CS4725/CS6705 - Fall 2017
Lab Activity # 2

Name: _____

Student ID: _____

This lab activity will involve completing a small programming task and then answering a few questions.

Note: Your submissions (one Java file, submitted through Desire2Learn, and your responses to Questions 2 and 3, submitted on paper) are to be completed by 4:00 pm on **Wednesday, October 11, 2017**, but you are encouraged to finish as soon as possible.

1. This first exercise involves a short Java programming problem. The expectation is that you will have to write only about 10-15 lines of Java code in order to complete the task. However, you should spend some time testing your program to ensure that it works correctly.

   The goal is to fill in the missing lines in an implementation of the minimax algorithm with alpha-beta pruning.

   - On one of the lab machines, create a new directory for yourself and move to that directory. For example:

         mkdir AI-Lab2
         cd AI-Lab2

   - Copy the two Java files from the `/fcs/courses/cs4725/F17/minimax` directory into your directory.

         cp /fcs/courses/cs4725/F17/minimax/* .   (Note the space and period at the end)

   - Note that the `alphaBetaSearch` algorithm is slightly different from what is in the textbook and our lecture slides, in that it returns the **value** of the root node instead of returning a choice of action.

- In the file `Minimax.java`, add the necessary code to the `maxValue` and `minValue` methods in the places indicated. For your reference, the relevant pseudocode algorithms from our lecture slides are presented below.

> **function** MAX-VALUE(*state*, α, β) **returns** a *utility value*
>   **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
>   *v* ← -∞
>   **for each** *action* in ACTIONS(*state*) do
>   *v* ← MAX(*v*, MIN-VALUE(RESULT(*state*,*action*), α, β))
>   **if** *v* ≥ β **then return** *v*   // ignore/prune all other actions
>   α ← MAX(α, *v*)
>   **return** *v*

> **function** MIN-VALUE(*state*, α, β) **returns** a *utility value*
>   **if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
>   *v* ← +∞
>   **for each** *action* in ACTIONS(*state*) do
>   *v* ← MIN(*v*, MAX-VALUE(RESULT(*state*,*action*), α, β))
>   **if** *v* ≤ α **then return** *v*   // ignore/prune all other actions
>   β ← MIN(β, *v*)
>   **return** *v*

- Compile and run your program to test it.

  Note that the code is set up so that running `java Minimax` will test the program with a simple game tree that is provided, but running it with a parameter (*e.g.*, `java Minimax 1234`) will test it with a game tree that has random values in the leaf nodes (using the parameter as the seed for the random number generator).

  ```
  javac *.java      or      javac *.java
  java Minimax              java Minimax 1234
  ```

  Note: You can also experiment with your own game trees by modifying the code in the `buildAlphaBetaTreeExample` method. (Please do not change the code in the `buildRandomAlphaBetaTreeExample` method.)

- In the `buildRandomAlphaBetaTreeExample` method, you can set verbose to true if you want to print the random numbers as they are generated. Please change this back to false before submitting your file, however.

- Submit **only** your `Minimax.java` file on Desire2Learn through
 `Assessments → Assignments → Lab #2`.

2. Consider the partially filled game tree pictured below.
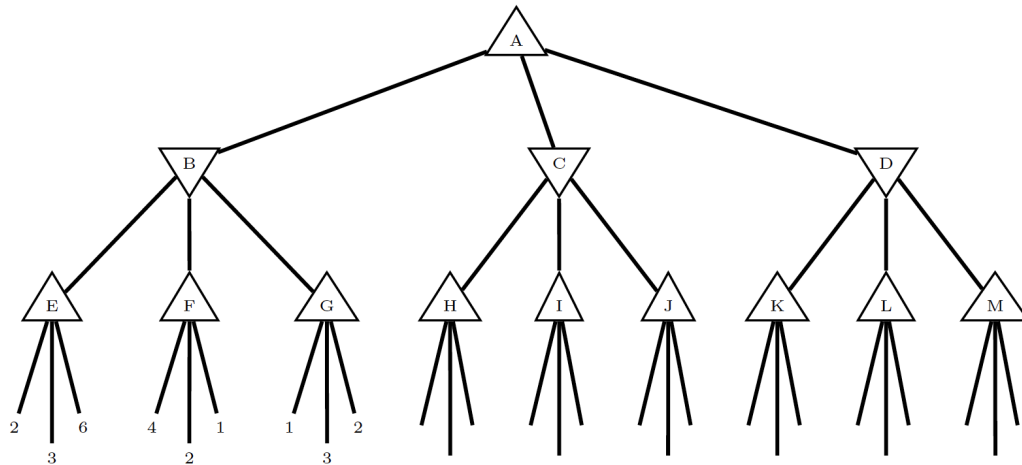
- Fill in example values for the remaining terminal nodes so that the alpha-beta pruning algorithm would **not** be able to do any pruning at all. In other words, the algorithm would have to visit every single terminal node in order for us to be certain that it would choose the best action for MAX at node A.

- (Optional) Once you think you have found appropriate values for the terminal nodes, you can try entering your values in the `buildAlphaBetaTreeExample` method and then compile and run your program, to confirm that no pruning will take place.

3. When we talked in class about pruning in game trees, we were focusing on trees that had only MAX and MIN nodes. This question considers the possibility of pruning in games that also have chance nodes.

   Consider the game tree below, which includes chance nodes B and C. Suppose that we know in advance that all terminal nodes will have values between -20 and 20, inclusive.
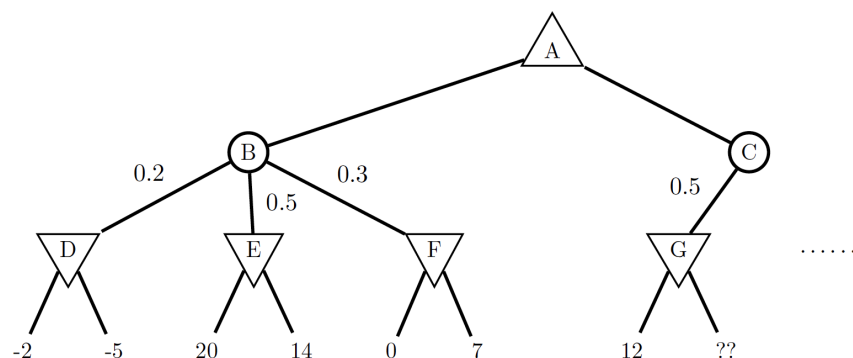
   We are exploring the nodes in this game tree in a depth-first order. Suppose that we have already explored the entire left side of the tree (node B and its descendants). On the right side, we have seen the first child of G and we know that 0.5 is the probability associated with the edge from C to G.

   (a) When we explore the second child of node G, what **range** of values could it have that would allow us to conclude for certain that there is no need to explore any other children that node C might have?

   Explain the reasoning behind your answer, showing any necessary calculations.

   (b) If the probability on the edge from C to G were changed from 0.5 to 0.4, how would that change your answer to part (a)?