Assignment 2

Task 1:

The file task1.pdf contains an illustration of the minimax tree for the given board state. The nodes represent possible game states and the lines connecting the nodes represent possible moves. Each node is labeled with it's utility value. Dashed lines are the moves that are chosen according to the minimax algorithm. If there is more than one dashed line per level, it just means that either move could be made, as they would both result in the same utility value on the tree.

Task 2:

a. The file task2a.pdf contains an illustration of the solution to this problem. Nodes that are crossed out have been pruned. Circled lines indicate the move that would be taken by the algorithm at each level. If more than one line is circled on a level, then that means that the algorithm can make either move, as both moves would result in the same utility value.
b. The file task2b.pdf contains an illustration of the solution to this problem. I would use this knowledge and implement it within my alpha-beta pruning algorithm. When it decides whether or not to prune a branch, it will check if the current value of the current node is equal to 10. If it is, and the current node is a MAX node, the algorithm will prune every other branch that comes from the current node, since the MAX player can't obtain a utility value higher than 10 and it won't choose any values lower than 10, making it pointless to explore any further. The minimizer function will remain unchanged.

Task 3:

This new algorithm will only need to account for the maximum value, since that is what is being used by the player. Instead of using code to find the minimum value, DeepGreenMove(S) will be used, since that is what is being used by the opponent.

Pseudocode:

```
Function MiniMaxPlus(S) returns action
inputs: S, current game state
return the a in Actions(S) maximizing DeepGreenMove(Result(a, S))

Function Max-Value(S) returns utility value V
if(Terminal-Test(S) then return Utility(S))
V <— -infinity
for a, S in Successors(S) do V <— Max(V, DeepGreenMove(S))
return V
```

——END OF PSEUDOCODE——

This code is very nearly identical to the normal Minimax algorithm. The major difference is the replacement of any Min-Value functions with DeepGreenMove(S).

This algorithm has one major guaranteed advantage over normal Minimax. In this situation, Minimax may not necessarily end up with the most optimal end state. This is all because of the

fact that normal Minimax has no knowledge of the fact that Deep Green is using a different algorithm. Instead, Minimax will make the assumption that the opponent is acting as a MIN player, and choose moves based on that assumption. The new algorithm accounts for the new Deep Green algorithm and will only choose moves based on how well it can beat the new algorithm.

As for speed/efficiency, it is hard to say, and depends on the complexity of the DeepGreenMove(S) function. If that function is more efficient than the standard Min-Value function (in Minimax), then this new algorithm could indeed be faster and more efficient than Minimax.

Task 4:
The file task4.pdf contains an illustration of the solution to this problem. Each non-terminal node has been labeled with it's utility value, which is decided based on the Expectiminimax algorithm. The circled line indicates the action that the algorithm will take. The root has not been labeled, as it is dependent on the outcome of the action that the algorithm takes. According to the algorithm, the root node will obtain a utility value of either 1000 or zero.

Task 5:
In the best-case scenario, the opponent uses an algorithm that also maximizes utility values. In this case, the best outcome for the MAX player would be a final value of 100. In the worst-case scenario, the opponent would just use a standard Min-Value function, and the game would play out like a normal Minimax-based game. In this worst-case scenario, the outcome for the MAX player would be a final value of 40.