EECS 391: Introduction to AI (Spring 2016) Programming Assignment 2 (Max Points: 100)

**Assigned Tuesday February 16, due midnight Thursday March 3. Turn in your code using blackboard. Please comment your code extensively so we can understand it, and use sensible variable names. If two people worked on a submission, please include both your names and IDs in a README. You may only turn in an assignment as a pair if both of you have contributed equally to it.**

In this assignment, you will implement the alpha-beta algorithm for playing two player games to solve some SEPIA scenarios.

In the zipfile provided are three maps and config files in data/, an opponent agent in archer_agent/ and skeleton files for your agent in src/. The archer agent is part of the edu.cwru.sepia.agent package while the Minimax agent is part of the edu.cwru.sepia.agent.minimax package (place them suitably in your class hierarchy). The maps have two Footmen belonging to player 0 and one or two Archers belonging to player 1. Footmen are melee units and have to be adjacent to another unit to attack it, and they have lots of health. Archers are ranged units that attack at a distance. They do lots of damage but have little health. In these scenarios, your agent will control the Footmen while the provided agent, ArcherAgent, will control the Archers. The scenario will end when all the units belonging to one player are killed. So your goal is to write an agent that will quickly use the Footmen to destroy the Archers. However, these Archers will react to the Footmen and try to outmaneuver them and kill them if they can. You will use game trees to figure out what your Footmen should do.

Your agent should take one parameter as input. This is an integer that specifies the depth of the game tree in plys to look ahead. This is specified in the configuration XML file as the "Argument" parameter under the Minimax agent. At each level, the possible moves are the *joint* moves of both Footmen, and the *joint* moves of the Archers (if more than one). For this assignment, assume that the only possible actions of each Footman are to move up, down, left, right and attack if next to the Archer(s). The Archers have the same set of actions: move up, down, left right and attack (which means they stay where they are). Thus when your agent is playing, there are 16 *joint* actions for the two Footmen you control (if you are next to an Archer, you also have the Attack action). When ArcherAgent is playing, it has either 5 or 25 (*joint*) actions depending on whether there are one or two Archers. You can see that even for this simple setting, the game tree is very large!

Implement alpha-beta search to search the game tree up to the specified depth. Use linear evaluation functions to estimate the utilities of the states at the leaves of the game tree. To get these, you will need state features; use whatever state features you can think of that you think

correlate with the goodness of a state. A state should have high utility if it is likely you will shortly trap and kill the Archer(s) from that state.

Since the game tree is very large, the order of node expansion is critical. Use heuristics to determine good node orderings. For example, at a Footman level in the game tree, actions that move the Footmen *away* from the Archers are almost always guaranteed to have low utility and so should be expanded last. If adjacent to an Archer, a Footman should always attack. Similarly, if the Archer(s) is (are) very far away from your Footmen, they will not run but shoot your Footmen, so expand that action first, and so forth.

We will award up to 10 bonus points for well written code that is able to quickly search a large number of plies relative to the rest of the class (e.g. if you are in the top three runtimes to finish a scenario with a fixed large number of plies). Note that we will test your code with other maps than the ones provided with this assignment.

## Code and Data Structures

In the skeleton agent files, MinimaxAlphaBeta is the main class. It includes the main alphaBetaSearch method and a node reordering method (orderChildrenWithHeuristics). These are the methods you will fill in. A helper class, GameState, is provided to track SEPIA's state, which can then be used to compute the utility (getUtility) and to find the possible results after taking an action from a state (getChildren). You will fill this in as well. You should not modify the GameStateChild class. It just pairs an action map with a GameState.

The code is structured in this way so that your alphaBetaSearch method can be implemented abstractly (i.e. it will not need to contain any SEPIA specific code). The SEPIA related code should reside in GameState. You can add fields and functions to GameState as needed (add comments to explain what they are doing).

## What to turn in

Prepare a ZIP file with MinimaxAlphaBeta.java, GameState.java and a README (do NOT include any class files, or any SEPIA code). Include a README with your name(s) and ID(s) and any other comments you have. Name your file as "yournetworkID(s)_PA2.zip" and use Blackboard to submit it. If you worked as a pair, ensure that both your IDs appear in your submission. Only one submission is required if you worked as a pair.