Introduction to Artificial Intelligence

Assignment 2

Cooperating and adversarial agents in the Hurricane Evacuation Problem

In the second exercise you will be using the environment simulator from the first assignment, the Hurricane Evacuation problem, as a platform for implementing **intelligent** cooperating and adversarial agents. The environment is the same as before, except that now we will assume two evacuation agents (perhaps they are employed at competing companies) who seek to evacuate as many people as possible. We will examine settings ranging from cooperative to adversarial.

Game Environment

As before, the environment consists of an undirected weighted graph. An agent can apply 2 types of action: **traverse** and **terminate**. Semantics of the action are as in assignment 1, repeated below. The **terminate** action ends effective actions in the game for this agent. The traverse action always succeeds if the edge (road) is unblocked and the time limit has not passed. and fails otherwise. A failed action behaves like terminate. Although this is a bit unnatural, we will assume for simplicity that the agents take turns at every time unit, rather than moving truly in parallel. The game ends when both agents are terminated.

Implementation Steps

The simulator should query the user about the parameters, the type of game (see below) as well as other initialization parameters.

After the above initialization, the simulator should run each agent in turn, performing the actions returned by the agents, and update the world accordingly. Additionally, the simulator should be capable of displaying the world status after each step, with the appropriate state of the agents and their score. The agent individual Each agent program (a function) works as follows. The agent is called by the simulator, together with a set of observations. The agent returns a move to be carried out in the current world state. The agent is allowed to keep an internal state if needed. In this assignment, the agents can observe the entire state of the world. You should support the following types of games:

1. Adversarial (zero sum game): each agent aims to maximize its own individual score (number of people saved) minus the opposing agent's score. Here you should implement an "optimal" agent, using mini-max, with alpha-beta pruning.
2. A semi-cooperative game: each agent tries to maximize its own individual score. The agent disregards the other agent score, except that ties are broken **cooperatively**.
3. A **fully cooperative** game: both agents aim to maximize the sum of individual scores.

Since the game tree will usually be too big to reach terminal positions in the search, you should also implement a cutoff, and a heuristic static evaluation function for each game. You may use the same heuristic for all games, if you think this is justified.

Deliverables

The program and code sent to the grader, by e-mail or otherwise as specified by the grader, a printout of the code and results. You need to show example scenarios where the optimal behavior differs for the 3 kinds of games (you will need to make the example scenarios and/or **very** small time limits in order to be able to reach terminal states in the search). A description of your heuristic evaluation functions and their rationale. Set up a time for frontal grading checking of the delivered assignment, in which both members of each team must demonstrate at least **some** familiarity with their program.

Due date: TBA