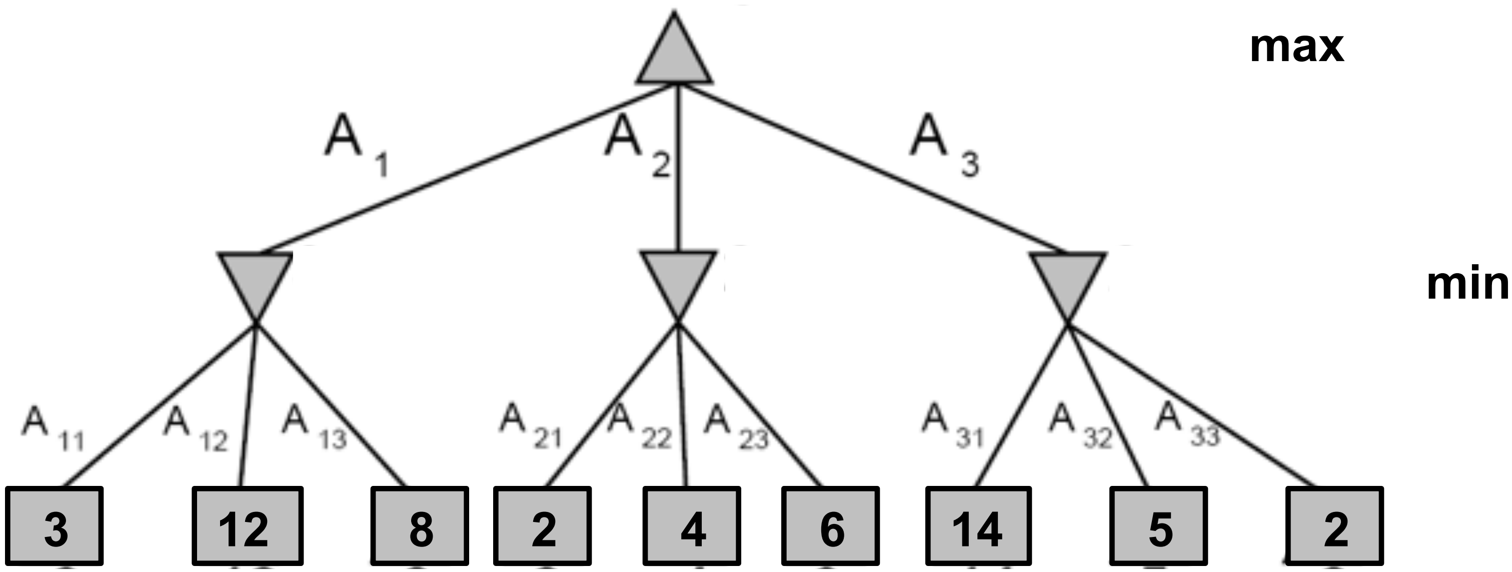
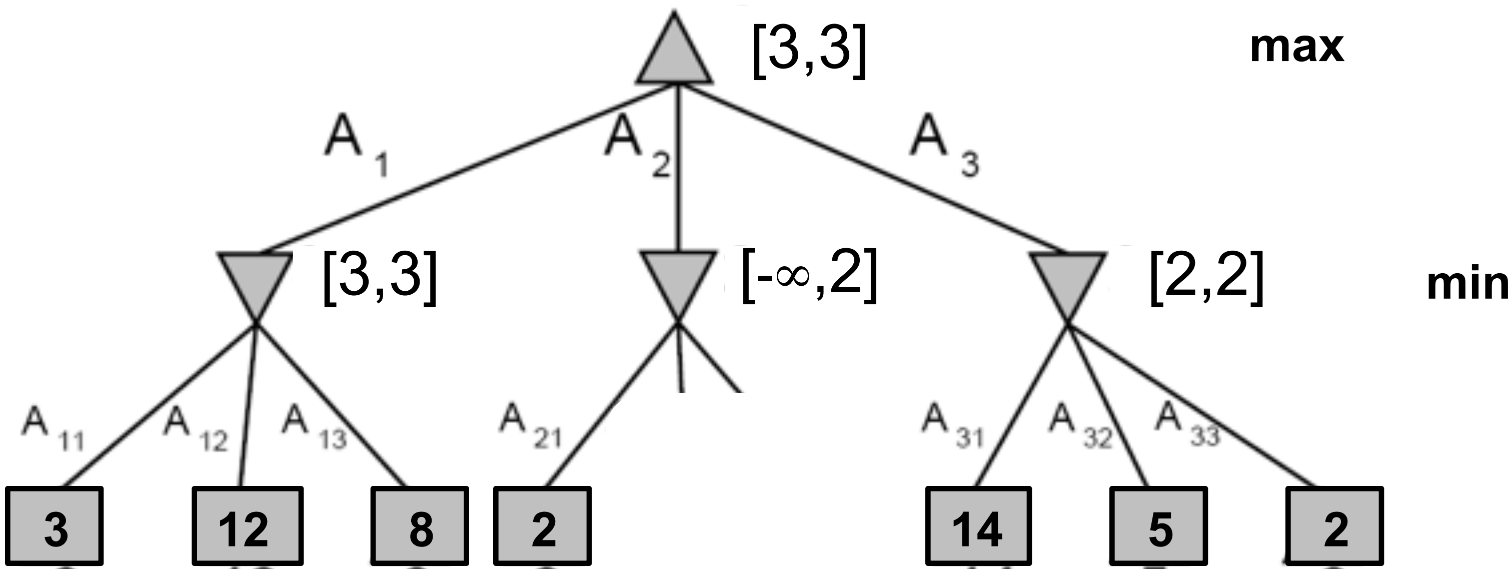
Can we do better?

