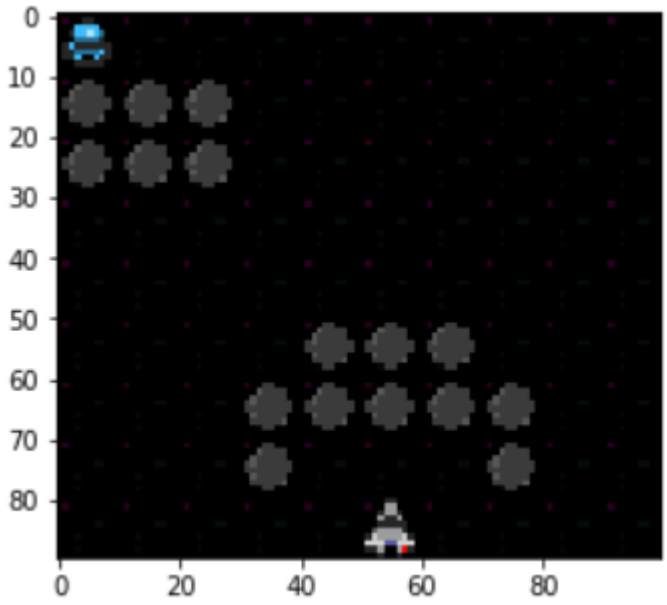
AAI Assignment5

11849332-王小雪

一 算法描述

1 游戏概述

本次实验使用 Q-learning 算法实现一个 agent，用来玩一个单人游戏。
Agent.py 用于每次 act 决策，testRLAgent.py 则进行整个游戏。**trainRLAgent.py 进行模型训练。**
游戏初始状态如下图所示：



图一 初始状态

由初始状态可以得到很多信息：
① 整个状态像素是 90X100，每个物体占位为 10X10，整个地图可以看成是 9X10 大小
② 每个物体的像素差异很大，只需要 10X10 中的少许像素即可了解当前物体是什么

在游戏进行过程中，以及打印出 actions，可以得到更多信息：
③ 游戏中一共有 5 种物体：飞机/外星人/石头/子弹/炸弹
④ 飞机是可控的，每次的行动有 4 个：'ACTION_NIL', 'ACTION_USE', 'ACTION_LEFT', 'ACTION_RIGHT'，意味着飞机可以左右行走，以及发射子弹，不动。
游戏规则不在赘述。

2 算法概述

本次实验使用 Reinforcement Learning 中的 Q-learning 算法。
当飞机处于不同位置时，我们称为不同的状态。在每一个状态中，我们要决定选择一个 action，从而得到更多的分数。为此填充 Qtable。

Qtable 是一个表格。通过它，我们可以为每一个状态（state）上进行的每一个动作（action）计算出最大的未来奖励（reward）的期望。Qtable 格式如下图:

	actions' score			
state	ACTION_NIL	ACTION_USE	ACTION_LEFT	ACTION_RIGHT
Cur_state	0	0	0	0

图二 Qtable 格式

每次行动时，只需要根据表格找到最大分数相应的 action 返回即可。

Q-learning 算法就是填充 Q-table。算法伪代码如下(某些细节将在 3 中说明，如α的含义):

Algorithm ： Q-learning

Input: None

Outout: a good Q-table

```
Initialize Q-table arbitrarily
α←0.5，γ←0.9，β←0.05
Initialize epsilon = 1
for t from 1 to 1000 do
    epsilon ← exp(-β* t)
    s ← getCurrentState()
    actions ← getCurrentPossibleAction()
    if random.rand() < epsilon do
        a ← randint(0,len(actions)-1)
    else
        a ← action index of max(Q-score(s)) in Q-table
    endif
    newState, increScore, done, debug ← env.step(a)
    s' ← newState
    R ← getReward(increScore)
    Q(s,a) ← Q(s,a) + α[R + γ·max_a'Q(s',a') - Q(s,a)]
    if done do
        break
    endif
endwhile
```

