作业 4: FreeWay 游戏 实验报告

丁云翔 (191250026、191250026@smail.nju.edu.cn)

摘 要: 在本次作业中，需要阅读程序，理解其中使用的强化学习算法，并尝试修改程序提高学习性能。阐述强化学习的方法和过程；尝试修改特征提取方法，得到更好的学习性能；尝试修改强化学习参数，得到更好的学习性能；并报告修改的尝试和得到的结果。

关键词: 强化学习，特征提取方法，强化学习参数。

1 任务 1

强化学习的方法和过程：框架代码在 act 函数中调用 learnPolicy 函数来完成强化学习，并根据强化学习得到的策略返回一个最优动作以供 Agent 执行。强化学习的方法和过程主要体现在 learnPolicy 函数中，在 learnPolicy 函数中进行了 10 次迭代，每次迭代都调用一次 simulate 函数来模拟，每次模拟的最大深度为 20 层。模拟探索采用了 epsilon-greedy 策略，有 epsilon 的概率随机选取一个动作探索，有 1 - epsilon 的概率选取 Q 值最大的动作来探索，并通过一个公式来更新 Q。每轮迭代都会根据模拟探索得到的结果更新数据集。迭代结束后，learnPolicy 函数会调用 fitQ 函数，根据模拟探索得到的数据集使用 weka 的 REPTree 模型训练策略。这样就完成了一次强化学习的过程

问题一：策略模型用一个有 epsilon-greedy 的 Q-learning 方法表示。该方法的缺点一个在于 m_epsilon 值固定，而 m_epsilon 随着探索的进行减小效果会更好。因为再开局时探索范围小，不确定性较大，应更偏向随机探索；而到后期探索范围较大了，不确定性小，应更偏向于选择 Q 值高的动作。另一个在于每次都选择 Q 最大的动作，容易导致过估计。所以应该将 m_epsilon 改为一个变量，随着探索的进行，其值应在一定范围内不断减小。另外可以采用 Double Q-Learning 的方法，防止出现过估计。

问题二：Agent.java 代码中 SIMULATION_DEPTH 变量表示模拟采样的最大深度，该参数值为 20，限制了模拟采样进行的最大深度为 20。m_gamma 变量表示折扣因子，每一轮模拟时得到的 factor 都要乘以这个 m_gamma，它可以控制未来不同深度下采样回报的重要程度，如果它的值小（接近 0），则更重视层数较小时的回报；如果它的值大（接近 1），则层数对回报的影响小，这里的 m_gamma 值为 0.99，说明层数对回报的影响较小，但层数高时的回报还是会被减弱。m_maxPoolSize 变量表示 m_dataset 的最大容量，这里它的值为 1000，说明最多只能存储 1000 个状态。

问题三：两个函数的不同之处在于 getAction 函数应用了 epsilon-greedy 策略，而 getActionNoExplore 函数则没有使用。而 epsilon-greedy 策略主要是解决选择最优动作还是随机选择动作的问题，getAction 函数有一定概率随机选择动作，而 getActionNoExplore 则总是选择 Q 值最大的动作。正因如此，getAction 函数用在 simulate 函数中，用来进行模拟探索，因为探索时需要一定的随机性，从而可以避免贪心算法的缺陷，提高找到最优解的概率；而 getActionNoExplore 函数用在 act 函数中，情境是已经完成探索了，需要根据当前特征进行决策，选取执行的动作，所以直接选取最优的动作就行了。

2 任务 2

首先我直接使用框架代码运行了一次，可能是由于电脑硬件的限制，运行速度比较缓慢，大约半小时才进行到第八轮游戏，然后程序因为出现了爆堆的情况而被迫中止。通过对前八轮游戏进行观察，我发现框架现有的学习方法效果较差，没有一次到达顶端，基本只在最下面两层活动，尤其是右下角的部位，因此我尝试对特征提取方法进行改进。

