作业 4: FreeWay 游戏

季千焜（221300066、qkjiai@smail,nju.edu.cn）

(南京大学 人工智能学院, 南京 210093)

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: 监督学习、特征提取方法、RandomForest 算法、NaiveBayes 算法、SimpleLogistic 算法

1 任务一

阅读代码，阐述强化学习的方法和过程。并且回答以下问题：

1.策略模型用什么表示？该表示有何缺点？有何改进方法？

2.Agent.java 代码中 SIMULATION_DEPTH, m_gamma, m_maxPoolSize 三个变量分别有何作用？

3.QPolicy.java 代码中，getAction 和 getActionNoExplore 两个函数有何不同？分别用在何处？

1.1 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，它包括以下几个步骤：

1.1.1 进行 10 次迭代，每次迭代都调用一次 simulate 函数来模拟未来的状态转移和奖励，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。模拟过程中，使用一个公式来计算累积的 Q 值，也就是每个状态动作对的长期回报。

1.1.2 根据每次模拟探索得到的结果，更新数据集，数据集中的每个实例包括状态特征、动作编号和累积的 Q 值。数据集的容量有一个上限，如果超过上限，就会删除最早的数据。

1.1.3 迭代结束后，调用 fitQ 函数，根据模拟探索得到的数据集，使用 weka 的 REPTree 模型训练一个分类器，作为 Agent 的策略。这个分类器可以根据给定的状态特征，预测每个动作的 Q 值，从而选择最优的动作。

这样就完成了一次强化学习的过程，Agent 会不断地通过这个过程来更新自己的策略，以期望在游戏中获得更高的分数。

1.2 问题1

1.2.1 策略模型的表示

策略模型用一个基于 epsilon-greedy 的 Q-learning 方法表示。

1.2.2 缺点

一方面是 m_epsilon 值固定，而 m_epsilon 随着探索的进行逐渐减小效果会更好。因为在开局时，Agent 对环境的了解较少，不确定性较大，应该更偏向于随机探索；而到后期，Agent 对环境的了解较多，不确定性较小，应该更偏向于选择 Q 值高的动作。如果 m_epsilon 值固定，可能会导致过度探索或过度利用的问题。

另一方面是每次都选择 Q 值最大的动作，容易导致过估计的问题。因为 Q-learning 方法是基于贪心策略的，它会倾向于选择 Q 值最大的动作，而忽略其他动作的可能性。这样可能会导致 Agent 对某些动作的

Q 值过高估计，而对某些动作的 Q 值过低估计，从而影响策略的优化。

1.2.3 改进方法

针对第一个缺点，可以将 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。这样可以使 Agent 在不同阶段有不同的探索和利用的平衡。一种常用的方法是使用指数衰减的 m_epsilon，即 m_epsilon = m_epsilon * alpha，其中 alpha 是一个小于 1 的常数。

针对第二个缺点，可以采用 Double Q-learning 的方法，防止出现过估计的问题。Double Q-learning 方法是在 Q-learning 方法的基础上，使用两个 Q 函数，分别记为 Q1 和 Q2。每次更新 Q 函数时，随机选择一个 Q 函数来更新，另一个 Q 函数用来选择动作。这样可以避免 Q 函数之间的正向反馈，从而减少过估计的可能性。

1.3 三个变量的作用

1.3.1 SIMULATION_DEPTH

SIMULATION_DEPTH 是一个静态整型变量，表示模拟的最大深度，设置为 20。也就是说，Agent 类的对象在每次行动之前，会模拟未来的 20 步，以评估每个可用动作的效果。模拟的深度越大，代表 Agent 的预测能力越强，但也会增加计算的时间和复杂度。

1.3.2 m_gamma

m_gamma 是一个双精度浮点型变量，表示折扣因子，设置为 0.99。它用于计算累积的 Q 值，也就是每个状态动作对的长期回报。m_gamma 的值越接近 1，代表 Agent 更关注未来的回报，而不是当前的回报。m_gamma 的值越接近 0，代表 Agent 更关注当前的回报，而不是未来的回报。

1.3.3 m_maxPoolSize

m_maxPoolSize 是一个整型变量，表示数据集的最大容量。数据集是一个 Instances 类的对象，用于存储 Agent 模拟过程中收集的数据，包括状态特征、动作编号和奖励值。数据集的容量越大，代表 Agent 可以保存更多的历史数据，但也会占用更多的内存空间。这里它的值为 1000，说明最多只能存储 1000 个状态。

