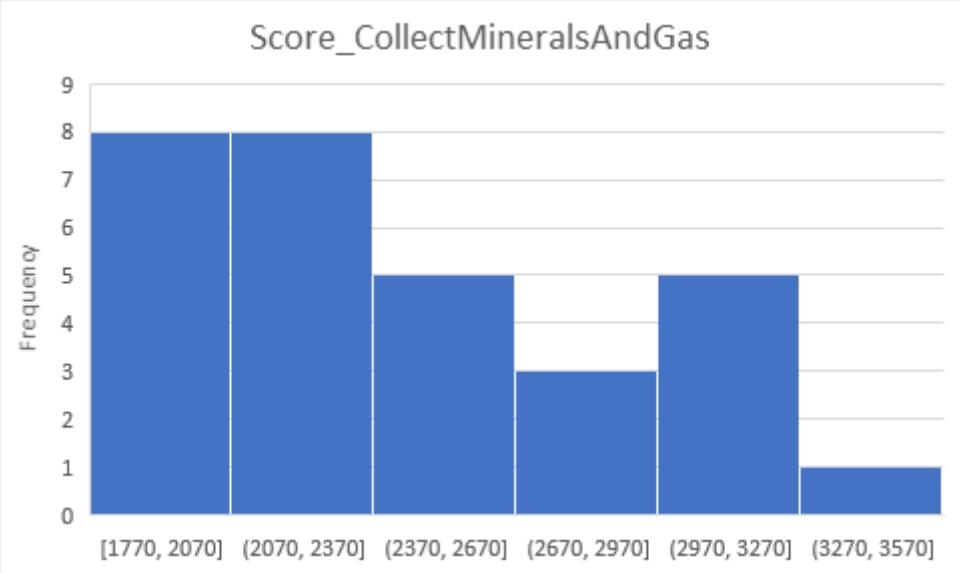
2.6 CollectMineralsAndGas

该minimap任务包含12个SCV，1个指挥中心，16个矿场和4个瓦斯间歇泉的地图。 奖励基于收集的矿物质和瓦斯气的总量。 花费矿物质和瓦斯气体来训练新部队并不会减少您的报酬。 要实现最佳采集，就需要通过建造额外的SCV和额外的指挥中心来扩大收集矿物质和瓦斯气体的能力。

为了取得较高的分数，该minimap需要的action为select_point(function_id=2),Build_Barracks_screen(function_id=42),Harvest_Gather_screen(function_id=264),Build_SupplyDepot_screen(function_id=91),Train_Marine_quick(function_id=477),Train_SCV_quick(function_id=490)。是一个很大的动作集，动作种类繁多导致较少的训练时间很难得出一个较好的效果，我们尝试使用rule_base预训练，但是效果并不是很明显，所以我们舍弃了预训练，直接用A2C进行强化学习训练，没有时间使用PPO进行强化学习训练。由于该地图不对observation进行数据处理很容易梯度爆炸，所以我们首先将observation中的screen和minimap中的标量参数进行one hot处理，提高特征层的维数，降低梯度爆炸的风险。但是导致训练速度变慢，所以算力不足，没有训练出一个最好的模型。

训练时由于该地图只需要在初期让工人去采矿，之后一直采取no option就可以得到一个可观的分数，但是由于A2C的on policy算法，在一张地图的大多数时候，no option的动作就可以获得不错的reward，由于我们一般是在一个epidode结束后储存模型，导致训练结束之后即使是在初期，agent也以一个很大的概率选取no option动作，所以虽然在训练的时候有一个较好的分数（3000+），但是测试的时候分数就比较低甚至没有分数。我们尝试将模型的储存点设置在游戏初期，在得到了10到100个score的时候就储存模型。这样的操作取得了一定的成效，但是没有完全解决问题，这样训练出来的模型最好也只能达到2000+。