原有的特征提取方法记录了地图中所有的位置信息，以及 GameTick、AvatarSpeed、AvatarHealthPoints 和 AvatarType 四个参数。在此基础上我添加了 Avatar 的横纵坐标，因为地图是固定的，而目标均位于最上层，一种可行的方法是先以一种较容易地固定路径移动到最顶层，再左右移动到达目标，因此 Avatar 的横纵坐标是一个重要特征；添加了 Avatar 的前方是否有障碍物这一特征，防止其被卡在障碍物前；添加了精灵与目标的横向距离和纵向距离这两个特征，希望能让它更有针对性地向目标移动；加入了 Avatar 所在行和 Avatar 前方一行中 Avatar 左侧或右侧（根据高度而定，因为移动障碍物的运动方向不同）最近的移动障碍物的横向距离这两个特征，希望它能躲开移动的障碍物。

修改完运行以后发现这次 Avatar 能够向上越过最低的那层固定障碍物了。但又发现了新的问题，Avatar 对于绕开固定障碍物的能力较差，所以我又加入了一个记录 Avatar 前方一行中 Avatar 两侧最近的固定障碍物的横向距离这一特征。

这次修改完以后再运行，Avatar 能够向上越过中间的那层固定障碍物了，又有所进步。但是积极的向上运动仅限于开局时血量较满时，当血量较少时，Avatar 依旧只会在最低两层活动。

最终的代码修改如下：

makeInstance 函数与 featureExtract 函数：

```
public static Instance makeInstance(double[] features, int action, double reward){
    features[880] = action;
    features[881] = reward;
    Instance ins = new Instance( weight: 1, features);
    ins.setDataset(s_datasetHeader);
    return ins;
}

public static double[] featureExtract(StateObservation obs){

    double[] feature = new double[882];  // 868 + 12 + 1(action) + 1(Q)

    // 448 locations
    int[][] map = new int[28][31];
```

```
// Extract features
Vector2d avatarPos=obs.getAvatarPosition();
double avatarX = avatarPos.x;
double avatarY = avatarPos.y;
boolean up = true;
double distanceX = 0;
double distanceY = 0;
double sameMoving = 10000;
double frontMoving = 10000;
double frontImmoving = 10000;
LinkedList<Observation> allobj = new LinkedList<>();
if(obs.getImmovablePositions() != null){
    for(ArrayList<Observation> l : obs.getImmovablePositions()){
        allobj.addAll(l);
        for(Observation o:l){
            if(o.position.x == avatarX && o.position.y + 28 == avatarY){
                up = false;
            }
            if(avatarY == o.position.y + 28){
                frontImmoving = Math.min(frontImmoving, Math.abs(o.position.x - avatarX));
            }
        }
    }
}

if(obs.getMovablePositions() != null){
    for(ArrayList<Observation> l : obs.getMovablePositions()) {
        allobj.addAll(l);
        for (Observation o : l) {
            if (o.position.y == avatarY) {
                if(avatarY >= 196){
                    if(avatarX - o.position.x >= 0){
                        sameMoving = Math.min(sameMoving, avatarX - o.position.x);
                    }
                }
                else{
                    if(o.position.x - avatarX >= 0){
                        sameMoving = Math.min(sameMoving, o.position.x - avatarX);
                    }
                }
            }
            if (o.position.y + 28 == avatarY) {
                if(avatarY >= 224){
                    if(avatarX - o.position.x >= 0){
                        frontMoving = Math.min(frontMoving, avatarX - o.position.x);
                    }
                }
```

```
                else{
                    if(o.position.x - avatarX >= 0){
                        frontMoving = Math.min(frontMoving, o.position.x - avatarX);
                    }
                }
            }

        }
    }
}
if(obs.getNPCPositions() != null){
    for(ArrayList<Observation> l : obs.getNPCPositions()) allobj.addAll(l);
}
if(obs.getPortalsPositions()!=null) {
    for (ArrayList<Observation> l : obs.getPortalsPositions()) {
        allobj.addAll(l);
    }
}

for(Observation o : allobj){
    Vector2d p = o.position;
    int x = (int)(p.x/28); //squre size is 20 for pacman
    int y= (int)(p.y/28);  //size is 28 for FreeWay
    map[x][y] = o.itype;
    if(o.itype == 4) {
        distanceX = avatarX - o.position.x;
        distanceY = avatarY - o.position.y;
    }

}
for(int y=0; y<31; y++)
    for(int x=0; x<28; x++)
        feature[y*28+x] = map[x][y];

// 4 states
feature[868] = obs.getGameTick();
feature[869] = obs.getAvatarSpeed();
feature[870] = obs.getAvatarHealthPoints();
feature[871] = obs.getAvatarType();
feature[872] = avatarX;
feature[873] = avatarY;
feature[874] = up ? 1000.0 : -1000.0;
feature[875] = distanceX;
feature[876] = distanceY;
feature[877] = sameMoving;
feature[878] = frontMoving;
feature[879] = frontImmoving;

return feature;
```