算法 1 Q-learning

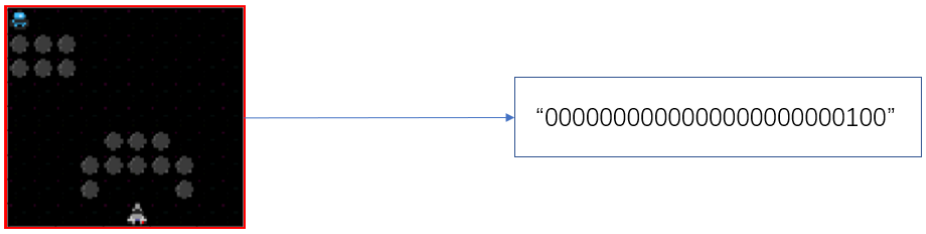
3 算法细节

针对以上概述，将所有细节描述如下:

① 填充 Q-table 时，状态的表示非常重要。我曾使用了三种表示，最终选择了效果最好的一种。

最终表示：以飞机为中心的 5X5 的地图信息。保存 5X5 长度的字符串，每个字符串表示相应位置的物体，飞机/外星人/石头/子弹/炸弹/空 分别表示为 1，2，3，4，5，0。考

虑到状态过多，**为了让胜率更大，因此保存时，只考虑外星人，飞机及炸弹的位置，分别表示为 1，2，3，其余均表示为 0。这种做法会导致最终得分不高**(未考虑石头)，**胜率却较大**(更加关注外星人和炸弹，因此更容易躲过炸弹)，**所有状态数少，训练时间短。**例如，下面状态表示如下图：



图三 状态表示

当飞机处于边缘时，只表示以飞机为中心的 5X3 大小，或者 5X4 大小。

② 发现物体时，具体是根据 10X10 中心点及中心左侧点像素点之和作为考虑，从而判断该 10X10 是何种物体。例如图三中的外星人，考虑

S = sum(stateObs[5][5]) + sum(stateObs[5][4])

即可了解所有物体，根据测试得结果如下：

①飞机：S=1446；②外星人：S=738；③炸弹：S=1863

③ 每次的行动有 4 个：'ACTION_NIL', 'ACTION_USE', 'ACTION_LEFT', 'ACTION_RIGHT'。当飞机处于边缘时，不做更多考虑；因为即使处于边缘，env.step(a)执行无法行动的动作时，会自动执行 ACTION_NIL。

④ 更新 Q-table 所用的 reward 并非直接使用返回的 increScore，而是做了相应处理；处理过程如下：

Algorithm ： getReward

Input: increScore

Outout: Reward

```
1   Initialize Reward ← 0
2   if increScore = -1 do
3       reward ← -80
4   elif increScore = 0 do
5       reward ← -0.2
6   elif increScore = 2 do
7       reward ← 20
8   elif increScore = 1 do
9     reward ← 5
10  endif
11  return reward
```

算法 2 getReward

这样做的目的是，当游戏失败时，给较大惩罚；当未得到任何分数时，给少许惩罚，促使飞机更多样化尝试，从而可以更容易发现好的或者差的行动；当打中外星人时，给较多奖励，因为这样更容易使得游戏成功；当打中外星人时，给较少奖励，因为这样虽然得分更高，但对于游戏胜利影响不大。

游戏胜利时，得分相对更高，因此考虑让外星人更大概率赢得游戏，而不是得更多的分。

⑤ 算法中的 epsilon 参数是随机探索率。游戏开始，由于 Q-table 中都是不准确的值，因

此飞机的行动在最初时会随机。当产生的(0,1)随机数小于 epsilon，飞机会随机行动；当产生的(0,1)随机数大于 epsilon 时，飞机会根据 Q-table 进行选择。epsilon= exp(-β* t)，随着游戏时间增加，epsilon 会逐渐减少，因为 Q-table 将会越来越准确，采用 Q-table 行动的概率将会增加，β = 0.05 是经验值。**而在训练结束后，即测试阶段，每次行动时均利用 Q-table 行动。**

⑥ 更新 Q-table 采用的方程如下：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \cdot \max_{a} Q(s',a') - Q(s,a)]$$

其中方程右边的 Q(s,a)是更新前的 Q 值，左侧是更新后的 Q 值。

α是学习率，看作是有多快地抛弃旧值、生成新值的度量。如果学习率是 1，新的估计值会成为新的 Q 值，并完全抛弃旧值。γ为折扣因子，表示对行动后的状态奖励的一个折扣。α取 0.5，γ取 0.9 均为经验值。R 的计算在④已经叙述。

