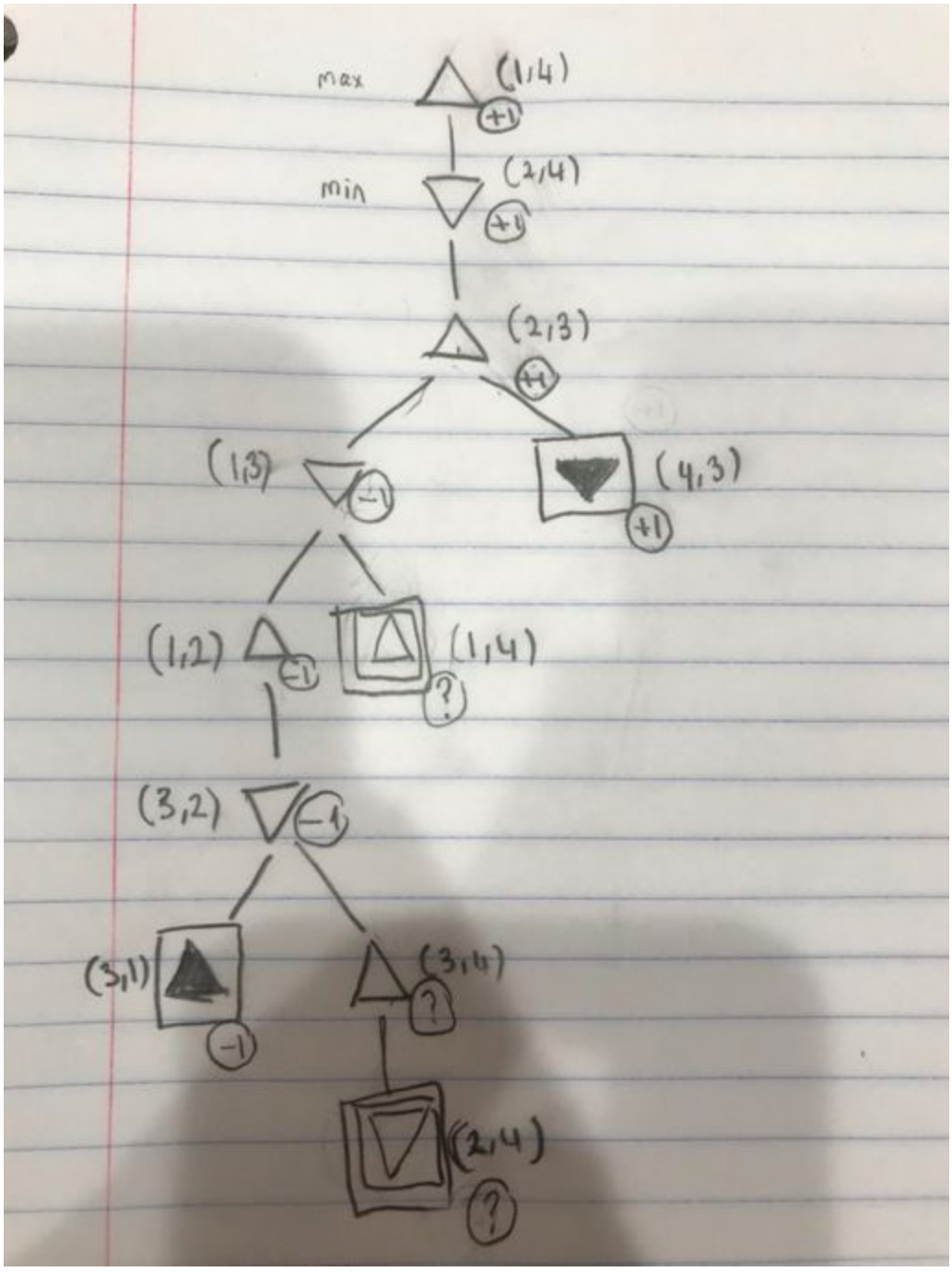
Assignment 2
Ege Ersü, Linus Shih

Question 1:
a)



b) Whenever we have a mystery node, we just backreference it by looking at a node with the same (sa, sb) that is present at an upper lever of the tree. So as an example, (1,4) used the be

a loop state, but we also had it as the root with a value of +1, so we just substituted the same value to the loop state as well. We assumed that ? has the bound: -1 <= ? <= 1.