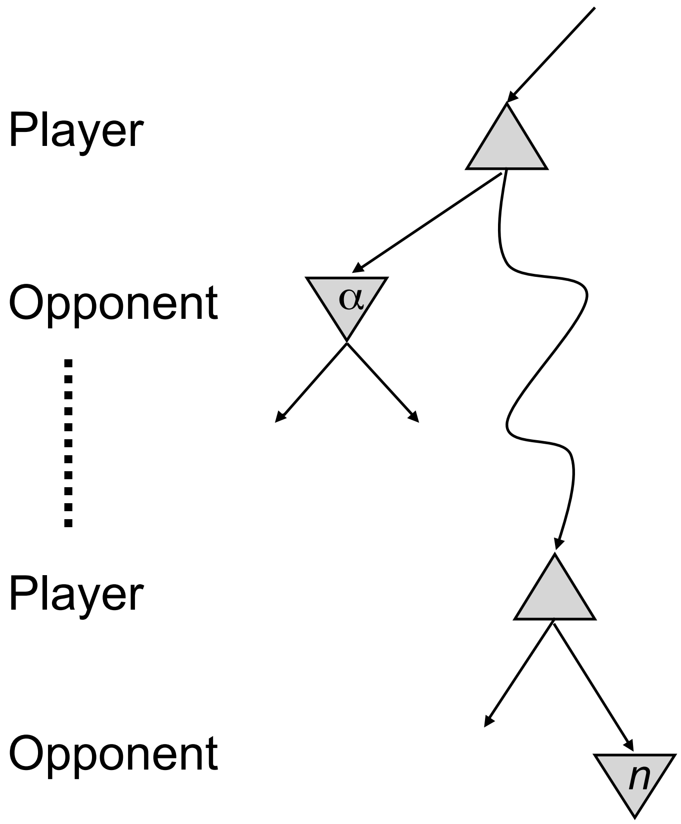


α-β Pruning Example

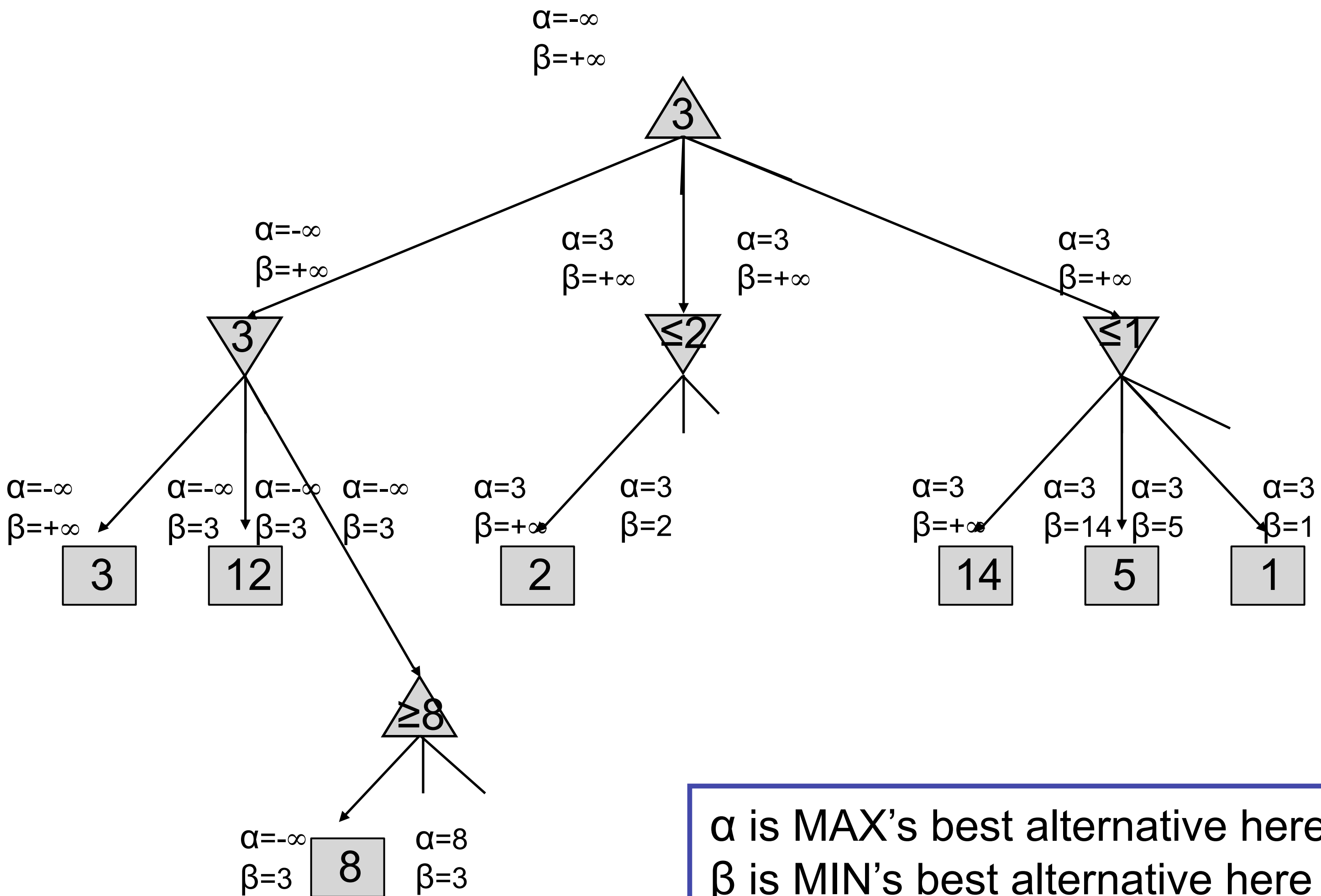


α-β Pruning

- General configuration
  - α is the best value that MAX can get at any choice point along the current path
  - If *n* becomes worse than α, MAX will avoid it, so can stop considering *n*'s other children
  - Define β similarly for MIN



Alpha-Beta Pruning Example



α is MAX's best alternative here or above
β is MIN's best alternative here or above

