Artificial Intelligence CSCI 350/761
Spring 2020 Assignment 2: Adversarial Search

Due Date: 11pm March 30, 2020

Assignment 1 has two parts: a Written section and a Coding section. The questions descriptions are uploaded in 2 documents on blackboard. The points will be formatted as follows: A/B where A is the points for the undergrad section (CSCI 350) and the B is the points for the grad section (CSCI 761 for the same question) while the coding section has more requirements for the graduate section.

WRITTEN SECTION (50 points)

Deliverable: **Upload to blackboard by deadline and submit print out in class March 31, 2020 .**

Please follow this format for submission. You may reproduce it in LaTeX or word if you wish

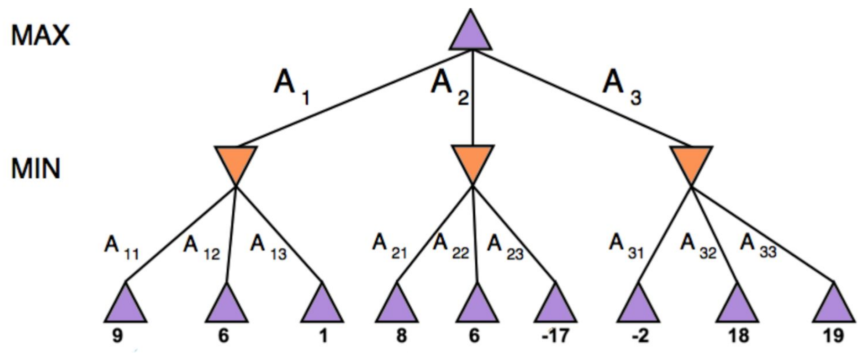
Name: Shadman Quazi

Course #: CSCI76100

Please justify all your answers to receive full credit (unless stated otherwise). There are 3 questions.

Question 1: Minimax and Alpha-Beta Pruning (20/20 points)

Consider the following search tree:



a. Using minimax, which of the three possible moves should MAX take at the root node? What is the value of Max at the root?

It should take A1, The value of the Max at the root node should be 1. The min from the previous nodes will be 1, -17, and -2.

b. Using minimax with alpha-beta pruning, compute the value of alpha and beta at each node (show your work on the tree). Which branches are pruned?

In this tree, branches A2 and A3 will be pruned, as the highest values they can possibly take will still be less than A1 branch, MAX will not care about their values.

Question 2: Iterative Deepening in Adversarial Search (10/10 points)

Provide at least two reasons why **Iterative Depth Search** (also called **Depth First Iterative Deepening DFID**) is useful in solving adversarial two-player games like chess.

Suggested reading: Section 7 of Depth-First Iterative Deepening Korf 1985, found in the References folder on the Blackboard.

Iterative Depth Search is a Depth First search that changes the max depth per iteration of the search. It shares advantages with DFS, namely linear space complexity. In regards to two-player games, assuming a minmax strategy like in Chess, Korf brings up some interesting applications, one of which involves a combination with heuristic search such as alpha-beta pruning. Korf says that the bounds for alpha and beta can be updated between depths, allowing for more immediate pruning of the game tree and a faster search time as a result. Another useful aspect of IDS in regards to two player games that Korf mentions is the search time, search at one ply time out, we can simply use the result from the ply above(i.e. Next shallowest ply) to make a decision instead.

Question 3: Checkers is solved! (20/20 points)

Read the Schaeffer “Checkers is solved” paper also in the References module. Describe in the space below how checkers were solved. Explain what methods were used and how. You can provide a general algorithm or specific explanations. Use your best judgement to provide the key elements.

The paper describes the method in three major portions:

- **Backwards search** - an endgame database was built and the win/loss/draw/result was recorded for each position by going working backwards, first enumerating and analyzing each position obtained after moving one piece, then analyzing each move obtained for moving two pieces, etc.
- **Proof-tree manager -** the tree manager hold the final copy of the proof, along with a list of positions of interest to be passed onto the solvers. The paper does not elaborate on what “positions of interest” are in this case.
- **Proof-tree solver -** The first part of the solver solver is describing an IDS approach with alpha-beta pruning, as it increases the ply it also updates the alpha beta bounds for what branches should be expanded on in the next move. Should the first part of the solver fail, it moves on to the Df-pn algorithm, a variant of the Proof Number search, which is a “best first” approach, which expands the moves that require the least effort to execute.