c) We fix the algorithm by keeping a data structure of all previously visited nodes. When we encounter a mystery node with a ? value, we backreference to see if there are any previously visited states in the data structure with the same (sa, sb), and we substitute that value into our mystery node.
Suppose we have a game where we can win or lose by varying degrees instead of only absolute win/loss like in our 4-square game example problem. In this case, we would have a variety of possible utility values that represent wins or losses. This creates more uncertainty about what value each mystery node should adopt during backreferencing. This uncertainty reduces the reliability of the minimax algorithm and therefore does not guarantee an optimal solution for all games with loops. An example of such a game with varying degrees of win/loss is basketball, where there can be various ways of "winning", such as getting 2 points from a lay-up or getting 3 points from a free throw. In both cases, the "winning state" will be the same (sa, sb), but depending on the path taken to get to this winning state (layup vs free throw), we can win with a variety of possible point values (2 points vs 3 points).

d) The following proof assumes that both A and B are perfect players who play to win as soon as possible (i.e. no loops allowed).

1. In a base case with n = 3 squares, the only possible game progression goes as follows: (1, 3) → (2, 3) → (2, 1). B wins.
2. In a base case with n = 4 squares, the only possible game progression goes as follows: (1, 4) → (2, 4) → (2, 3) → (4, 3). A wins.
3. In a case with n = 5 squares, the only possible game progression goes as follows: (1,5) → (2,5) → (2, 4) → (3, 4) → (3, 2) → (4, 2) → (4, 1). B wins. Looking more closely at this series of moves, we can see that it follows a pattern. A moves first, B moves second, then both of them perform the same 2 maneuvers done in the base case of n = 3. Then, A moves once more, followed by B's last move which secures the win for B. This shows that n = 5 is just an extension of the base case with 2 more moves before and after the base case.
4. In a case with n = 6 squares, the only possible game progressing goes as follows: (1,6) → (2,6) → (2, 5) → (3, 5) → (3, 4) → (5, 4) → (5, 3) → (6, 3). A wins. Here we can see a similar pattern like with n = 5, except with the base case of n = 4. A moves first, B moves second, then both of them perform the same 3 maneuvers done in the base case of n = 4. Then, B moves once more, followed by A's last move which secures the win for A. This shows that n = 6 is just an extension of the base case with 2 more moves before and after the base case.

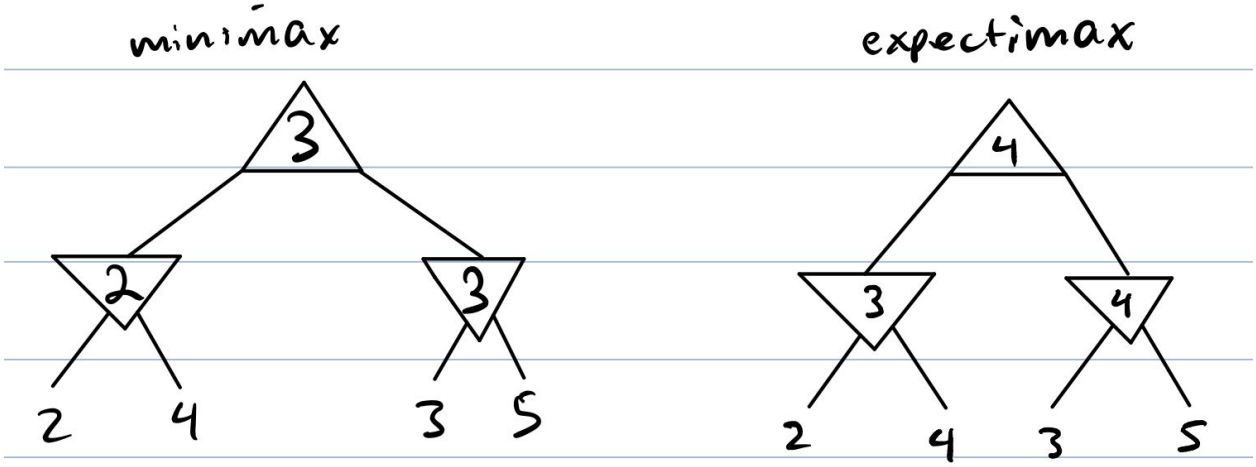
For a general case of any n > 2, by induction, we can observe the following phenomenon:

1. When n is odd, we add k = n - 3 moves before entering the base case of n = 3, and add k moves afterwards to get to the state where B wins.

2. When n is even, we similarly add k = n - 4 moves before entering the base case of n = 4, and k moves afterwards to get to the state where A wins.
3. In both cases, the winner of the base case is also the winner of the overall game because k is always even because (odd) - 3 = even, and (even) - 4 = even. Thus, we are always adding an even number of moves both before and after the base case, resulting in the base case winner being the overall winner.

Question 2:

a. Yes, it is possible to build a tree that has higher root values for expectimax than minimax. See the below example.



As we can see, minimax produces a root utility value of 3, while expectimax produces a value of 4, so expectimax produces the higher value at the root.

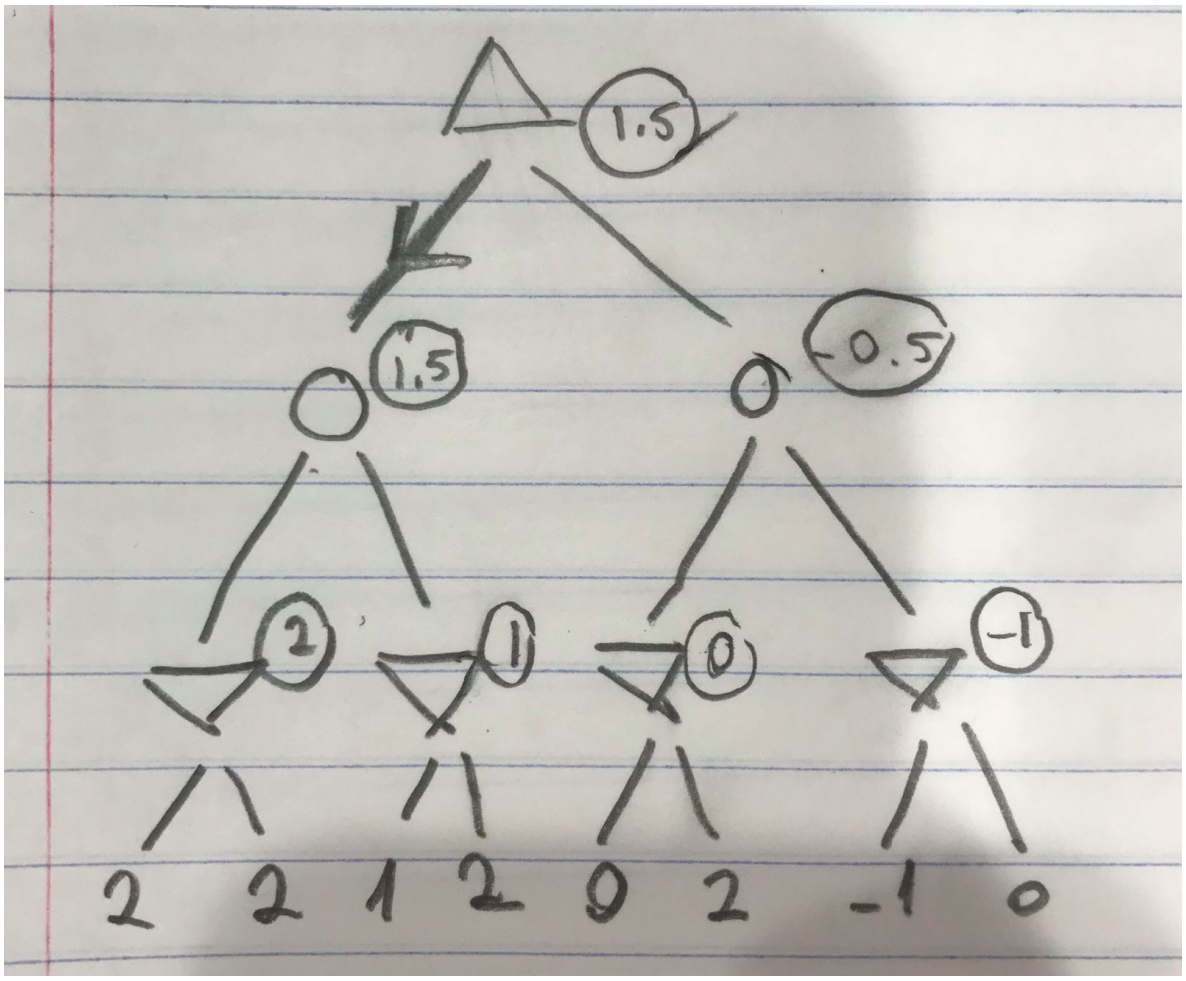
b. It's not possible for minimax search to have a larger value for the root node, compared to expectimax. Expectimax uses moves that are not always optimal, whereas minimax always picks the smaller node no matter what. The value picked by an expectimax layer will always be greater than or equal to the value picked by a min layer. That's why when you propagate the values up to the root, the root value for expectimax will also be greater than or equal to the root value for minimax. The only case where expectimax and minimax choose the same value, is when two children have the same value.
c. If player1 knows nothing about player2, it should use minimax under the assumption that player2 will move optimally.
d. If player1 has some knowledge about player2 and their outcome distribution, then player2 will use a single random sampling to determine their move. In this case, it's better to use expectimax.
e. First, player1 will generate a tree of what he thinks player2 will have. That tree will have the structure: chance - min - chance - … - min - leaves. Using this tree, player1 will calculate what the policy for the min nodes should be, and then store the calculated values.