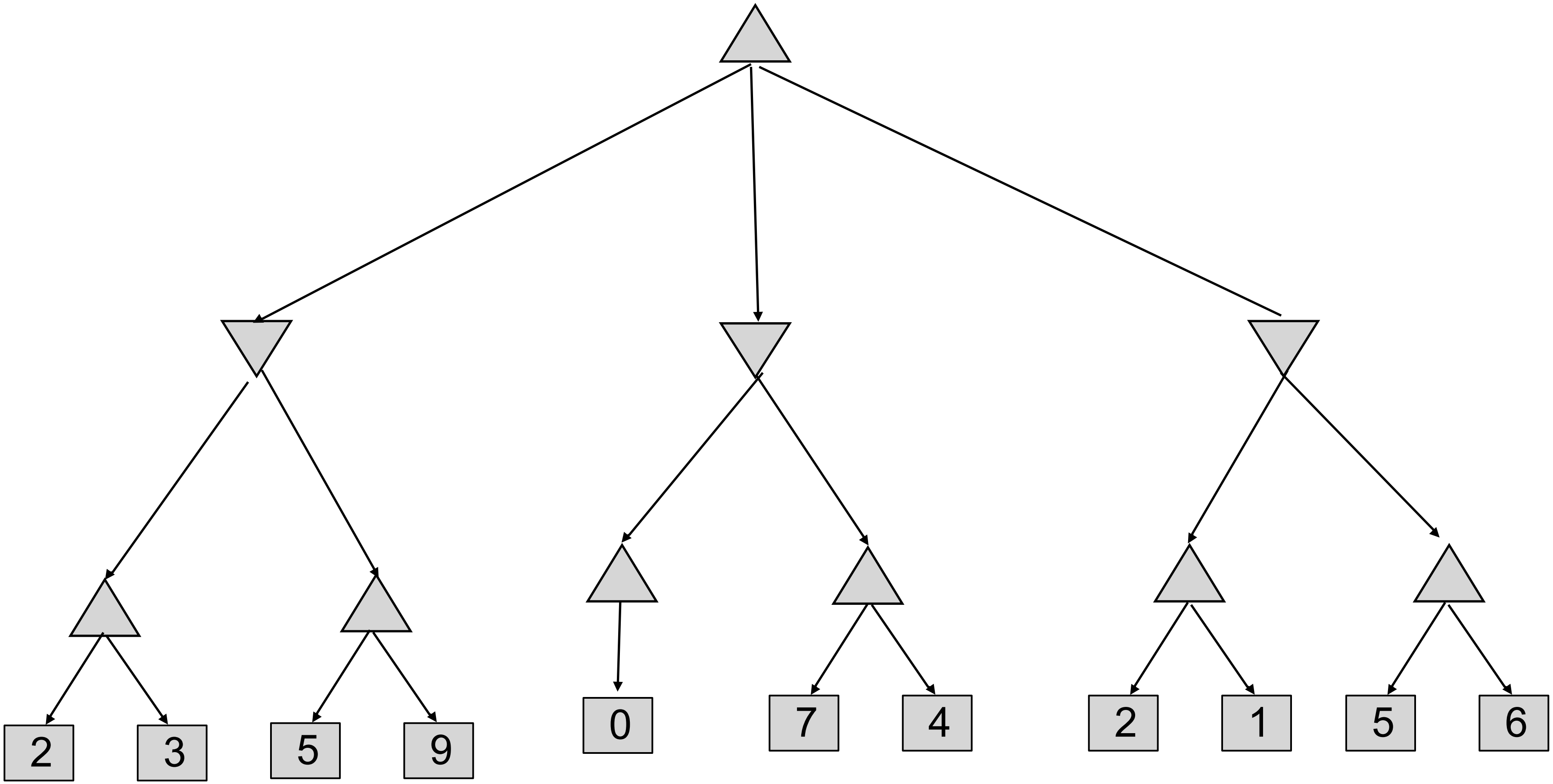
Alpha-Beta Pseudocode

inputs: *state*, current game state
α, value of best alternative for MAX on path to *state*
β, value of best alternative for MIN on path to *state*
returns: *a utility value*

```
function MAX-VALUE(state,α,β)
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s,α,β))
        if v ≥ β then return v
        α ← MAX(α,v)
    return v
```

