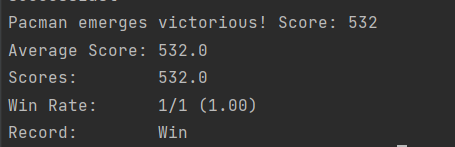
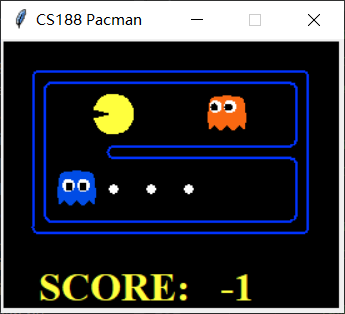
Question2:

使用

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3

大量测试中糖豆人无可避免地死亡（无解）但少量测试中侥幸成功，因为ghost并不智能，他没有选择最佳路径击败糖豆人。

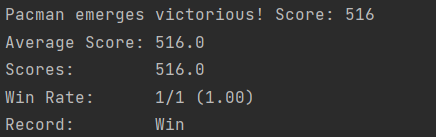


使用

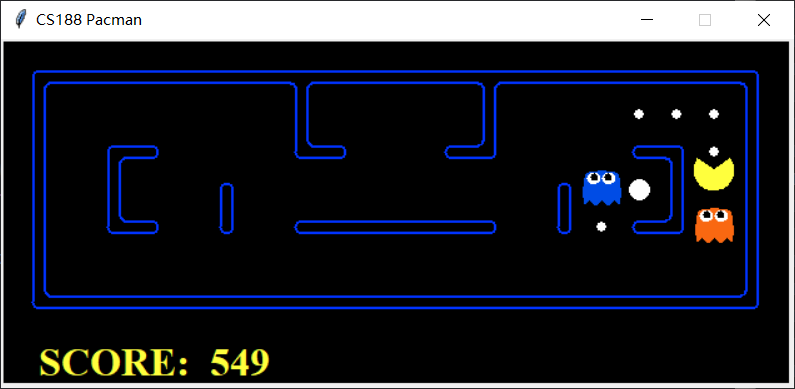
python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

结果与trappedClassic相近。部分测试失败了，但是大部分是有解的，minmax算法帮助糖豆人找到了解。

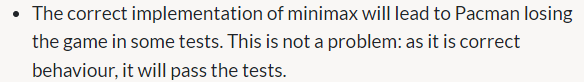




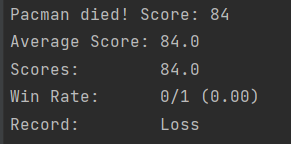
最终为了通过测试花了很长时间研究结尾q2test的结尾为何会被夹击而死。

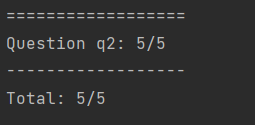


我以为我的程序没有通过测试的原因是糖豆人会选择这条注定死亡的道路，但其实不是：



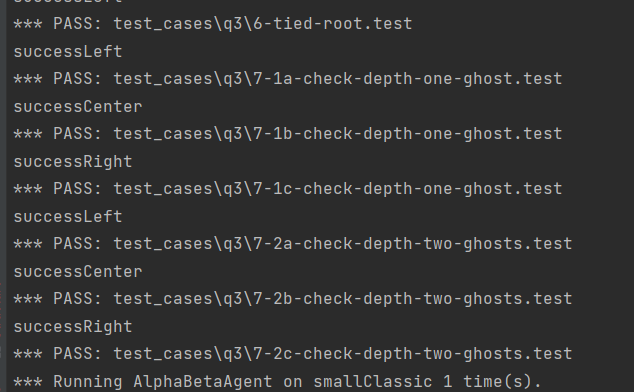
实际上我的程序有一个很小的错误找了我近两个小时，问题出在min函数和max函数中同时递增了树的层数，导致搜索提前终止（因为判断搜索终止的条件之一是到达题目给定的递归层数）最终结果如图：