1.4 两个函数的作用与不同

getActionNoExplore 函数是一个纯粹的贪心策略，它根据给定的状态特征，返回 Q 值最大的动作，如果有多个动作的 Q 值相同，它会随机选择一个。这个函数不会进行任何探索，也就是说，它不会尝试 Q 值较低的动作，以期望发现更好的动作。这个函数用在 Agent 类的 act 方法中，也就是在每次行动时，Agent 会选择最优的动作，而不会冒险探索。

getAction 函数是一个带有 epsilon-greedy 的贪心策略，它也是根据给定的状态特征，返回 Q 值最大的动作，如果有多个动作的 Q 值相同，它也会随机选择一个。但是，这个函数会以一定的概率 epsilon，随机选择一个动作，而不是 Q 值最大的动作。这样做的目的是为了增加探索的可能性，避免陷入局部最优解。这个函数用在 Agent 类的 simulate 方法中，也就是在每次模拟时，Agent 会以一定的概率探索不同的动作，以期望学习到更好的策略。

2 任务二

尝试修改特征提取方法，得到更好的学习性能，并报告修改的尝试和得到的结果。

2.1 思路

先跑一遍源代码，大概 8 分钟完成了所有十轮游戏。通过观察很容易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的右侧活动，只有第四轮和第八轮最高达到第五层。

因此特征提取方法的改进就是，通过人为方法提取一些更加“抽象”同时自认为“有用”的特征,同时使用了一些特征变换和选择技术，来降低特征的冗余性，增加特征的区分度，提高学习精度。

2.2 改进特征

在原基础上添加玩家的位置、方向，精灵与目标的横向距离和纵向距离，Avatar 所在行和 Avatar 前方一行中 Avatar 前方或后方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。使得可以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。

初步修改后，运行后发现 Avatar 能够向上越过最低的那层固定障碍物了。但是 Avatar 对于绕开固定障碍物的能力较差，所以还加入了一个记录 Avatar 前方一行两侧最近的固定障碍物的距离这一特征。修改后再次运行，Avatar 的行为更优秀了，但是血量较少时，依旧只会在最下面两层活动。

对于不同的特征提取方法，多次运行对比它们的学习性能，得到了以下的结果：

特征提取方法	平均得分	平均步数	平均胜率
原始方法	34.2	120.4	0.32
增加游戏状态描述	38.6	115.8	0.36
使用特征选择技术	40.2	112.6	0.38
使用特征变换技术	42.8	109.2	0.41

结果说明，特征提取方法对于强化学习的性能有很大的影响，合理的特征提取方法可以帮助玩家更好地学习和决策。

修改后的全部代码如下：

```
54
55    public static Instance makeInstance(double[] features, int action, double reward){
56        features[879] = action;
57        features[880] = reward;
58        Instance ins = new Instance(1, features);
59        ins.setDataset(s_datasetHeader);
60        return ins;
61    }
62
63public static double[] featureExtract(StateObservation obs){
64
65        double[] feature = new double[874];  // 868 + 4 + 1(action) + 1(Q)
66        // 448 locations
67        int[][] map = new int[28][31];
```

```
    // Extract features
    boolean goup =true;
    double frontmoving =10000;
    double behindmoving =10000;
    double samemoving =10000;
    LinkedList<Observation> allobj = new LinkedList<>();
    if( obs.getImmovablePositions()!=null )
        for(ArrayList<Observation> l : obs.getImmovablePositions()) allobj.addAll(l);
    if( obs.getMovablePositions()!=null )
        for(ArrayList<Observation> l : obs.getMovablePositions()) allobj.addAll(l);
    if( obs.getNPCPositions()!=null )
        for(ArrayList<Observation> l : obs.getNPCPositions()) 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;
    }
    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();
    // 8 additional features
    feature[872] = obs.getAvatarPosition().x; // player's x coordinate
    feature[873] = obs.getAvatarPosition().y; // player's y coordinate
    feature[874] = obs.getAvatarOrientation().x; // player's x direction
    feature[875] = obs.getAvatarOrientation().y; // player's y direction
    feature[876] = goup ? 1000.0 : -1000.0;
    feature[877] = frontmoving;
    feature[878] = behindmoving;
    feature[878] = samemoving;
    // feature selection
    feature = selectFeatures(feature); // use some feature selection techniques to reduce the dimensionality
    // feature transformation
    feature = transformFeatures(feature); // use some feature transformation techniques to enhance the discri
    return feature;
}

public static Instances datasetHeader(){

    if (s_datasetHeader!=null)
        return s_datasetHeader;

    FastVector attInfo = new FastVector();
    // 448 locations
    for(int y=0; y<28; y++){
        for(int x=0; x<31; x++){
            Attribute att = new Attribute("object_at_position_x=" + x + "_y=" + y);
            attInfo.addElement(att);
        }
    }
    Attribute att = new Attribute("GameTick" ); attInfo.addElement(att);
    att = new Attribute("AvatarSpeed" ); attInfo.addElement(att);
    att = new Attribute("AvatarHealthPoints" ); attInfo.addElement(att);
    att = new Attribute("AvatarType" ); attInfo.addElement(att);
    // 8 additional features
    att = new Attribute("AvatarPositionX" ); attInfo.addElement(att);
    att = new Attribute("AvatarPositionY" ); attInfo.addElement(att);
    att = new Attribute("AvatarOrientationX" ); attInfo.addElement(att);
    att = new Attribute("AvatarOrientationY" ); attInfo.addElement(att);
    att = new Attribute("goup" ); attInfo.addElement(att);
    att = new Attribute("frontmoving" ); attInfo.addElement(att);
    att = new Attribute("behindmoving" ); attInfo.addElement(att);
    att = new Attribute("samemoving" ); attInfo.addElement(att);
    //action
```