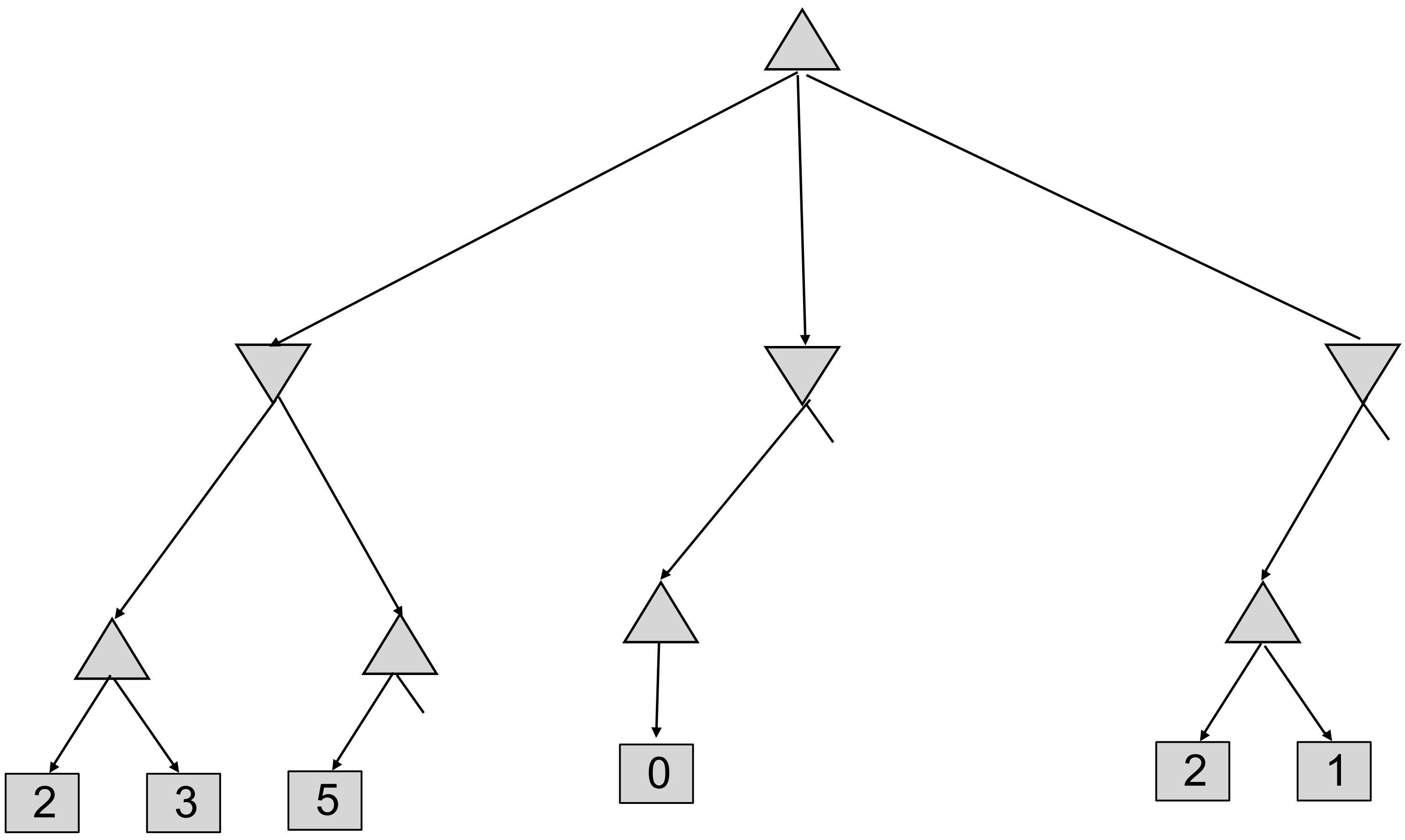
```
function MIN-VALUE(state,α,β)
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s,α,β))
        if v ≤ α then return v
        β ← MIN(β,v)
    return v
```

Alpha-Beta Pruning Example



α is MAX's best alternative here or above
β is MIN's best alternative here or above

Alpha-Beta Pruning Example



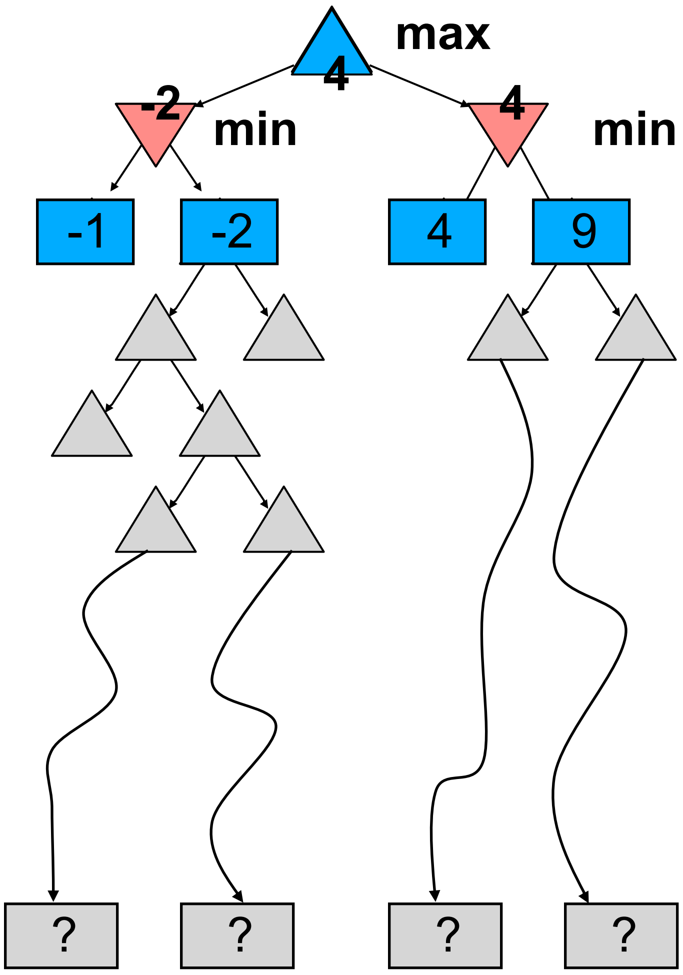
α is MAX's best alternative here or above
β is MIN's best alternative here or above

Alpha-Beta Pruning Properties

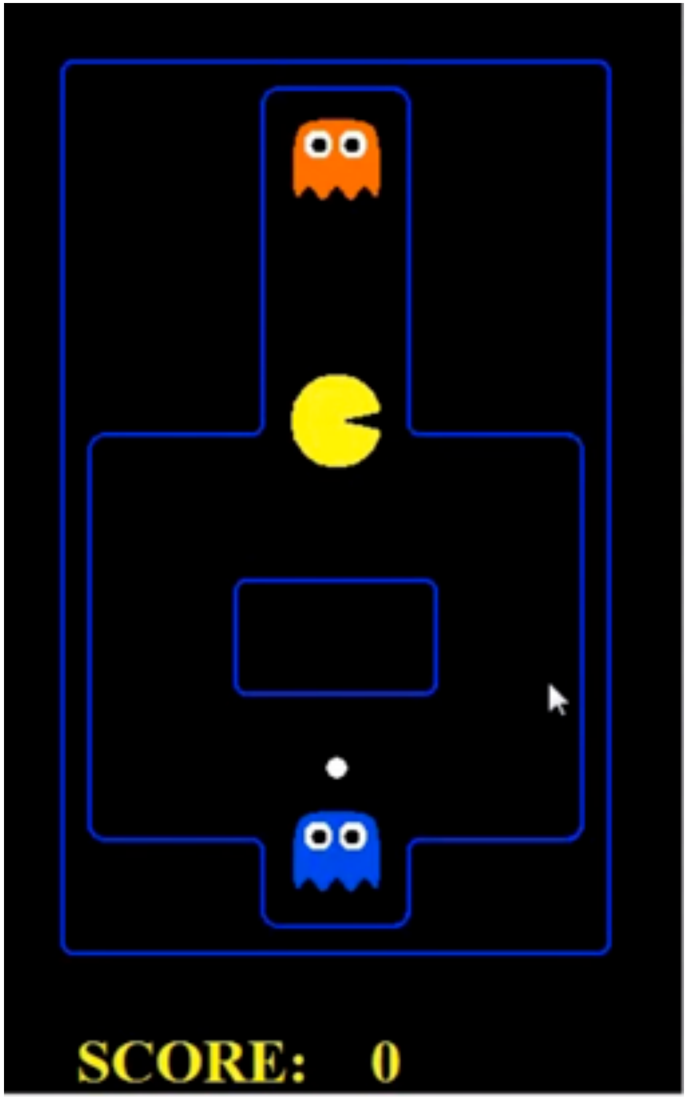
- This pruning has no effect on final result at the root
- Values of intermediate nodes might be wrong!
  - but, they are bounds
- Good child ordering improves effectiveness of pruning
- With “perfect ordering”:
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth!
  - Full search of, e.g. chess, is still hopeless…