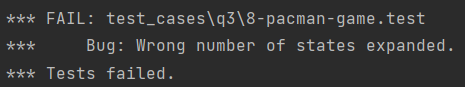




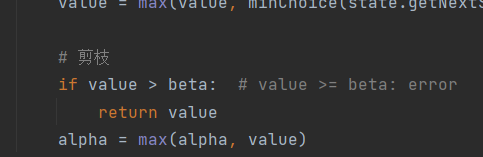
Question3:

Q3只需要在Q2的基础上加上剪枝操作。但是调试仍然花费了我很多时间，通不过最后一个测试用例：

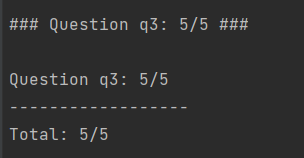




排查超过两个小时后才发现小错误发生在我的max层函数中：应该是alpha大于beta的时候剪枝，但我写成了alpha大于等于剪枝：

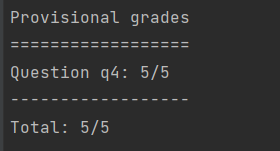


实在是难以发现的错误。最终结果如图：

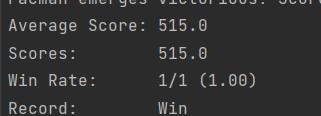


Question4：

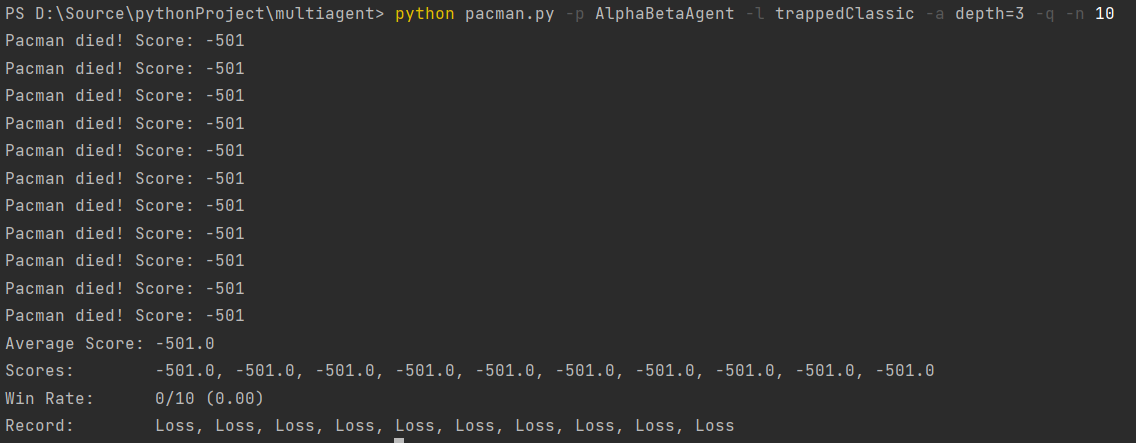
这道题在maxmin的基础上，考虑到ghost是随机行走而不是选择最佳路径，而修改模型适应ghost的算法。其实也很简单，就是在min层不再返回ghost的最佳选择返回的min-value而是返回ghost所有选择返回的综合的期望值。测试结果：