由于这是我组训练的最后一个模型，算力不够，所以我们训练出来的模型只是进行了采矿的动作，没有训练新的部队和收集瓦斯气体，我组认为如果持续使用PPO进行训练，会取得一个很好的效果。我们的训练结果如下：



max score=3305，average score=2427

2.7 BuildMarines

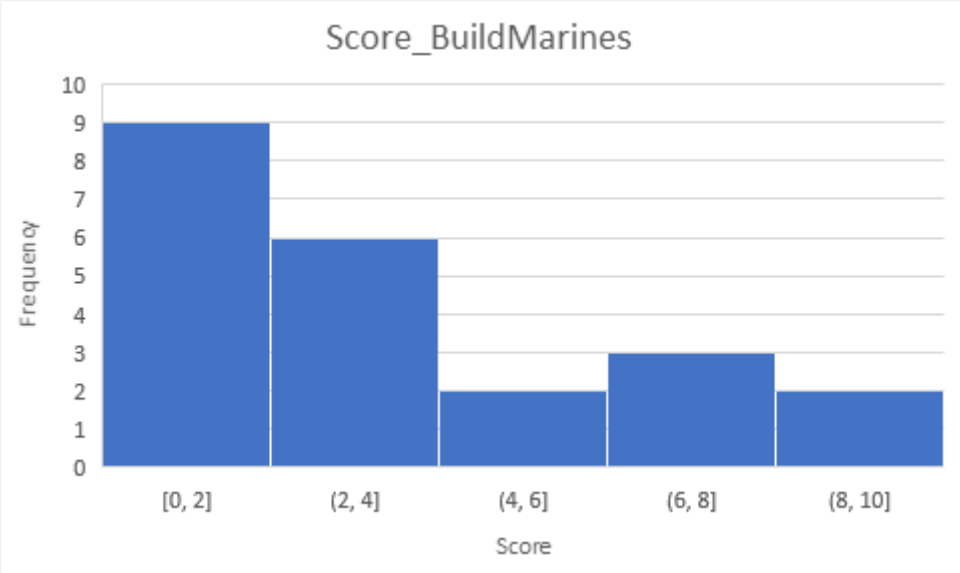
该minigame任务是积累矿物修建补给站和兵营，最后训练出士兵越多越好。游戏开始时会在固定位置生成一个指令部和八个矿物产地，同时会生成12个未被选中、已在进行挖矿的工人。由于兵营的修建需要至少一个补给站的存在，士兵的训练也需要足够多的补给站提供食物，因此游戏的主要流程大致可分为：积累矿物 -> 修建补给站 -> 修建兵营 -> 持续训练士兵 -> 后期策略，其中后期策略需要规划继续修建补给站和兵营各自的数目比例，还可以尝试训练出更多工人来快速获取大量矿物。

该minigame中需要的actions较为繁多，共有六个，分别是select_point(function_id=2)、Build_Barracks_screen(function_id=42)、Build_SupplyDepot_screen(function_id=91)、Harvest_Gather_screen(function_id=264)、Train_Marine_quick(function_id=477)、Train_SCV_quick(function_id=490)。由于动作种类繁多，时间跨度极长，我们曾一度担心第一个reward可能很难通过初始的随机策略得到，后续的训练也就更加无从谈起了。但事实上，虽然动作繁多，但他们并不是在一开始就全部开放了的，而是在经过一段时间矿物积累后一步步开启下一个阶段，逐渐解锁新的动作，因此动作空间得以大幅多缩减，使得随机策略也有一定可能造出士兵收获reward，尽管开始时效率低下，但在后续学习中会不断优化、提高效率。

我组在训练该minigame时并没有进行模仿学习预训练，只靠PPO算法最后就学到了不错的成果。我们对此的理解是，在收获+1reward时当前帧可能在执行各种不同的动作，但在某固定数量的帧数之前，一定会有一个造兵的动作，因此造兵一定会在学习过程中不断提高其动作估值；而造兵固定需要的前提条件就是兵营，因此建兵营的的动作估值也会逐渐提高；继续往前推理，由于这些操作逻辑都是硬性的必要条件，而非可有可无的软条件，因此更前期的动作估值总能随着后续动作估值的提高而逐渐学习提高，最后得到一个可以收获成果的动作逻辑链条。