Resource Limits

- Cannot search to leaves
- Depth-limited search
  - Instead, search a limited depth of tree
  - Replace terminal utilities with an eval function for non-terminal positions
  - e.g., α-β reaches about depth 8 – decent chess program
- Guarantee of optimal play is gone
- Evaluation function matters
  - It works better when we have a greater depth look ahead

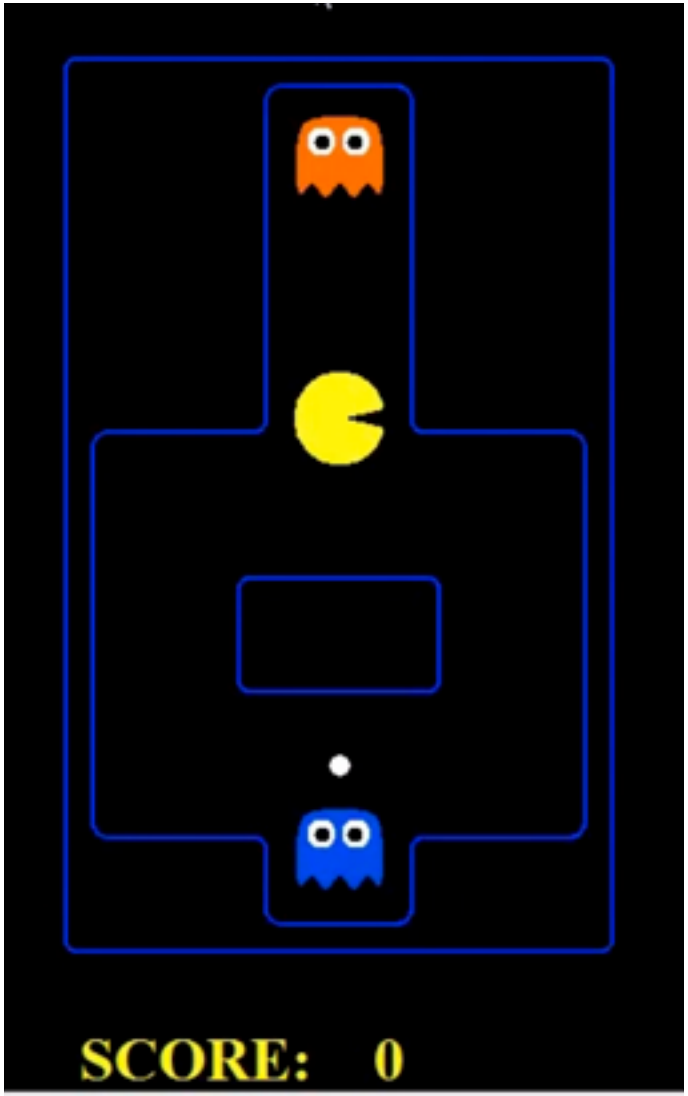


