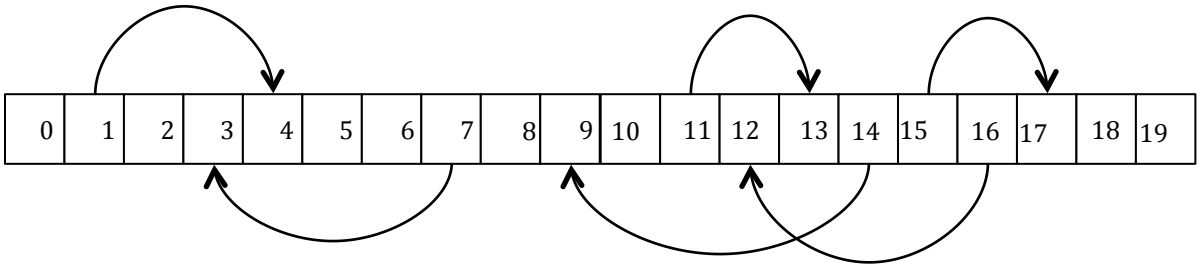
CS 3600 – Introduction to Artificial Intelligence

Adversarial Games with Dice

Consider the following board game, based loosely on chutes and ladders. Two players race from position 0 to position 19. The first person to land exactly on position 19 wins. If you land on any position with an arc emanating, you automatically leap to the position at which the arc terminates. For example, if you stop on position 1, you are teleported to position 4.



On each turn, a player may perform one of the following actions:

- Move one position forward or backward.
- Roll one 4-sided die (1D4) and move the resultant number of positions forward or backward.
- Roll two 4-sided dice (2D4) and move the resultant summed number of positions forward or backward.

Problems:

1. Design a state representation. What is the initial state?

2. Write the pseudocode for a successor function. What kinds of states are there? What does each ply in an adversarial search tree represent?

3. Write the pseudocode for a terminal function.

4. Write the pseudocode for a utility function.

5. Draw out a portion of the adversarial search tree. Show enough ply for at least two turns.

6. Suppose the board is very long and it is impractical to reach terminal states each time. Design a cut off function and evaluation function that returns the value of intermediate, non-terminal states.

1. Design a state representation. What is the initial state?

State is a tuple (p1-location, p2-location, die-roll, whose-turn, phase). The first two are board positions. Die-roll is the sum of the numbers on the dice at the moment. Phase determines which part of a player's turn a player is in. Phase can be {start, 1D4, 2D4, move} as if these were the states of a small FSM with possible implicit transitions {start→1D4, start→2D4, 1D4→move, 2D4→move}.

2. Write the pseudocode for a successor function. What kinds of states are there? What does each ply in an adversarial search tree represent?

States can be min, max, or chance. However, each player has two max nodes per turn: one for picking how to roll the dice, and one for moving forward or backward.

Ply 1: Agent picks to move-1, roll 1D4, or roll 2D4
Ply 2: Chance nodes for die rolls
Ply 3: Agent picks to move forward or backward the number of spaces determined during ply 1 or ply 2.
Ply 4: Opponent picks to move a single space, move 1D4, or move 2D4
Ply 5: Chance nodes for die rolls
Ply 6: Opponent picks to move forward or backward
…

```
Function successors (state = (p1, p2, die, turn, phase))
    IF (phase == 'start')
        RETURN ( [action: 'move-1', state: (p1, p2, 1, turn, 'move')],
                 [action: '1D4', state: (p1, p2, 0, turn, '1D4')],
                 [action: '2D4', state: (p1, p2, 0, turn, '2D4')] )
    IF (phase == '1D4')
        RETURN  ([action: 1, state: (p1, p2, 1, turn, 'move')],
                 [action: 2, state: (p1, p2, 2, turn, 'move')],
                 [action: 3, state: (p1, p2, 3, turn, 'move')],
                 [action: 4, state: (p1, p2, 4, turn, 'move')] )
    IF (phase == '2D4')
        RETURN ( [action: 2, state: (p1, p2, 2, turn, 'move')],
                 [action: 3, state: (p1, p2, 3, turn, 'move')],
                 …
                 [action: 8, state: (p1, p2, 8, turn, 'move')] )
    IF (phase == 'move' and turn == 'p1')
        RETURN ( [action: 'left', state: (p1-die, p2, 0, 'p2', 'start')],
                 [action: 'right', state: (p1+die, p2, start, 'p2', 'start')] )
    IF (phase == 'move' and turn == 'p2')
        RETURN ( [action: 'left', state: (p1, p2-die, 0, 'p1', 'start')],
                 [action: 'right', state: (p1, p2+die, 0, 'p1', 'start')] )
```