  
python pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3

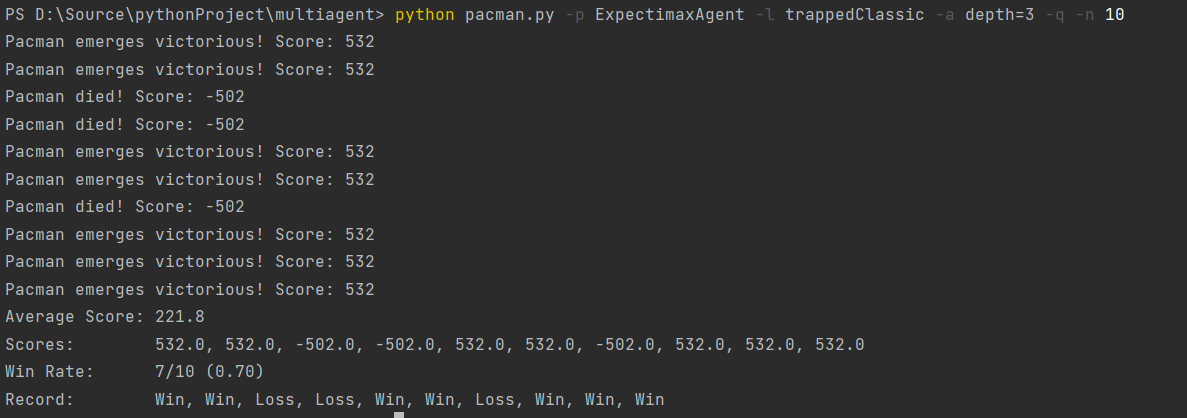




python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10



python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10



Question5:

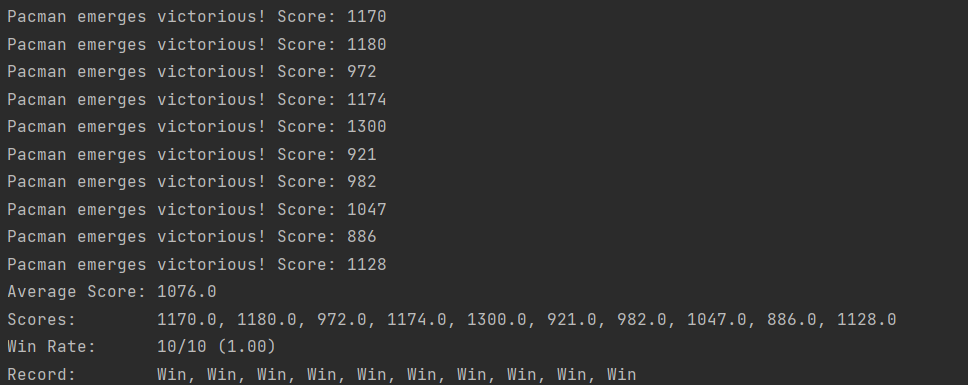
我的evaluateValue回传值由以下部分组成：



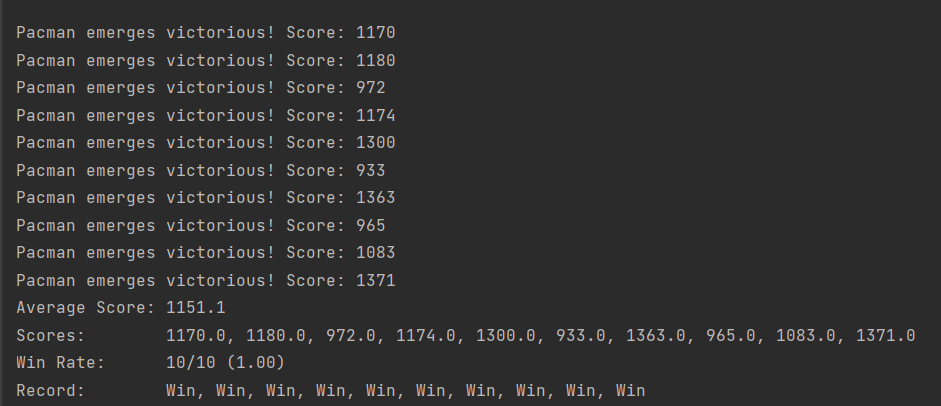
因为发现pacman似乎完全没有在追逐食物而不断调大food的权值，而pacman始终徘徊，甚至调大到一万以上仍然如此，我百思不得其解。随后请教了同学后才知道这是因为food的权值太大了导致的。于是我调小food的权值，程序很快正常运行起来。