   Then, player1 will generate a tree for itself to play the game with. This tree will have the structure: max - min - max - … - min - leaves. For the min nodes, player1 will retrieved the min node policy values taken and stored earlier from the first tree, substitute them

into the min nodes on the second tree, and then use them to calculate the final max node policy values on the second tree.

Question 4:

a) The agent will take the left action at the root node.



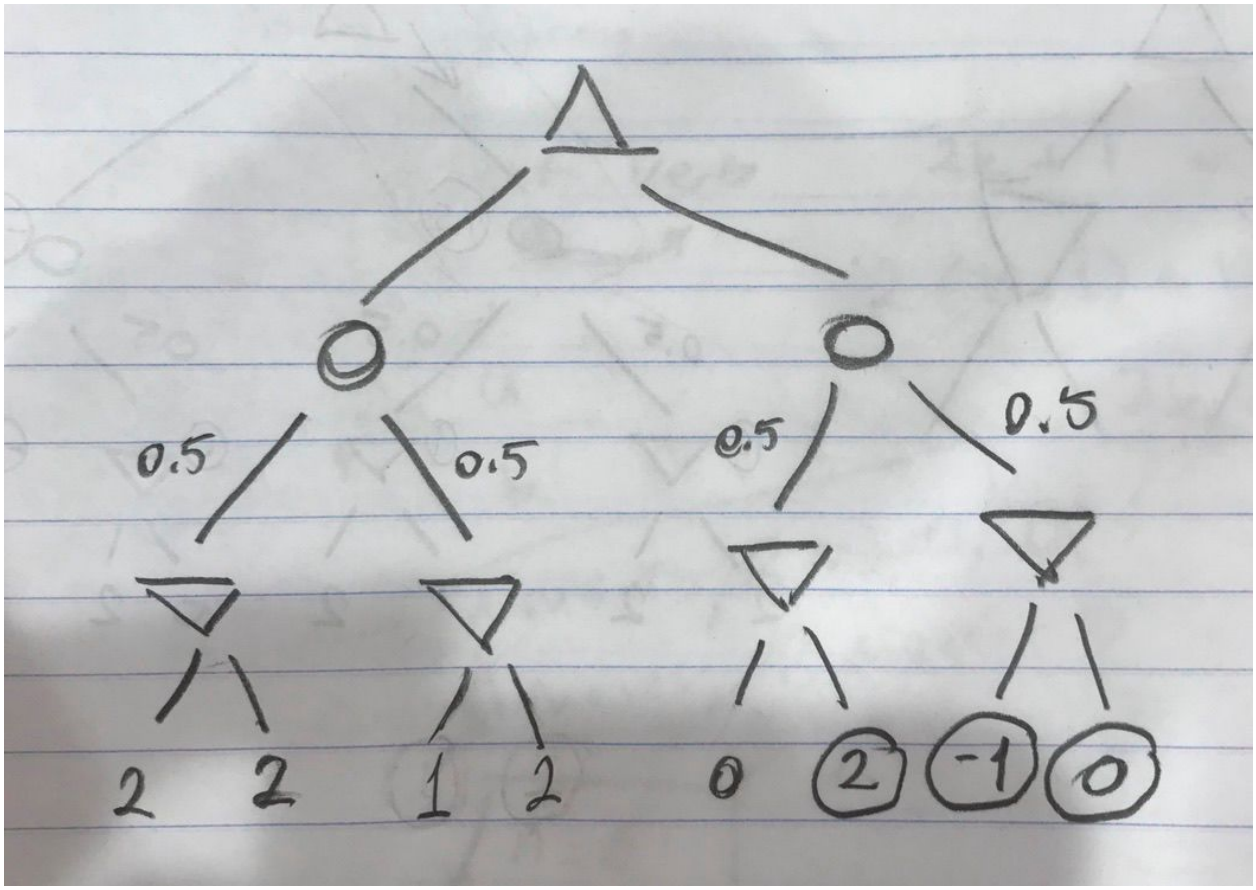
b) Seventh and eighth leaves should be evaluated in the first case. Let's say the value of the min node with children 7 and 8, has value m. Then the chance node will have the value m/2. Now the max root value will pick the greater amongst 1.5 and m/2. So if both leaf 7 and 8 are really huge numbers (like ∞), m/2 will be picked by the max node over 1.5. But also if one them is a really small negative number, m/2 will be smaller than 1.5. So depending on the values of leaf 7 and 8, the value of the root node changes, therefore the agent's optimal move changes.

But in the case where we have information about the seventh leaf having the value -1, we don't actually need to know the value of the eighth leaf. Whatever the value of the eighth leaf is, the min node will have value m/2, where m/2 is between -0.5 and -∞.

