RICE UNIVERSITY®

Artificial Intelligence

Assignment II

Adversarial Search

September 17, 2014

Professor Devika Subramanian
Student:
Juarez Aires Sampaio Filho S01217136

Question 1

Question 2

- example where expectimax value of the root is greater than minimax value of the root:

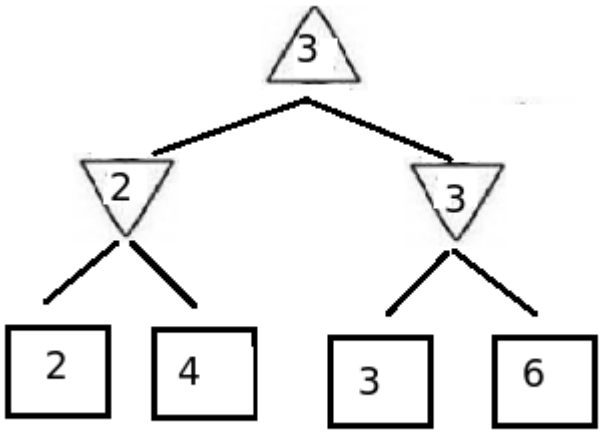

Figure 1: small game tree for minimax

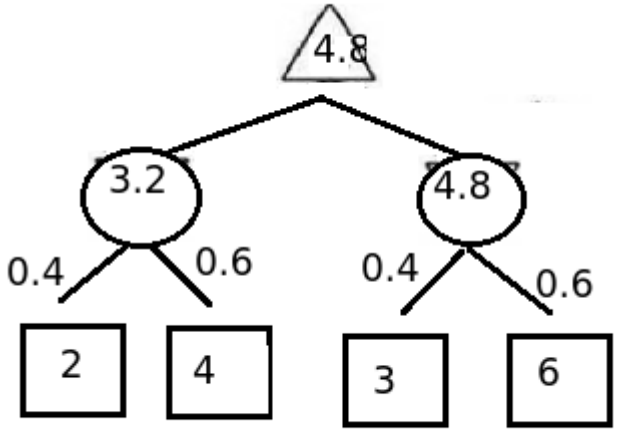

Figure 2: small game tree for expectimax

- This is not possible. The minimax value of the root will never be greater than its expectimax because the chances nodes will always have a value greater or equal to the value they would have it they were min agents. That is, every element of the of values the root can choose in case of minimax is less or equal the value in case of expectimax. Therefore, the root value cannot be greater.
- Player 1 should use minimax in case it is assumed that player 2 is a perfect minimal agent. That is, that player 2 is always choosing the best possible actions.
- Expectimax should be use whenever there is a possibility of player 2 making mistakes. That is, if player 2 is not optimal.
- Player 1 should run a expectimax where he considers himself a chance player and consider player 2 a min player. This way he can look for the path player 2 is going to lead the game and study weather there is a really good outcome node in this path that would never be achieved if played 2 thought player 1 was optimal. Player 1 then pretends to be random until this branch appears and then chose this awesome value. It is better than playing a simple minimax because in this new approach player 2 will allow the chances of tracking a way that could leave to a huge outcome for player 1 because player 1 is only random. The algorithm is as follows:

```
1.Run a minimax search and keep a record
of what the game would be.
2.Play as a chance player until Player 2 decides to take a
branch that wouldn't be taken on a regular minimax game.
3.When this happens, starts playing as a regular max player.
```

Question 3

Problem 1: Minimax

The base for this algorithm is the algorithm described on figure 5.3 of the textbook. We'll have to add to it the limited depth strategy discussed on session 5.4 and some extra details to deal with multiple adversaries. It is important to note that the ghosts are not playing against one another, they play only against pacman.

```
function MINIMAX_VALUE(s, deep)
    n = len(A) // A = [a0, a1,...,an]
    return MMValue(s, d*n)
```

```
function MMValue(s, d)
    if isEnd(s) or (d == 0)
        return Evaluation(s)
    else if Player(s) in [a1, a2, ..., an] //a min agent
        return min[MMValue(Result(s,a), d-1) for a in Actions(s)]
    else if Player(s) in [a0]//a max agent
        return max[MMValue(Result(s,a), d-1) for a in Actions(s)]
```

where Result(s,a), as usual, is the state resulted when aplying action a to the state s.

Questions

- **Why does pacman thrash around right next to a dot?**

  There might be other factors on the evaluation function that compenses the fact that pacman would score if he just grab the food. Maybe by approaching the food he'd also get closer tho a ghost and then the evaluation function tells him this is not a good state.

- **Why does pacman rush to the closest ghost in minimax search on trappedClassic?**

  This happens because pacman believes that the second ghost is going after him, so going in that direction is bad because this way he'll be trapped by two ghosts while going to the right he'll have to face only one. He forgets, however, that the ghosts are not optimal and there is a chance the second ghost is going down instead of going straight for him.

Problem 2: Alpha-beta pruning

Problem 3: Expectimax

The algorithm is based on the one presented on page 178 of the textbook. We add to it the fact that the probability of each ghost move is of 25% and that all min players are indeed chance player.

```
function EXPECTMINIMAX(s, deep)
    if isEnd(s) or (d == 0)
        return Evaluation(s)
    else if Player(s) in [a1, a2, ..., an] //a chance agent
        p = 1.0/len(Actions(s))
        return sum(p*EXPECTMINIMAX(Result(s,r), d-1) for r in Actions(s))
    else if Player(s) in [a0]//a max agent
        return max[EXPECTMINIMAX(Result(s,a), d-1) for a in Actions(s)]
```

Questions

- **Why does pacman's behavior in expectimax differ from the minimax case (i.e., why doesn't he head directly for the ghosts)?**

  Pacman nows recognize that there is a chance that the second ghost is going down instead of going straight to him. So he gambles and takes the left, trying he's luck and wishing for a good result.

Problem 4: Evaluation Function

My simple evaluation function is based on four features:

- total time of scared ghosts
- sum of distances* to ghosts
- sum of distances* to food
- scores

*by distances I mean manhatan distances. This was chosen to make the evaluation function cheap and it also represents the way pacman moves.

There are some details: if the ghosts are near packman, than he needs to stop whatever he is doing to run away. This is made by assigning a stupid negative value to that condition so he'll aways be running from this situation.

Finally, the weight associated to each feature were set based on intuition and trial and error

results:

I'd says the results obtained are impressive for such a simple approach. On a 50 times run on the smallClassic stage the winning ratio was of 43/50 (0.86) and the average score was 1290.86

IMPORTANT: after tests on other stages it is noted that the values of the constants weights assigned to each feature must be calibrated to each stage. This code will run fine(with the promised >0.8 winning ratio) on the smallClassic but it is not guaranteed to do good on every stage. In fact, for classicMinimax it looses every time.

Question 4

-
- No. The idea of pruning is eliminating nodes that you're sure will never be reached, but since every branch has a non zero probability of being taken you'll have to read the entire tree in order to calculate the expectimaxvalue.

-
-
-
-
- Does not make difference, since you won't be able to prun.