下面是一组对比试验数据，food权重越低，pacman越害怕ghost，越容易做出远离ghost的举措：

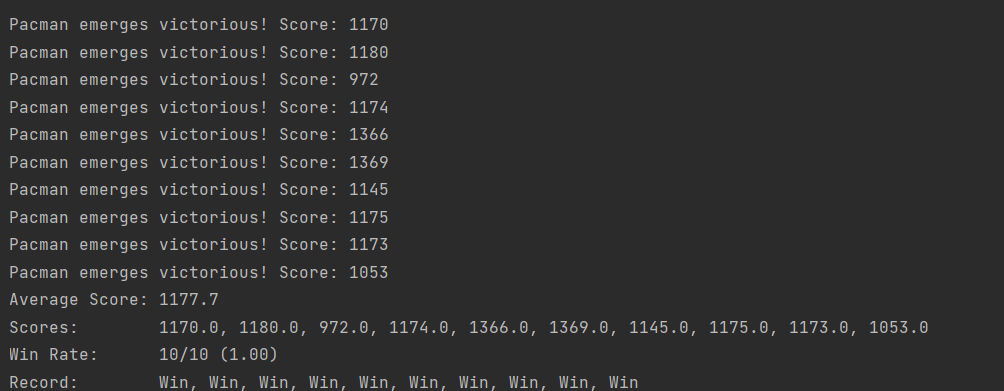
food权重为0.5时，score：1151.1



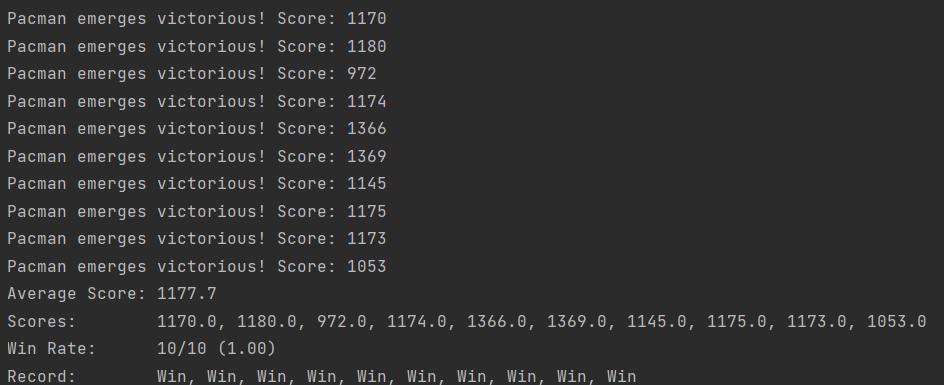
food权重为1时，score：1151.1



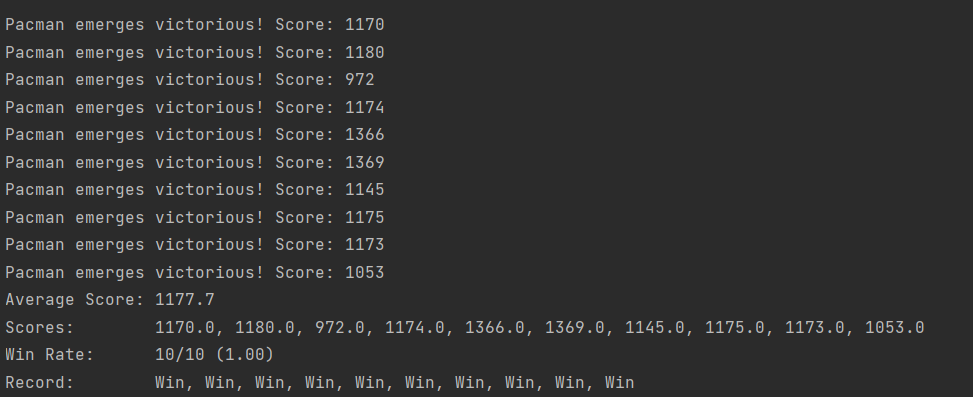
food权重为2时，score：1177.7



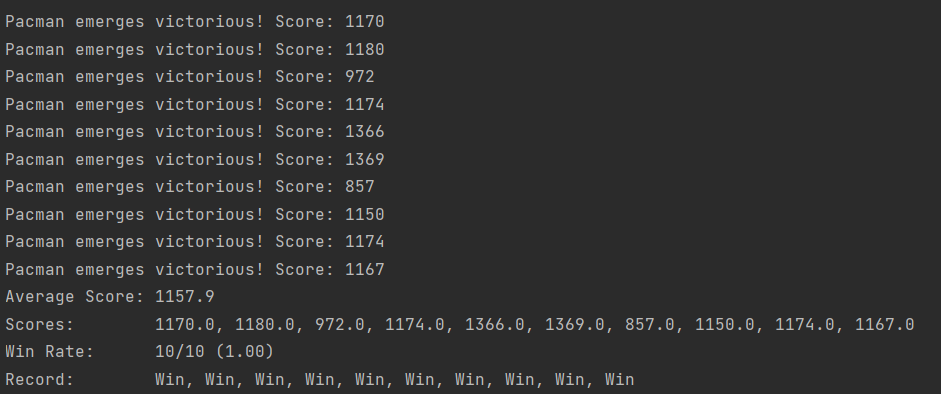
food权重为3时，score：1177.7



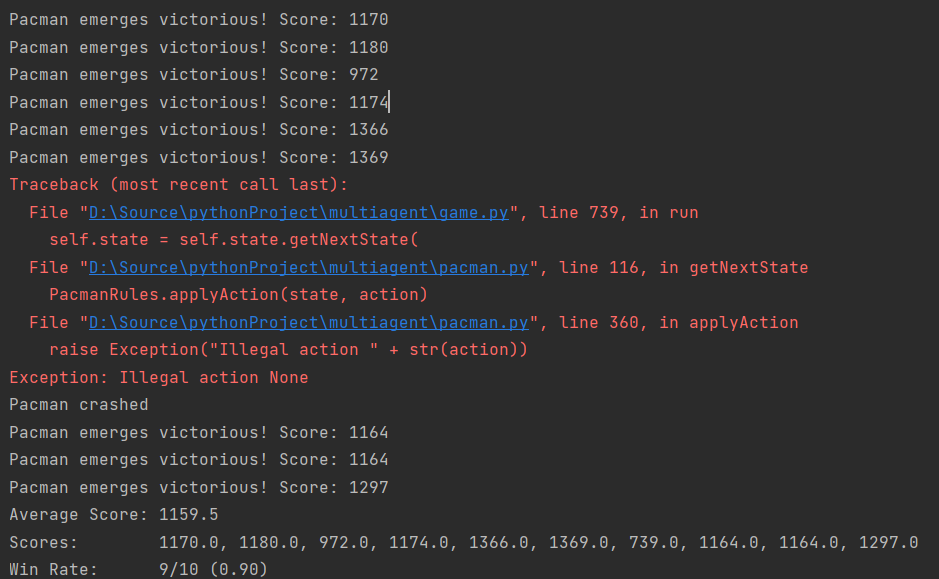
food权重为4时，score：1177.7



food权重为5时：



food权重为10时，开始出现失败的情景了



可以看出来，随着food权重的上升，pacman寻路算法的效果先提升再下降，取3~4时得到最佳，平均score为1177.7。

如果我们不止考虑最近的食物呢？假如最近的食物是个食物群，pacman能快速找到它们。

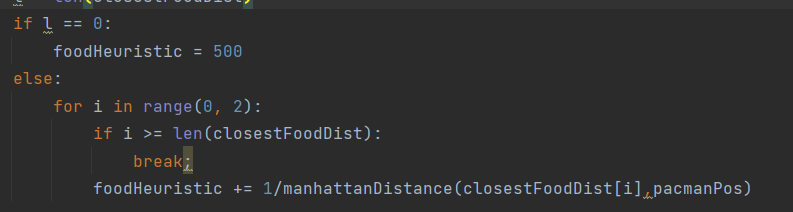
考虑两个食物的情况，测试结果如下：

每个food权重为0.2时，score：997.1

每个food权重为0.4时，score：1020.2

每个food权重为0.6时，score：1030.3

每个food权重为0.8时，出现失败



考虑五个食物的情况，测试结果如下：

每个food权重为0.1时，score：1027.7