Depth Matters



depth 2

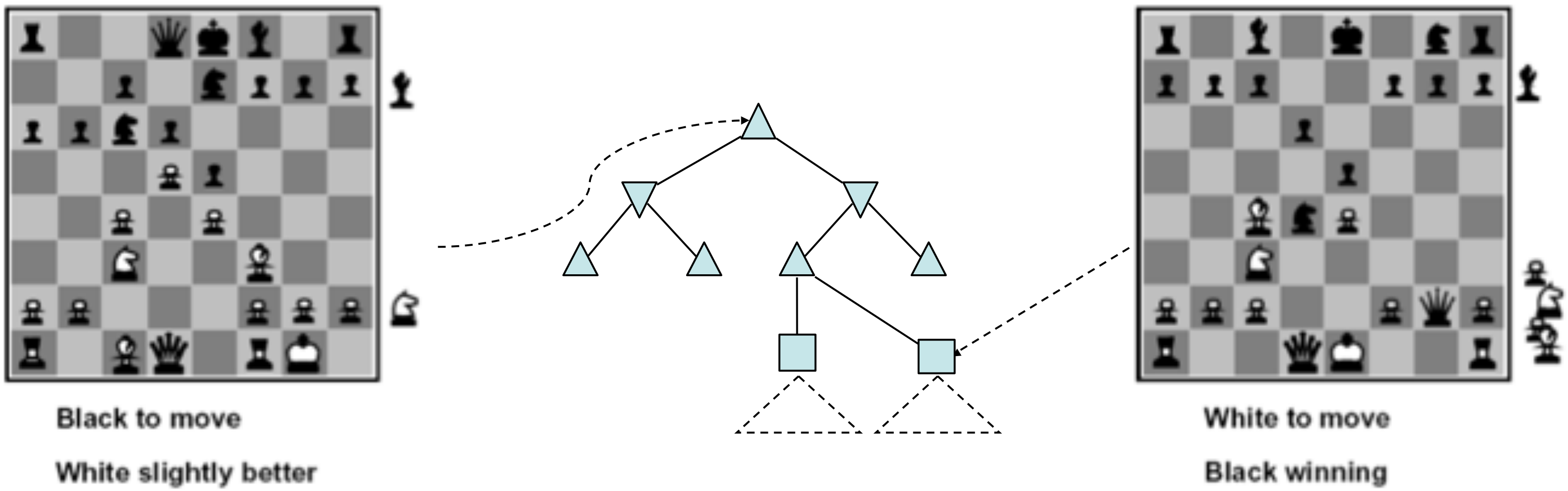
Depth Matters



depth 10

Evaluation Functions

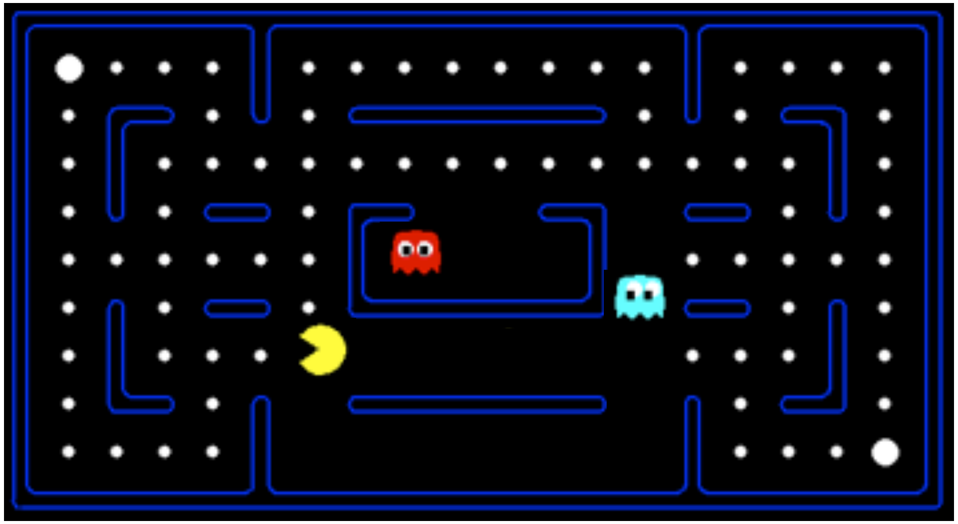
- Function which scores non-terminals



$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  - e.g. $f_1(s)$ = (num white queens – num black queens), etc.

Evaluation for Pacman



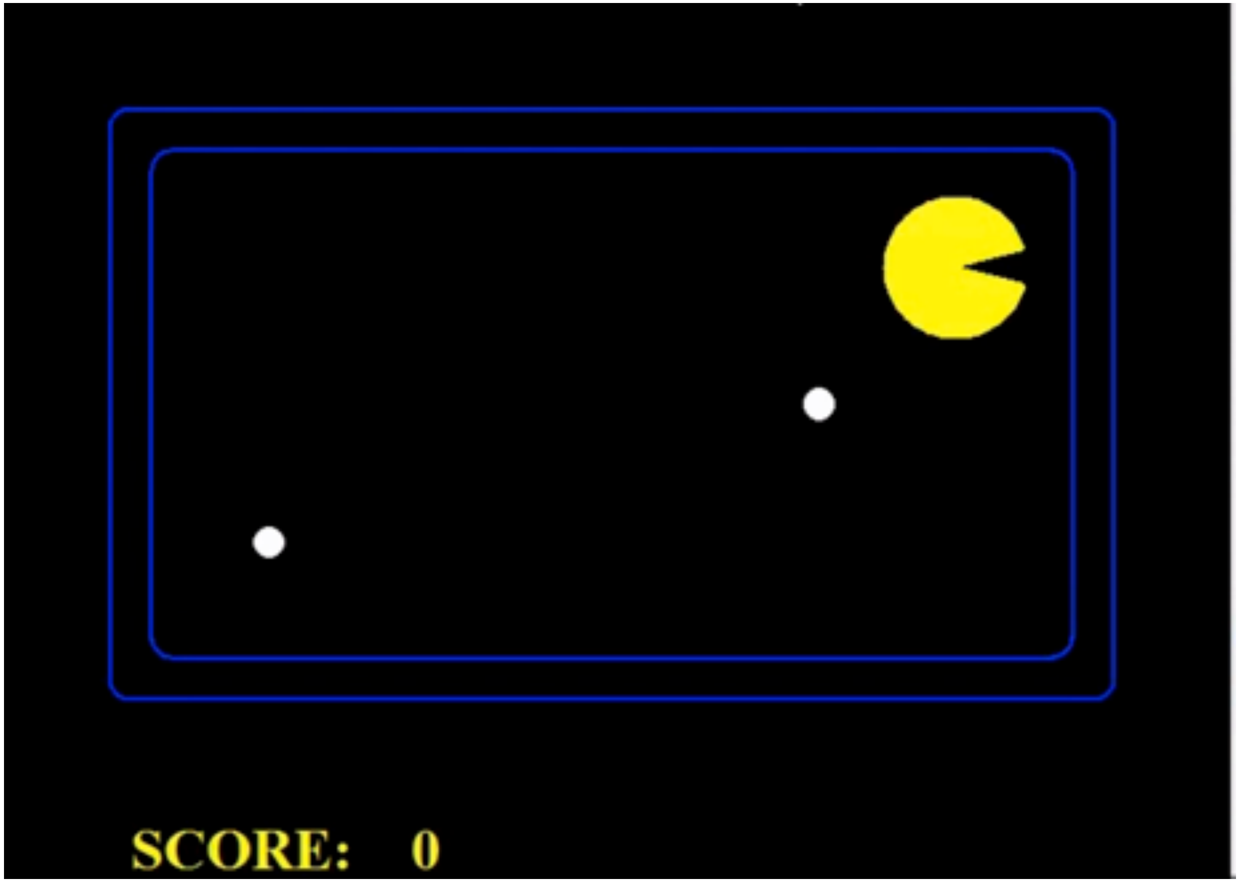
What features would be good for Pacman?