datasetHeader 函数：

```
Attribute att = new Attribute( attributeName: "GameTick" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarSpeed" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarHealthPoints" ); attInfo.addElement(att);
att = new Attribute( attributeName: "AvatarType" ); attInfo.addElement(att);
att = new Attribute( attributeName: "avatarX" ); attInfo.addElement(att);
att = new Attribute( attributeName: "avatarY" ); attInfo.addElement(att);
att = new Attribute( attributeName: "up");attInfo.addElement(att);
att = new Attribute( attributeName: "distanceX");attInfo.addElement(att);
att = new Attribute( attributeName: "distanceY");attInfo.addElement(att);
att = new Attribute( attributeName: "sameMoving");attInfo.addElement(att);
att = new Attribute( attributeName: "frontMoving");attInfo.addElement(att);
att = new Attribute( attributeName: "frontImmoving");attInfo.addElement(att);
//action
```

3 任务 3

针对在任务二中通过修改特征提取方法无法解决的问题，我希望能够通过修改强化学习参数来解决。

阅读强化学习框架中原有的启发式函数代码，我发现这个函数基本没有什么作用，如果赢了就返回 1000，输了就返回-1000，如果没赢也没输就返回当前局面的游戏得分，但是因为单次游戏中一次都没有达到目标所以都是 0，而且每次游戏都是以失败告终，所以这个启发式函数对于游戏的学习基本上没有什么作用。

接下来就是修改启发式函数，简单地添加了一些影响得分的因素，如 Avatar 离目标的距离、Avatar 的血量、Avatar 离本行最近的移动障碍物的距离、Avatar 离前方固定障碍物所在行的最近缺口的距离等等。尝试添加了其中几个因素以后，发现运行时间比之前更长了，卡顿明显（电脑配置有点吃不消），可能是由于修改了启发式函数导致运算量增大所致。因为运行时间实在是太长了，运行一轮游戏都需要十几分钟，而最初的一两轮学习效果并不显著，以现有设备无法观测后续学习效果是否有所改善。故修改启发式函数只好作罢。

接下来我试图对其他几个参数进行修改，其它强化学习参数主要就是 SIMULATION_DEPTH、m_maxPoolSize、m_gamma 和 m_epsilon。SIMULATION_DEPTH 是最大深度，理论上深度越大，距离成功就越近，学习效果也会更好，但考虑到设备性能和时间开销，SIMULATION_DEPTH 不能太大。m_maxPoolSize 是数据集的大小，理论上数据集越大学习效果也会更好，同样考虑到设备性能和时间、空间开销，m_maxPoolSize 也不能太大。m_epsilon 和 m_gamma 理论上太高或太低都不合适，应该都有一个中间值达到平衡，使学习效果最好。对这几个参数分别修改后进行测试，发现学习效果还是不明显，我认为原因主要有两点，一是计算量过大，电脑硬件性能不够好，导致花费的时间实在太长，难以观测到迭代后期的效果；二是启发式函数的缺陷，原有的启发式函数过于简单，没有起到它应有的作用，导致学习效果较差，也使修改其他参数对学习效果的影响难以体现，而修改后的启发式函数又大大增加了运算量，使得时间上的开销难以承受。

总结：通过修改特征提取方法和强化学习参数，强化学习效果有所提升，Avatar 探索到的最大高度显著提高，但还是未能到达最高层的目标位置，也未能赢得游戏。我认为如果实验时间更加充裕并且有性能更好的设备，应该能做更多修改，达到更好的效果。