每个food权重为0.2时，score：1186.6（显著提升）

每个food权重为0.3时，score：1100.3

每个food权重为0.4时，score：1150.2

每个food权重为0.5时，score：1152.6

每个food权重为0.6时，score：1177.3

每个food权重为0.7时，score：1146.6

每个food权重为1时，score：1127.9

每个food权重为1.5时，score：1110.3

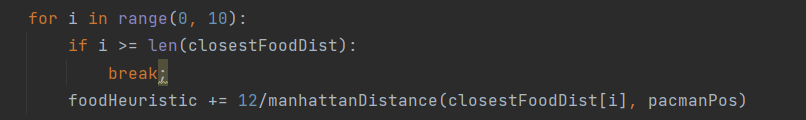
每个food权重为2时，score：1110.3

每个food权重为5时，score：1102.3

每个food权重为5时，score：1213.0（新的显著提升）

每个food权重为5时，score：974.8并发生了失败

考虑十个食物的情况测试结果：



每个food权重为0.2时，发生了失败

每个food权重为0.4时，score：1118.7

每个food权重为0.6时，score：1184.3

每个food权重为0.8时，score：1061.3

每个food权重为1时，score：1188.3

每个food权重为2时，score：1129.2

考虑所有食物的情况测试结果：

每个food权重为0.2时，score：1098.5发生了失败

每个food权重为0.4时，score：1098.3

每个food权重为0.6时，score：1061.7

每个food权重为0.8时，score：1061.3

每个food权重为1时，score：1188.3

每个food权重为2时，score：1223.9（新的提升）