⑦ 游戏中有少许 BUG，实际过程不需要对 BUG 进行过多处理。因为胜利时最终返回状态为 PLAYER_WINS，失败时返回 PLAYER_LOSES，由于持续到游戏结束则会返回 NO_WINNE。

二 模型训练

1 整体训练思路

a) 本次实验训练时，在 Agent.py 中增加了一个新的类 trainAgent，用于训练 Q-learning。在 trainRLAgent.py 中训练，构建了 trainAgent 的对象。Q-learning 具体过程已在算法描述中叙述。

b) 训练中，每次将执行 10 次游戏。每次游戏均最多进行 1000 个 tick(作业要求写的是 1000 个 tick，而代码中为 2000 个 tick，实际采用 1000 个 tick)。

c) 以 10 次游戏为一个循环。每次循环前，均从文件中读取当前的 Q-table；循环结束后，将训练好的 Q-table 写入到 json 文件。

d) 程序中保存 Q-table 使用 python 中的字典类型，读写文件时均与 json 类型互换。字典中的 key 表示状态，即一的算法细节中每个状态所对应的字符串；字典的 value 值表示行动得分，是一个长度为 4 的 list，表示当前状态对应每个行动的得分。行动顺序为 'ACTION_NIL', 'ACTION_USE', 'ACTION_LEFT', 'ACTION_RIGHT'。例如，[1,0,-1,3]表示向右的动作的奖励是 3，是最大值，因此当前状态会向右走。

   Q-table 中某次状态对应的 Q 值格式(字典和 json)如下：

   "00000000032000000210": [-17.96463658614988, 0, 0, 0]

e) 训练时，采用过三种状态表示方式，分别是

   ① 保存全部整个地图信息以及所有物体，未达到收敛，状态数非常多；

   ② 保存以飞机为中心的 5X5 大小的地图信息，包括所有物体，总状态数达到 20000；

   ③ 保存以飞机为中心的 5X5 大小的地图信息，不考虑石头，子弹等信息，总状态数只有 914；

   最终采用③，虽然状态数很小，但是最终的胜率却比较高。②的训练状态多，虽然也达到了收敛，但实际胜率不如③。

2 参数值及训练代数

本次实验采用 Q-learning 算法，本次实验参数不多，所有参数及含义如下：

表 1-Q-learning 参数值及含义

参数	值	含义	如何取值
α	0.5	学习率，多快地抛弃旧的Q值、生成新 Q值的度量	经验值
γ	0.9	折扣因子，表示行动后的状态奖励的折扣率	经验值
β	0.05	随机探索率减小率，β增加表示epsilon减少更快，更大概率采用Q-table中的值来行动	经验值
R	与返回的increScore相关	每次行动后的奖励	考虑让agent避免失败，并且更多的打中外星人

训练时，每次循环进行 10 次游戏。状态数变化即训练次数如下表所示：

表 2-训练代数及状态时间变化

每次循环游戏次数	10
训练至Q-table收敛的循环数	160
训练代数	1600
状态数变化	0→914
平均每次训练时间(10次游戏)	76.8s

说明：由于状态表示尽量简化，因此总的状态数和训练代数均比较少；继续训练，状态数会少许增加，但增加缓慢，效果几乎不变。实际使用过许多模型，选择了一个效果更好的作为最终模型。

3 action value function

训练时更新 Q-table 的方程如下：

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)]$$

Q(s,a)即为 action value function，表示状态 s 下执行动作 a 给出的奖励。

Q(s,a)的定义如下：

$$Q_{\pi}(s,a) = E[Gt|_{\pi}, St=s, At=a]$$

其中

$$\boldsymbol{G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}}$$

因此

$$\begin{aligned} \boldsymbol{q_\pi(s,a)} &= \mathbb{E}[\boldsymbol{G_t}|\boldsymbol{\pi}, \boldsymbol{S_t = s, A_t = a}] \\ &= \mathbb{E}[\boldsymbol{R_{t+1} + \gamma G_{t+1}}|\boldsymbol{\pi}, \boldsymbol{S_t = s, A_t = a}] \\ &= \mathbb{E}[\boldsymbol{R_{t+1} + \gamma v_\pi(S_{t+1})}|\boldsymbol{\pi}, \boldsymbol{S_t = s, A_t = a}] \end{aligned}$$