So the root node which is a max node, will never pick the value m/2, since its left child is a positive number whereas m is at most -0.5, independent of the value of the eighth leaf.

c) The first min node will pick up the value 2. Chance node will have the value = 0.5*2 + 0.5*m, where -2 <= m <= 2. Therefore, 0 <= (1 + m/2) <= 2.

d)



Write-ups

Minimax:

$$V_{opt}(s) = \begin{cases} \text{Evaluation}(s) & \text{if } \text{IsEnd}(s) \text{ or } d = d_{max} \\ \max_{a \in \text{Actions}(s)} V_{opt}(\text{successor}(s,a)) & \text{if player}(s) = \text{O} \\ \min_{a \in \text{Actions}(s)} V_{opt}(\text{successor}(s,a)) & \text{if player}(s) > \text{O} \end{cases}$$

Expectimax:

$$V_{opt,\pi}(s) = \begin{cases} \text{Evaluation}(s) & \text{if } \text{IsEnd}(s) \text{ or } d = d_{max} \\ \max_{a \in \text{Actions}(s)} V_{opt,\pi}(\text{successor}(s,a)) & \text{if player}(s) = \text{O} \\ \sum_{a \in \text{Actions}(s)} \boxed{\pi_{Ghost}(s,a)} \cdot V_{opt,\pi}(\text{successor}(s,a)) & \text{if player}(s) > \text{O} \end{cases}$$

$$\pi_{Ghost}(s,a) \rightarrow \frac{1}{\text{len}(\text{Actions}(s))}$$