Introduction to AI - assignment 2

Oded Yechiel   Matan Rusanovsky

27/11/18

1 Introduction

In this assignment we had to use the hurricane simulator from assignment 1, but operate several agents in parallel. The agents have 3 different options of operation:

1. Adversarial - where the agent must get more than the opponent.

2. Semi-cooperative - where the agent must maximize their score.

3. Cooperative - where the agent has to cooperate with the other agent to maximize the global score.

2 Base agent − Assignment 1 recap

In assignment 1 a basic agent was developed, in which the method BESTACTION (In actual code you should search for `choose_next_action`) had to be implemented by any other inheriting agent. The basic agent had all of the information required by the agent to make decisions, such as the,

- current vertex

- number of people in the vehicle

- number of people saved

3 Game tree agent

The three agents that are implemented in this assignment are all based on the game-tree-agent.

Algorithm 1 Game tree best action

1: **procedure** BESTACTION(State, Simulator)
2:     **if** ISTERMINAL(Simulator) **then**
3:         **return** NOOP()
4:     **end if**
5:     **return** RECURSIVETREE(Simulator, 0, IsZeroSumGame)
6: **end procedure**

Algorithm 2 Terminal position

1: **procedure** IsTerminal(Simulator)
2:     **if** Simulator.Time $\geq$ Simulator.Deadline **then**
3:         **return** True
4:     **else if** Simulator.PeopleInTowns $== 0$ and $\forall agent \quad agent.carrying == 0$ **then**
5:         **return** True
6:     **else**
7:         **return** False
8:     **end if**
9: **end procedure**

The BestAction function is described in Algorithm **??** and is fairly simple. It just checks if we are at a terminal state, and if so return a NoOp result, otherwise it starts the the recursive function described in Algorithm **??**.

The terminal state, shown in Algorithm 2, is a boolean function. It returns true if the time surpassed the deadline, or if there are no more people to save.

The recursive tree function in Algorithm **??** receives a simulator, that hold all of the data and used to emulate each action, depth value representing the depth of tree, and a boolean stating whether it is a zeros sum game.

The algorithm starts by checking for terminal states and checks if the depth is maxed out for cutoff purposes. For terminal states the actual value, i.e., number of people saved, of each agent is returned. Maxing out depth returns a heuristic of how many people each agent may save, disregarding the other agent and assuming each town with people is the only town with people. This same heuristic was used in assignment 1.

The function continues, if not returned, to a for loop (line 10) that branches every possible action of an agent, and calls recursively to itself with an emulated simulator where its the next agent's turn, and depth is increased by 1.

The result of this action is compared with all of the other actions and the best action's value is returned if the depth is larger than 0, otherwise the best action itself. For each type of agent (adversarial, cooperative, semi-cooperative) the comparison function is different and will be elaborated in the following section.

In case the game is adversarial, or a zero sum game, an alpha-beta cutoff is also compared to try and trim branches off.

2

Algorithm 3 Game tree recursive tree expansion
___

1: **procedure** RECURSIVETREE(Simulator, Depth, IsZeroSumGame)
2:     **if** ISTERMINAL(Simulator) **then**
3:         **return** VALUE(Simulator)
4:     **end if**
5:     **if** Depth reached MAX_DEPTH: **then**
6:         **return** HEURISTIC(Simulator)
7:     **end if**
8:     BestValue ⟵ $[-inf, -inf]$
9:     BestMove ⟵ -1
10:     **for** each Action $\in$ possible actions of current player **do**
11:         Simulator.EMULATEACTION(Action)
12:         NewValue $\leftarrow$ RECURSIVETREE(Simulator, Depth+1, IsZeroSumGame)
13:         **if** ISBETTERACTION(NewValue, BestValue) **then**
14:             BestValue ⟵ NewValue
15:             BestMove ⟵ Action
16:         **end if**
17:         **if** IsZeroSumGame is True **then**
18:         **end if**
19:     **end for**
20:     **if** Depth == 0 **then**
        return BestMove
21:     **else**
        return BestValue
22:     **end if**
23: **end procedure**
___

3.1   Adversarial agent

3.2   semi-cooperative agent

3.3   Cooperative agent