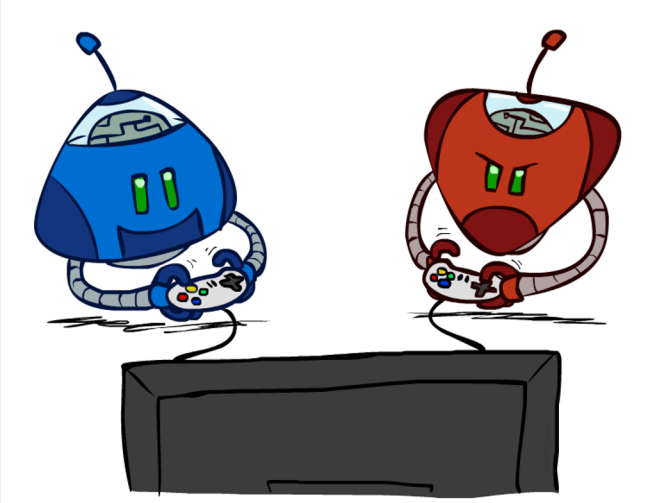
CS440 – Adversarial Search



Purpose: Consider two players competing against each other. A player can no longer can just do a search for its next action since it is now dependent upon what actions its adversary is going to do as well.

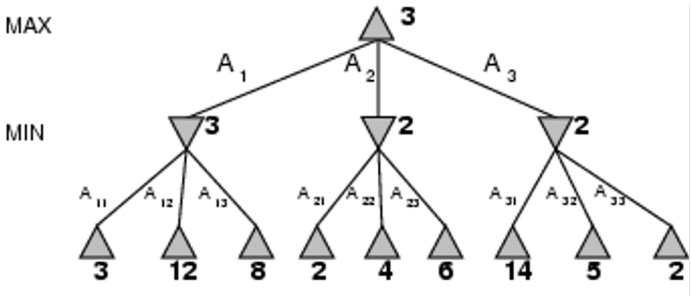
The purpose of this module is to present algorithms for solving this adversarial search which considers the possible actions of the adversary.

Knowledge: This module will help you become familiar with the following content knowledge:

- The minimax algorithm
- Alpha-beta pruning of the search tree
- The expectimax algorithm

Activity1 - Game Trees and Minimax:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 40 minutes.

1. Consider two players, Max and Min. Max is trying to maximize the score and Min is trying to minimize the score. This tree shows the possible outcomes:



   At the beginning of the game Max has three possible moves: A1, A2, A3. The tree then shows the results of Min's possible moves. Which move should Max make? Why?

2. Here is the minimax algorithm:

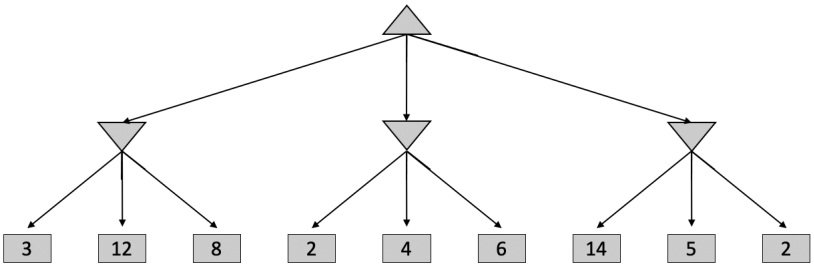
```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)

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
```

   Trace through the algorithm with the previous search tree.

3. Consider the following tree:



Do we really need to examine all the nodes? Why do we not have to examine nodes 4 and 6?

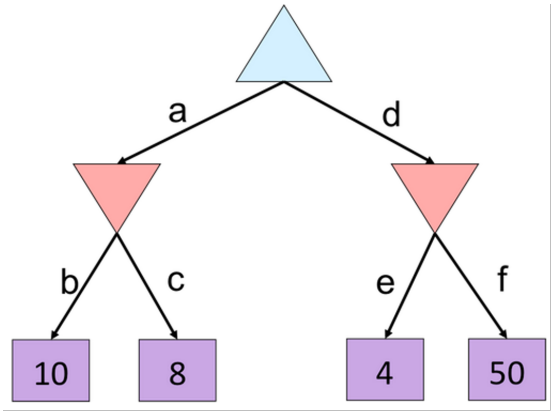
4. The method for pruning the tree is called alpha-beta pruning:

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

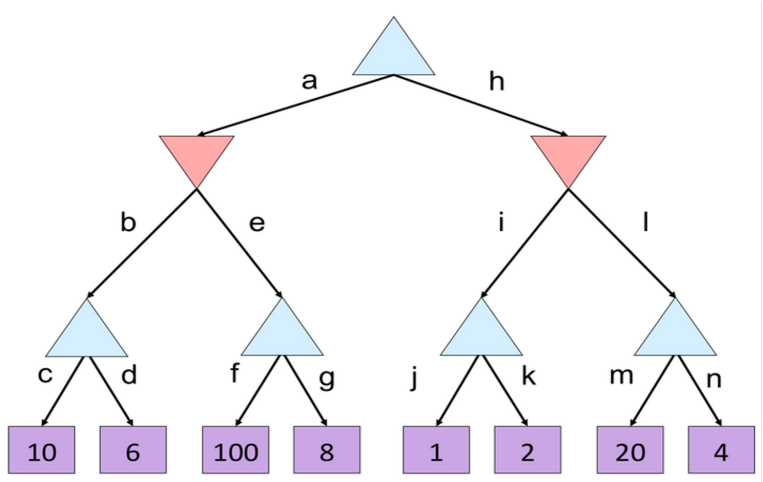
```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
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
```

What gets pruned in the following tree?



What gets pruned in this one?

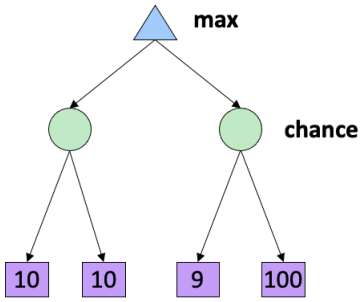


5. Even with pruning, there are resource limits for minimax. For chess, as an example, the branching factor is about 35 in the tree and the maximum depth is about 100 so we would not be able to evaluate the entire tree. Instead we would evaluate to a specified cut-off and use an evaluation function to estimate the expected utility of that game position.

   Give a possible evaluation function for tic-tac-toe.

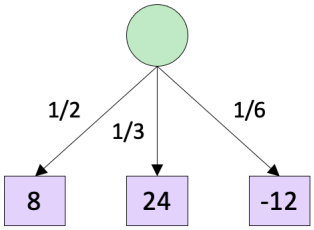
Activity2 - Expectimax:
With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 15 minutes.

1. . We can use a similar algorithm if, instead of having an adversary we have an uncertain outcome. Consider the following tree where we have "chance" nodes, instead of the min nodes that we saw previously, which indicate multiple outcomes that can occur with some probability. For the chance nodes we can calculate the *expected utility* by computing a weighted average.



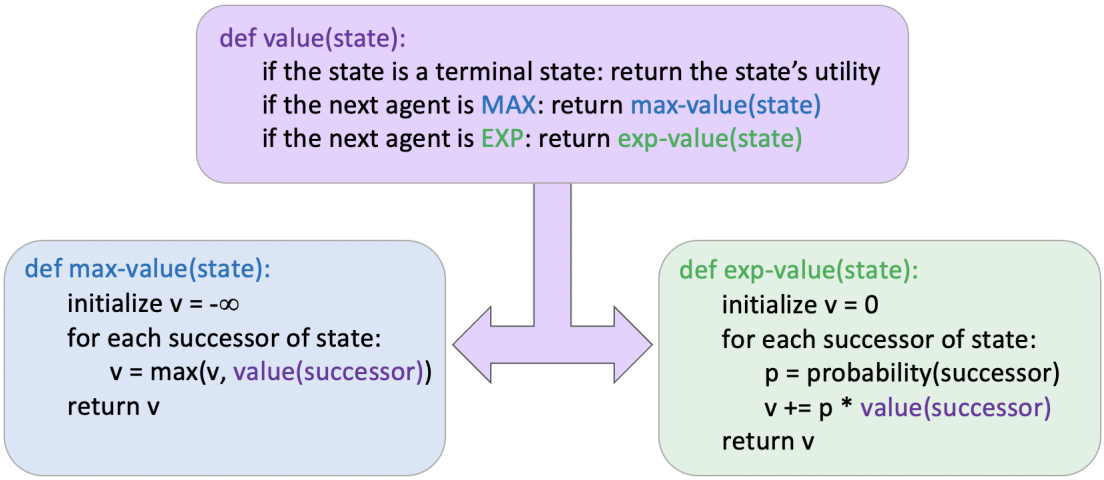
What move should max take if there is even probability in the chance nodes (each choice is equally likely)? Why?

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



What would be the expected value of the node above?

2. Here is the expectimax algorithm:



Trace through the algorithm with the first tree in the previous question.