EECS 391: Introduction to AI (Spring 2012) Programming Assignment 2 (Max Points: 100)

Assigned Thursday February 9, due midnight Tuesday February 28. 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.

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

In the zipfile provided are two maps and two agents. 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, they only have 1 health). In these scenarios, your agent will control the Footmen while one of the provided agents, AdversaryAgent, will control the Archers. The other agent provided, GameAgent, is for demonstration purposes only and can be used to control the Footmen to get a sense for how the scenarios play. 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 need to 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. 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 AdversaryAgent 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.

Code and Data Structures

You can create either one or two agents corresponding to the two scenarios given. Call your agents ABAgent1 and ABAgent2 (if you just have one, call it ABAgent). Remember to have SimpleRTS.jar in the classpath when you compile these. To implement the search, it is fine to use basic data structures, such as arrays and lists, to keep track of what you need. Since the game tree is large, you will need to be somewhat efficient in writing your code. The more efficient you are, the deeper you will be able to search! We may give bonus points for efficient code that is able to search large trees quickly.

What to turn in

Prepare a ZIP file with your agent code (do not include any class files). Include a README with your name(s) and ID(s) and any other comments you have, such as special compilation instructions if any. Name your file as “yourname_PA2.zip” and use Blackboard to submit it.