(Note: I am assuming the expectiminimax-value recursive function handles computation of probability)

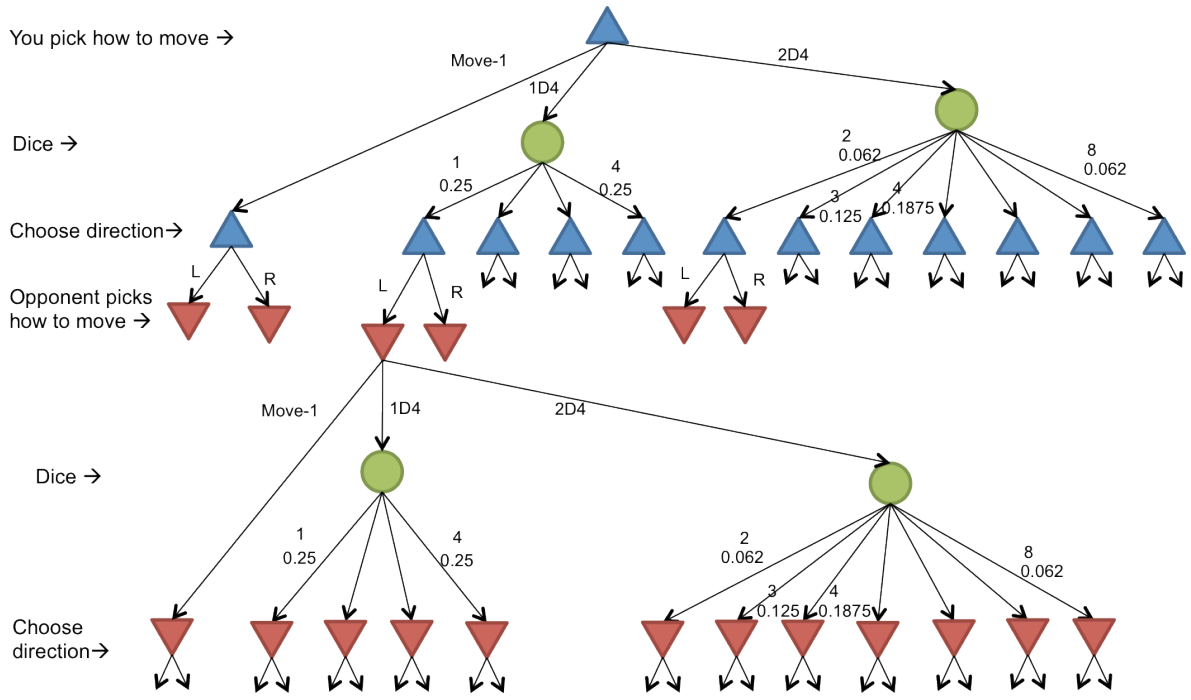
3. Write the pseudocode for a terminal function.

Function terminal (state)
    RETURN (p1 == 19 OR p2 == 19)

4. Write the pseudocode for a utility function.

Function utility (state)
    IF (p1 == 19) return +1
    ELSE return -1

5. Draw out a portion of the adversarial search tree. Show enough ply for at least two turns.



6. Suppose the board is very long and it is impractical to reach terminal states each time. Design a cut off function and evaluation function that returns the value of intermediate, non-terminal states.

A standard fixed depth cut-off can be used. However, it may also be able to compute a cut-off depth based on stability. For example, if one player has a large enough lead (and there are no ladders that will teleport the losing player close to or ahead of the winning player), one could assume that the utility of descendants is unlikely to change.

One way to compute the material evaluation value is the distance between p1-location and p2-location. A note of warning: If the utility function returns +1 or -1 and the evaluation function for non-terminal states returns numbers outside the [-1, +1] range, you can get unexpected results. You may want to find a way to *normalize* the evaluation function so that the range of results is comparable to that of the utility function.