虽然这种必要性推理能造出士兵，但它其实还与最优策略有一定距离，建补给站和建兵营可以大量并行工作，且其比例的决策也颇为考究，我组认为这些后期策略通过PPO算法也能稍加探索，但在算力有限的情况下已经很难再有大幅度优化了。

由于该minigame持续时间太长，我组只对PPO训练200episode的模型进行了20次测试，但也能看出实验成果分布如何了，得到max score=9，average score=2.78。



3. 实验分析与结论

在数周的学习研究与实验中，我组对pysc2实验环境有了较为深入的了解，在不断试错与debug过程中学会了其复杂的接口交互，最终搭建起了可以进行学习训练的agent模型。在尝试了A3C、PPO、模仿学习预训练三种学习途径，并针对每个minigame进行了深入的分析与改良调整后，我组进行训练并得到了以下的测试平均结果，与作业要求的Deepmind论文《StarCraft II: A New Challenge for Reinforcement Learning》中的平均结果进行了对比：

	Game1	**Game2**	**Game3**	**Game4**	**Game5**	**Game6**	**Game7**
Deepmind	25	**96**	**44**	**98**	62	3351	<1
Ours	**25.7**	75.64	~	74.68(方差极大)	36.675	2427	**2.78**

可以看出，我组在game1、game7获得了很好的结果，game2表现一般是由于agent尚未学会两个战士分开遍历地图，game3没能训练出收敛网络是由于该小游戏的战争迷雾与移动视野两个障碍大大增加了学习难度，game4的测试结果方差极大，实际也取得了不少超300的优秀结果，game5，game6

总体而言，未来可供进一步研究的方向主要有以下几个：

1. Loss爆炸：在学习过程中多次出现了计算Loss得到NaN的情况，我们认为有两个原因，其一是计算Loss函数时需要计算policy的log值，当policy中有的值过小时，计算log后会得到NaN；其二是学习率过大导致梯度爆炸。在对policy进行clip以及调低学习率后，该情况得到明显改善。除此之外，预处理过程中的对数变换也能起到防止梯度爆炸的作用。
2. 策略固化：agent在学会一个有所收获的初级策略后逐渐收敛稳定，缩减了探索的步长与范围，导致无法训练得出可能更优的策略。通过改变reward形式可以激励agent探索其他策略，但也可能导致其不稳定难以收敛，解决方案仍需进一步研究。
3. 不愿牺牲当前利益：由于计算动作估值时有小于一的衰减系数参与，长远的收益经过加权后往往无法抵消短期的负收益，且延迟越久的收益随机性越强，进一步加大了agent学会牺牲当前利益的难度。通过增大衰减系数接近于1可以略微缓解这一问题，但看重长远收益的代价是进一步增大了收益评估时的随机成分，使得网络的收敛速度减慢。
4. 复杂策略难以模仿：在game2中想让两个战士分头遍历的策略需要来回点选不同战士，这个来回的过程具有一定的时效性、周期性；game5中命令一名战士作为诱饵赴死的“风筝”战术分为了勾引和作战两个阶段，同样具有很强的时效性、顺序性，且不同于game7的阶段性策略，通过动作的逐个开放在环境交互中内嵌了一个必要推导的逻辑链条，为agent提供了时序信息，game5的“风筝”战术完全需要在一个开放的动作空间里自行习得获取阶段时序信息的方式。对于这种难题，下一点LSTM架构可能会提供不小帮助。

5. LSTM架构：我组在选择网络架构时尝试了Deepmind论文中提到的Atari-net, FullyConv 和 FullyConv LSTM三种架构，经过第一个minigame测试后发现第二种FullyConv得到了较好的结果，因此在后续小游戏沿用了这种架构。但对于一些时序性，阶段性较强的小游戏，LSTM有可能会得到更优的结果，值得尝试，不过代价是需要大量的算力来学习时间维上的信息。