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

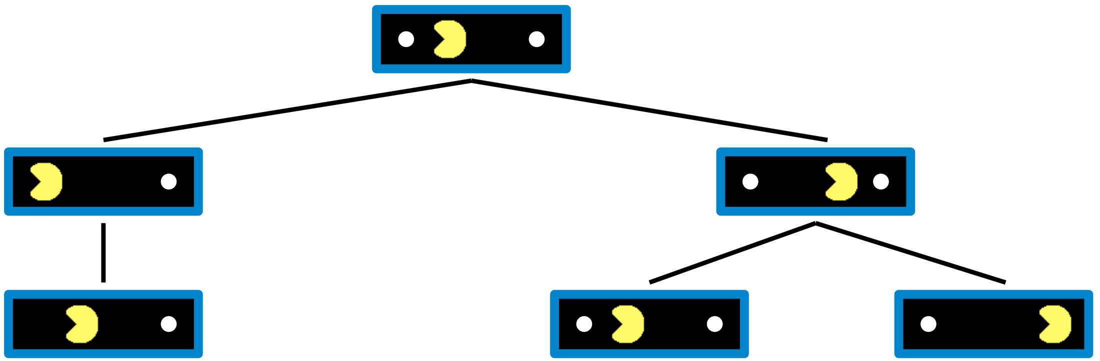
Evaluation Function

Evaluation Function

Bad Evaluation Function



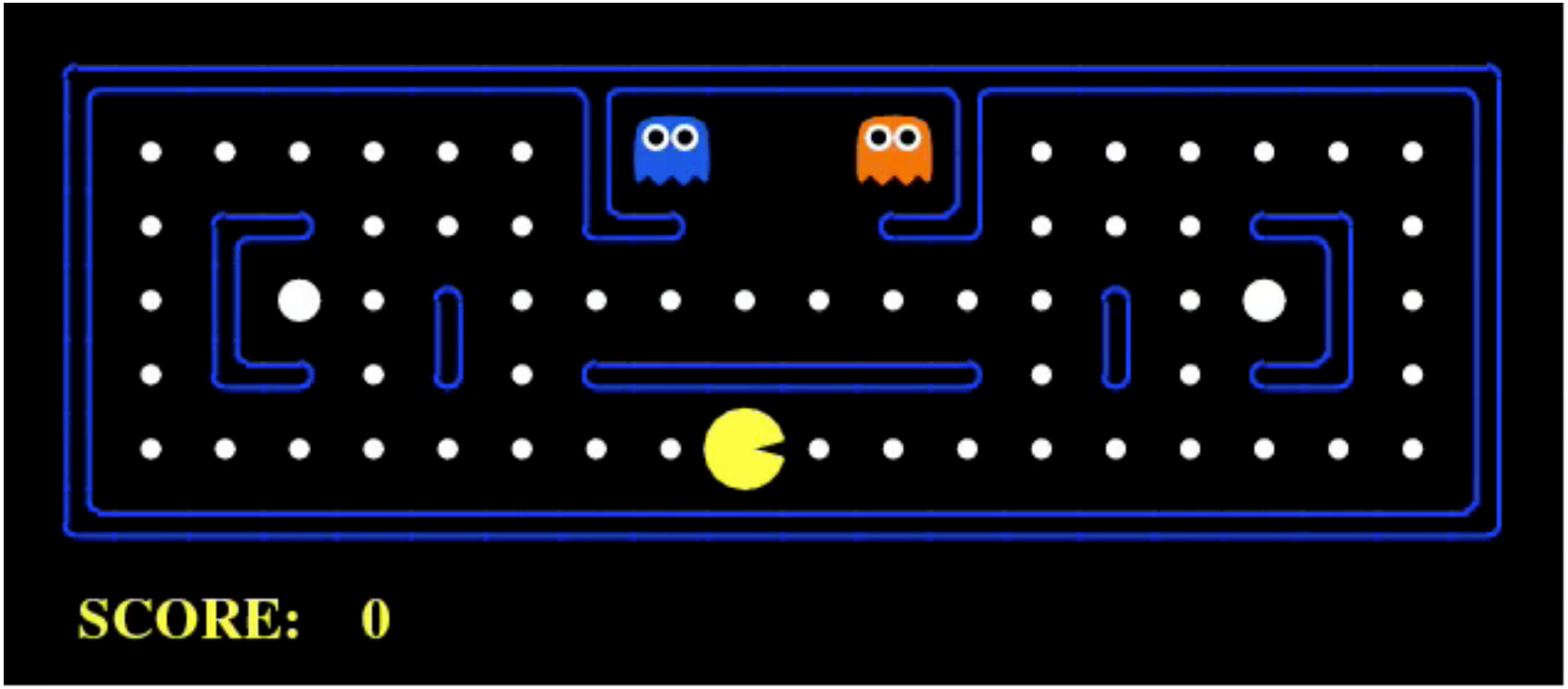
Why Pacman Starves



- He knows his score will go up by eating the dot now
- He knows his score will go up just as much by eating the dot later on
- There are no point-scoring opportunities after eating the dot
- Therefore, waiting seems just as good as eating