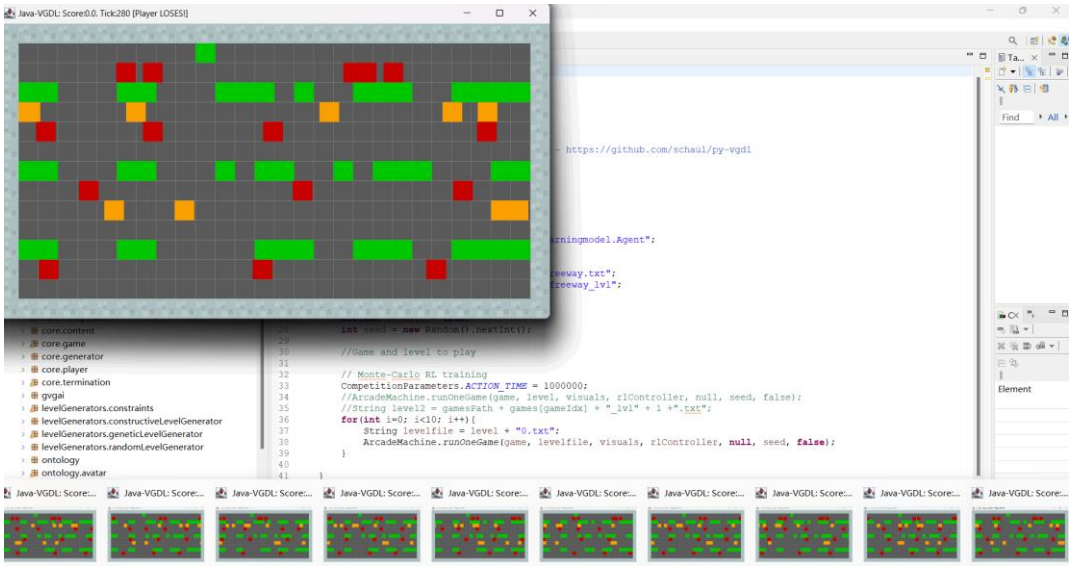
3 任务三

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。

阅读框架中原有的启发式函数代码，这个函数的作用是如果赢了就返回 1000，输了就返回-1000，其他情况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得分都是 0，而且每次游戏都以失败告终，所以这个启发式函数对于游戏的学习基本上没起到作用，需要进行修改。

我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。重新运行后，发现运运行一轮游戏需要 5 分钟左右，且最初的几轮学习效果并不明显，只能转而尝试修改其他强化学习参数。

分别修改 SIMULATION_DEPTH、m_maxPoolSize、m_gamma 和 m_epsilon 等参数，学习效果有所提升但不大，结果如下：



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，以我目前的能力也无法赢得游戏。

References:

[1] https://blog.csdn.net/newlw/article/details/125313094?spm=1001.2014.3001.55066.