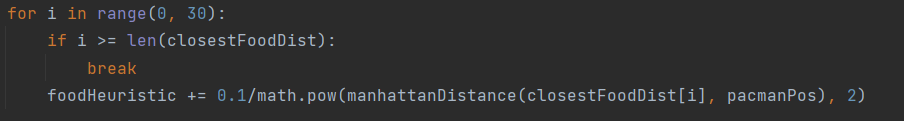
每个food权重为3时，score：1201.6

每个food权重为5时，score：1224.8（新的提升）

每个food权重为10时，score：1192.4

每个food权重为15时，发生错误

通过观察数据，我们找到了更多新的优质解法。考虑一种情况：一个pacman考虑了所有的食物，在他面前出现了两簇食物。他优先去吃食物多的那一簇，而这部分食物会慢慢消耗完，某一个时刻这些食物少于那簇没吃过的食物簇。根据算法，pacman将优先回头去吃那簇没吃过的，显然，这是不理智的。为了避免这种无意义的折返进食，考虑改进算法，我们增强距离的影响因素，这里我们利用平方倒数计算食物收益：



每个food权重为0.2时，score： 1035.9

每个food权重为0.4时，score：1052.7

每个food权重为0.6时，score：1061.5

每个food权重为0.8时，score：1061.3

每个food权重为1时，score：1112.7

每个food权重为2时，score：1213.6

每个food权重为3时，score：1095.5

每个food权重为5时，score：1129.1

每个food权重为10时，score：1193.6

每个food权重为15时，score：607.1

没有发现更好的解。

经过多次尝试，学生找到的最好的解是考虑所有食物且每个food的权重为5的情况，得到的平均score是1224.8。十道题偶然性仍然存在，所以结论不一定准确，但是可以看出考虑食物越多越可能发现优秀的解法，然后是选取一个合适的food权重是很重要的。