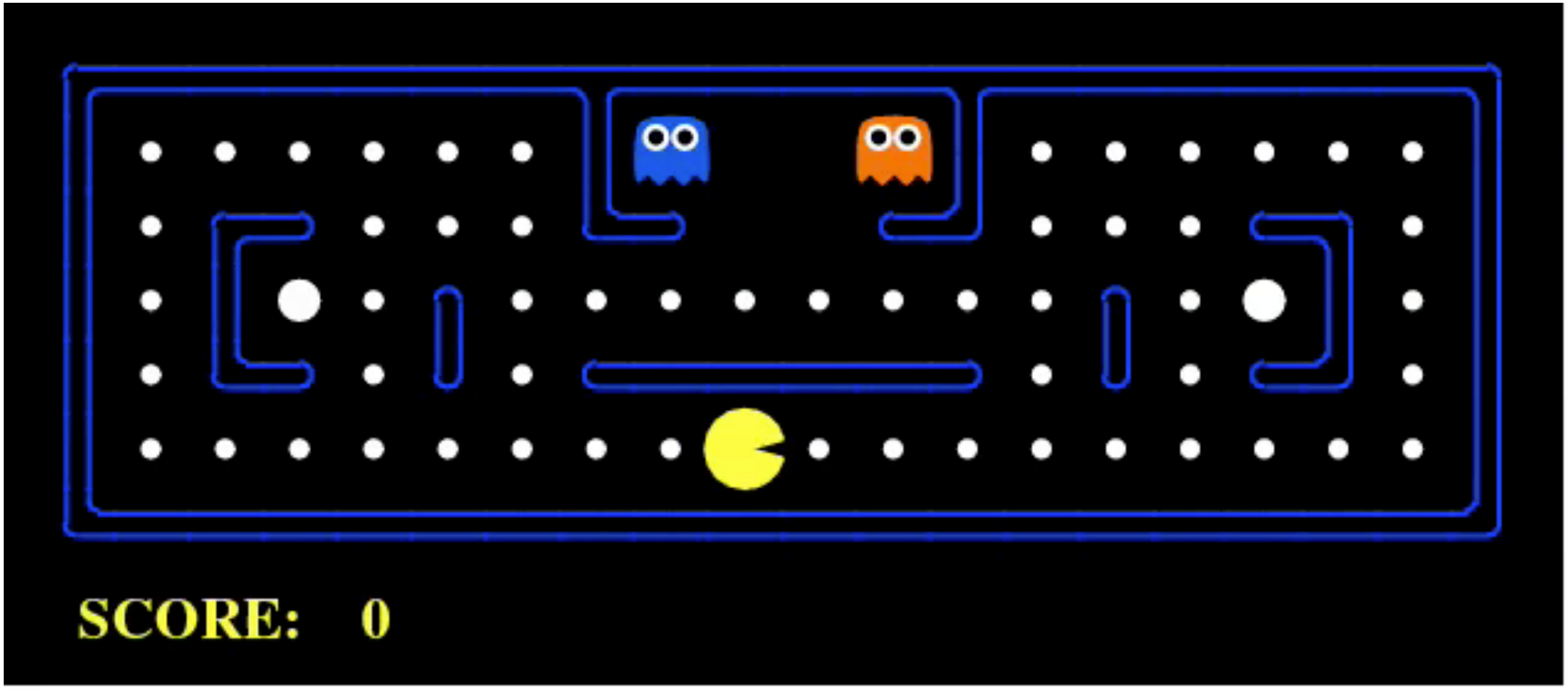
Which algorithm?

α-β, depth 4, simple eval fun

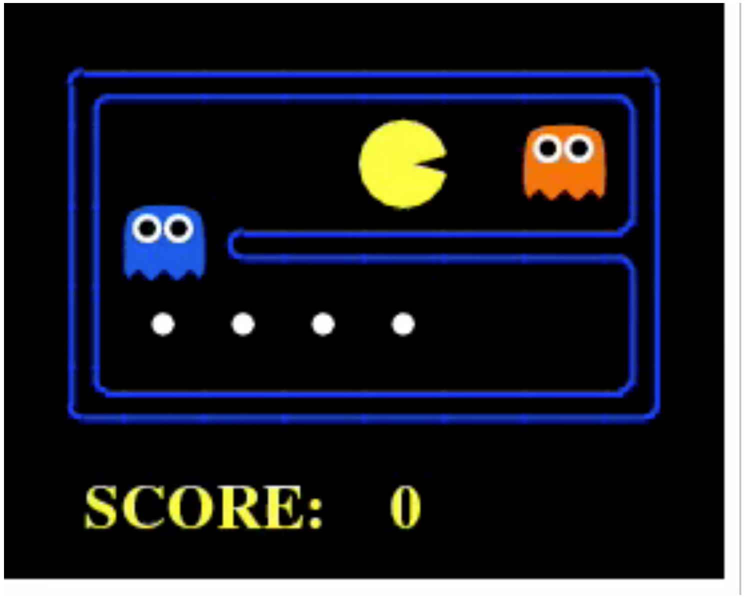


Which algorithm?

α-β, depth 4, better eval fun

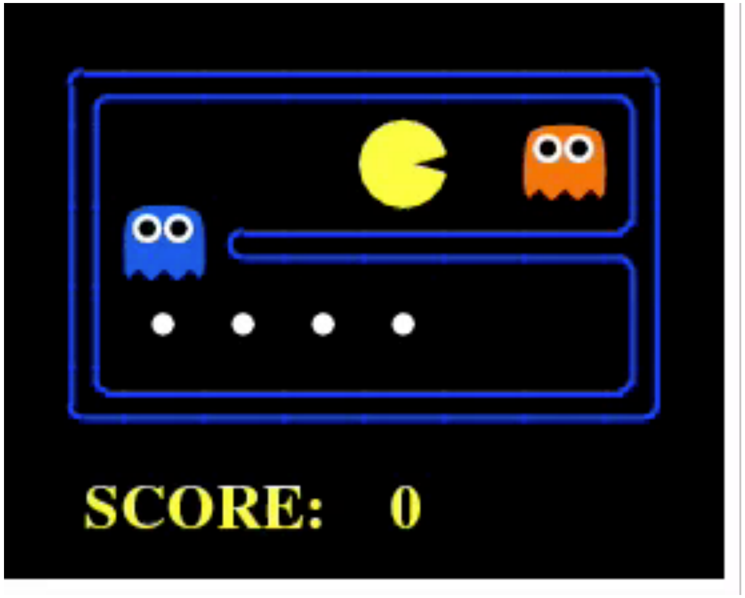


Minimax Example



Suicidal agent

Expectimax



- Uncertain outcomes are controlled by chance not an adversary
- Chance nodes are new types of nodes (instead of Min nodes)