Alexandria University
Faculty of Engineering
Computer and Systems Engineering
Department

Artificial Intelligence
Assigned: Saturday, October 27, 2018
Due: Saturday, November 17, 2018

Assignment 2 - Simple agent and environment simulator for game of RISK

1 Overview

In this assignment you are asked to implement a simple environment simulator that generates instances of a search problem, runs agent programs, and evaluates their performance according to a simple performance measure. The search problem we will use is a simplified and abstract version of the board game RISK. Before reading on to the rest of the assignments, be sure you understand the rules of the full version of the game. The following discussion will discuss how the rules of the simplified, abstract game differ from the full version of the game.

2 Application

In the abstract version of the game, the board is just an undirected graph, where each territory (country) is a vertex, and a graph edge signifies that territories have a common border (and thus a country represented by one vertex can attack the country represented by the other vertex). In addition, the graph is partitioned into several (connected) subgraphs, and each partition cell represents a continent and has its respective bonus (received for holding all of it).
Your program should be able to read the graph specification from a file, in a format such as:
V 4 ; number of vertices
E 4 ; number of edges
(1 2) ; edges
(2 3)
(3 4)
(1 3)
P 2 ; number of partition cells
5 1 2 ; value of the 1st partition cell, and its members
3 3 4 ; value of the 2nd partition cell, and its members
It is easy to specify the RISK map in the board game, if so desired...

This abstract version of the game makes it more general, but not necessarily more complicated. The following is a list of simplifications:

1. RISK is a multi-player game, but we will assume that only 2 players are in the game in the simplified version.
2. RISK has cards that can be cached in for armies - we will have no cards in our version. Instead of cards, a player which conquers at least one vertex (territory), receives an additional bonus of 2 armies at the next turn.

3. In this initial exercise, the battles are deterministic. Denote A(v) the number of armies in vertex v. Vertex v can attack vertex u only if there is an edge between these vertices, and $A(v) - A(u) > 1$. As a result of the battle, each opposing player loses A(u) armies, and the attacking player must move at least 1 army to u, but must leave at least 1 army in v.

4. Initial placement of armies is determined as part of the input, and not by the agent.

5. When getting the armies at the beginning of a turn, they must all be placed on the same vertex.

6. For this initial exercise, the enemy is completely passive, i.e. it never attacks, and always places all its additional armies on the vertex that has the fewest armies, breaking ties by favoring the lowest-numbered vertex.

7. Initially, a turn will consist of: placing the bonus armies at the beginning of the turn, and doing at most one attack (including moving armies into the captured territort). There will be no fortifying step, for simplicity.

The goal of the game is to conquer the world in the smallest number of turns.

3 Implementation

3.1 Phase 1

Initially you will implement the environment simulator, and several simple (non-AI) agents, as follows:

1. A human agent, i.e. read the next move from the user.

2. A completely passive agent, as discussed above.

3. An aggressive agent, that always places all its bonus armies on the vertex with the most armies, and greedily attempts to attack so as to cause the most damage - i.e. to prevent its opponent getting a continent bonus (the largest possible).

4. A nearly pacifist agent, that places its armies like the completely passive agent, then conquers only one vertex (if it can), such that it loses as few armies as possible.

In the above simple agents, ties are broken by selecting the smallest numbered vertex among all that are equally good. At this stage, you can try running agents against each other, or play human against agent. Allow user selection of which agents will be in a game run.

3.2 Phase 2

Later on, you will be implementing an inteligent agent that plays against a completely passive agent, and attempts to win in as few turns as possible. There will be three types of agents, each employing a different search algorithm, defined below. All the algorithms will use a heuristic evaluation function of your choice.

1. A greedy agent, that picks the move with best immediate heuristic value.
2. An agent using A* search, with the same heuristic.
3. An agent using real-time A*.

The performance measure will be composed of two parts: L, the number of turns it takes the agent to win the game, and T, the number of search expansion steps performed by the search algorithm. The performance of an agent will be:
$P = f * L + T$
Clearly, a better agent will have P as small as possible. The parameter f is a weight constant. You should compare the performance of the three agents against the completely passive opponent for the following values of f: 1, 100, 10000. Note that the higher the f parameter, the more important it is to expend computational resources in order to get a faster win!

4 Deliverables

1. Your well commented code.
2. A report showing your work and results.
3. Good GUI.

Also the report should contain the data structure used (if any) and algorithms, Assumptions and details you find them necessary to be clarified, Any extra work and Sample runs. You should show your algorithm and how it operate.

5 Further Notes

You may use Java , Python or C++ for your implementation.
Copied assignments will be severely penalized.
You can work in groups of 3.

6 References

https://www.cs.bgu.ac.il/ shimony/AI2004/AIass1.html
http://web.mit.edu/sp.268/www/risk.pdf

Good Luck