Assignment4 – Adversarial Search

Problem 4.1 (Games for Adversarial Search)

Consider the following *properties* a *game* can have.

A 2 *players* alternating *moves*

I *discrete state space*

C *players* have complete *information* about *state*

F *finite number* of *move* options per *state*

E *deterministic successor states* n

T *games* guaranteed to *terminate*

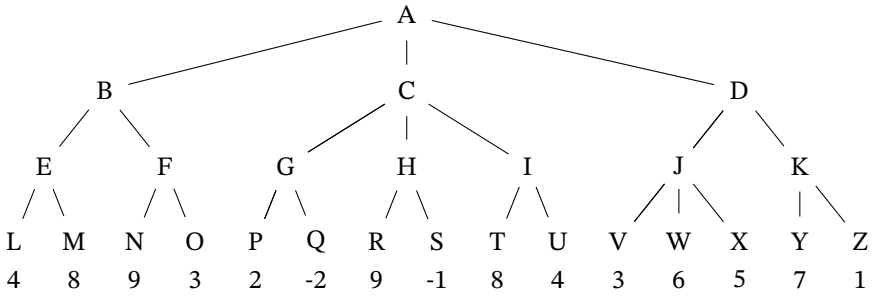
U *terminal state* has zero-sum *utility*.

For each of the following *games*, state whether the *game* violates the *property*; fill in the corresponding *letters* (no spaces) into the box or write "none".

1. 2-*player* poker (until one *player* is bankrupted): ☐
2. *Backgammon*: ☐
3. Wrestling (one 5 *minute* round): ☐
4. Connect Four: ☐
5. Rock-Paper-Scissors (with a repeat to break ties): ☐
6. Meta-Game (*player* 1's first *move* is to choose a *game* that satisfies all *properties*, subsequent *moves play* that *game*): ☐

Problem 4.2 (Game Tree)

Consider the following *game tree*. Assume it is the *maximizing player*'s turn to *move*. The *values* at the *leaves* are the *evaluation function values* of the *states* at each of those *nodes*.



1. *Compute* the *minimax* game *value* of *nodes* A, B, C, and D.
2. *Max* would select *move* 
3. List the *nodes* that the *alpha-beta algorithm* would *prune* (i.e., not visit). Assume *children* of a *node* are visited left-to-right.
4. What reordering of the *evaluation function values* at the bottom would *prune* as many *branches* as possible?

Problem 4.3 (Minimax Applicability)

Consider the following game:

- Initially, 2 players have 10 tokens each, and there is an empty bag of tokens in the center.
- The players take turns either putting an odd number of tokens into the bag or taking a non-zero even number of tokens from the bag.
- A player loses if they have no more tokens.

Explain why minimax can/cannot be used to find a perfect strategy for this game.

Problem 4.4 (Minimax Search in ProLog)

Consider the following *game*:

1. There is a pile of n matches in the middle.
2. Two *players* alternate taking away 1, 2, or 3 matches.
3. The winner is whoever takes the last match.

Solve this *game* (for all *values* of n) by *implementing* the *minimax algorithm* in *Prolog*. Specifically, *implement* exactly the following

- a *Prolog predicate* `value(S,P)` that holds if *player* `P` wins from *initial state* `S`,
- where the *Prolog* constructor `state(N,P)` *represents* the *game state* with `N` remaining matches and *player* `P` going next,
- where we *represent players* `P` using 1 for the *starting player* and -1 for the *opponent*.

Note: A partial solution will be explained in the *tutorials*, especially the use of `\+` for *negation-as-failure* and `!` for *cut*.
