Multi-Agent Search

Reading

To prepare, read in Chapter 5, *Adversarial Search*:

- 5.1, *Games*
- 5.2, *Optimal Decisions in Games*
- 5.3, *Alpha-Beta Pruning*
- 5.4, *Imperfect Real-Time Decisions*
- 5.5, *Stochastic Games*
- 5.9, *Summary*

Additionally/alternately, you may review the UC Berkeley material on this topic:

- adversarial search lecture slides
- expectimax search lecture slides

Assignment

Complete the multi-agent search problem set, http://ai.berkeley.edu/multiagent.html.

Get the starter files from Bottlenose and submit your work here: https://grader.cs.uml.edu/assignments/611

Do not search for solutions on the web.
We will use MOSS to verify that each submission contains only original work.

Pair programming allowed with acknowledgment.
Students are allowed to work in pairs on this assignment.
You must credit your partner at the point of Bottlenose submission.
There will be a 10% deduction if you forget to acknowledge your partner.

Pseudocode

Please see diagrams below for pseudocode useful for this problem set.
Right-click and view in new window or tab to enlarge each diagram.

Minimax Implementation

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, min-value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, max-value(successor))
    return v
```

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$

Minimax Implementation (Dispatch)

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Alpha-Beta Implementation

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Expectimax Pseudocode

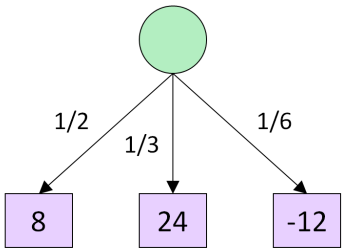
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)
```

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Expectimax Pseudocode

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```



v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10