因此更新函数中的 $R + \gamma \cdot \max_{a'} Q(s',a')$即为新的 Q 值。而更新 Q(s,a)时，会考虑旧的 Q 值与行动后的新的 Q 值。学习率的大小反映考虑新值的程度。

训练过程中的 reward 已经在算法细节中给出，不再赘述。

4 Agent 表现测试及评估

由于训练时，每次循环将进行 10 次游戏，并且打印出 Agent 的总得分，胜负，游戏的 tick 数。

① 对 Agent 的表现测试时，能够根据打印结果得到**每次循环(10 次游戏)的得分平均值，胜**

率，游戏的 tick 数。由此对 Agent 的表现进行测试。

② 对 Agent 的表现进行评估时，考虑 10 次游戏的得分均值，胜率；得到的**评估函数**如下：

value function of performance = 0.6 * Score + 0.3 * winRate + 0.1 * gameLength

其中 Score 是 10 次游戏的平均得分，winRate 是胜率 X200，gameLength 则是 10 次游戏的平均 tick 数/20。为了保证得分大小相差不大，因此对胜率乘以 200，tick 数除以 20，保证每一项相差不大。

这个评估考虑了得分，胜率，以及游戏时长的加权平均值。其中得分是首要考虑的，胜率次之；考虑游戏时长，因为它表现了 Agent 的存活时长，同样得分下，Agent 存活时间长，则说明躲过了更多炸弹，表现更智能。

5 Agent 表现结果

下面将给出训练初期的某次结果，并以此结果为例计算 Agent 的表现。

训练初期的某次结果如下：

表 3-训练代数及状态时间变化

游戏次数	得分	是否胜利	游戏时长(tick)
1	24	LOSE	223
2	15	LOSE	145
3	17	LOSE	190
4	17	LOSE	144
5	20	LOSE	193
6	52	WIN	346
7	11	LOSE	145
8	31	LOSE	301
9	11	LOSE	142
10	12	LOSE	145
Mean/winRate	21	0.1	197.4
SD	11.91637529	—	69.06692

根据 4 中的方法计算这 10 次的表现：

value function of performance = 0.6 * Score + 0.3 * winRate + 0.1 * gameLength

value function of performance = 0.6 * 21 + 0.3 * 0.1*200 + 0.1 * 197.4/20 = 19.587

因此这次循环表现分数约为为 19.6。

考虑到总的训练数为 1600 代(代数较少原因已经在前面给出解释)，故而给出每 150 次(即 15 次循环)的结果表现。

表 4-Agent 结果表现(见下页)

代数	得分	10次训练时长(单位:s)	当前状态总数
0-10	8.5585	47	129
150-160	9.3665	49	334
300-310	12.2385	51	485
450-460	14.2674	77	607
600-610	18.404	83	716
750-760	23.5354	85	745
900-910	26.6842	76	776
1050-1060	35.6544	96	807
1200-1210	46.0394	104	839
1350-1360	50.436	115	864
1500-1510	64.647	94	895

可以看到 Agent 得分越来越高，状态数越来越多。时间则趋于稳定。

三 实验结果即结论

1 实验结果

表 5-某 10 次测试的实验结果

游戏次数	得分	是否胜利	游戏时长(tick)
1	46	1	393
2	31	0	289
3	46	1	393
4	46	1	393
5	27	0	289
6	46	1	393
7	46	1	393
8	16	0	161
9	45	1	393
10	49	1	377
Mean/winRate	39.8	0.7	347.4
SD	10.54324428	—	74.10694974

由表可知，胜率在某次测试下可以达到 0.7，分数为 39.8±10.54，时长为 347±74.1 均优于随机的 Agent。但是由于训练时对 state 缩减太少，无法达到更高的分数。

2 实验结论

在经过 RL 训练后，Agent 已经能够达到 0.7 的胜率。分数也能比随机的 Agent 好。
但是**由于状态压缩**，导致 Agent 不能够更加智能，而且**因为每次游戏的相似性，可以发现 Agent 经常会有相似的动作，导致有几次游戏的 tick 和得分均相同。但是这些均为 Agent 自己训练的结果。**
状态压缩会导致训练时间代数短，但也会让 Agent 的上界受限。