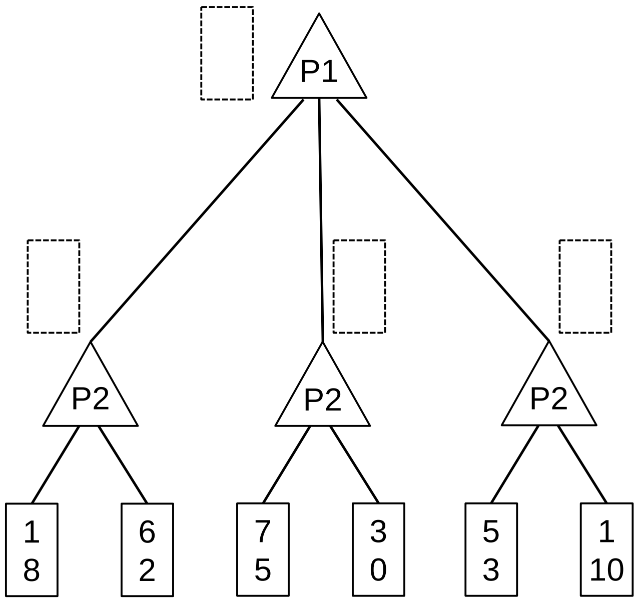
# Games

## Q1. Games

For the following game tree, each player maximizes their respective utility. Let $x, y$ respectively denote the top and bottom values in a node. Player 1 uses the utility function $U_1(x, y) = x$.



Both players know that Player 2 uses the utility function $U_2(x, y) = x - y$.

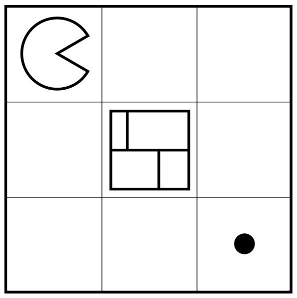
**(a)** Fill in the rectangles in the figure above with pair of values returned by each max node.

**(b)** You want to save computation time by using pruning in your game tree search. On the game tree above, put an 'X' on branches that do not need to be explored or simply write 'None'. Assume that branches are explored from left to right.

## Q2. Surrealist Pacman

In the game of Surrealist Pacman, Pacman ◔ plays against a moving wall ⊞. On Pacman's turn, Pacman must move in one of the four cardinal directions, and must move into an unoccupied square. On the wall's turn, the wall must move in one of the four cardinal directions, and must move into an unoccupied square. The wall cannot move into a dot-containing square. Staying still is not allowed by either player. Pacman's score is always equal to the number of dots he has eaten.

The first game begins in the configuration shown below. Pacman moves first.



**(a)** Draw a game tree with one move for each player. Nodes in the tree represent game states (location of all agents and walls). Edges in the tree connect successor states to their parent states. Draw only the legal moves.

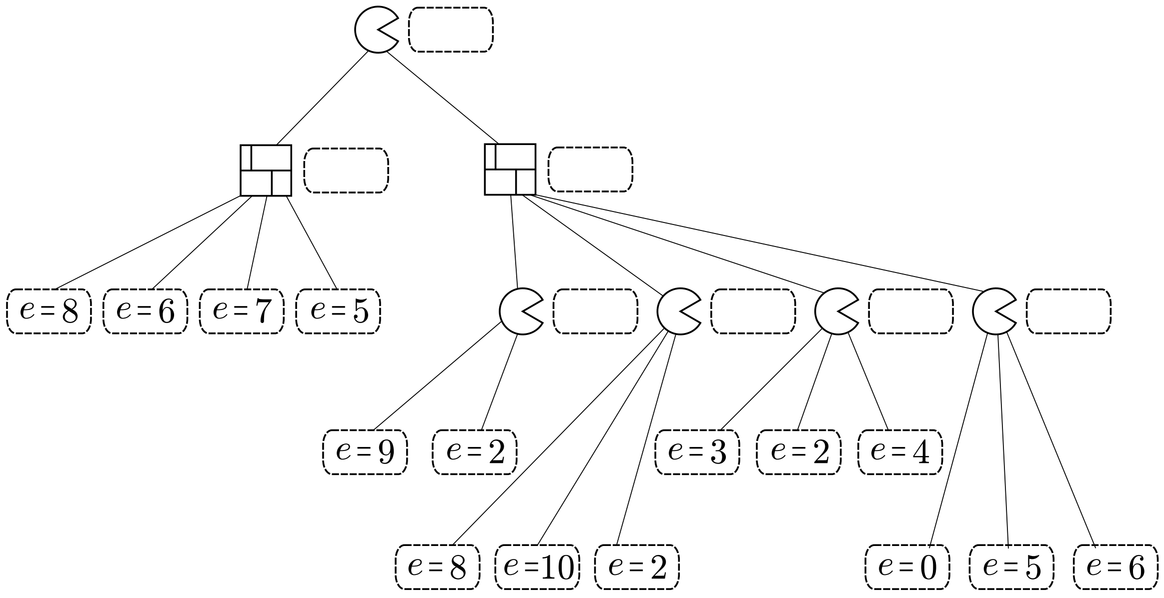
**(b)** According to the depth-limited game tree you drew above what is the value of the game? Use Pacman's score as your evaluation function.

**(c)** If we were to consider a game tree with ten moves for each player (rather than just one), what would be the value of the game as computed by minimax?

A second game is played on a more complicated board. A partial game tree is drawn, and leaf nodes have been scored using an (unknown) evaluation function *e*.

**(d)** In the dashed boxes, fill in the values of all internal nodes using the minimax algorithm.

**(e)** Cross off any nodes that are not evaluated when using alpha-beta pruning (assuming the standard left-to-right traversal of the tree).



Suppose that this evaluation function has a special property: it is known to give the correct minimax value of any internal node to within 2, and the correct minimax values of the leaf nodes exactly. That is, if $v$ is the true minimax value of a particular node, and $e$ is the value of the evaluation function applied to that node, $e - 2 \leq v \leq e + 2$, and $v = e$ if the node is a dashed box in the tree below.

Using this special property, you can modify the alpha-beta pruning algorithm to prune more nodes.

**(f)** Standard alpha-beta pseudocode is given below (only the max-value recursion). Fill in the boxes on the right to replace the corresponding boxes on the left so that the pseudocode prunes as many nodes as possible, taking account of this special property of the evaluation function.

```
function MAX-VALUE(node, α, β)
    e ← EVALUATIONFUNCTION(node)
    if node is leaf then
        return e
    end if
    [          ] (1)
    v ← −∞
    for child ← CHILDREN(node) do
        [ v ← MAX(v, MIN-VALUE(child, α, β)) ] (2)
        if v ≥ β then
            return v
        end if
        α ← MAX(α, v)
    end for
    return v
end function
```

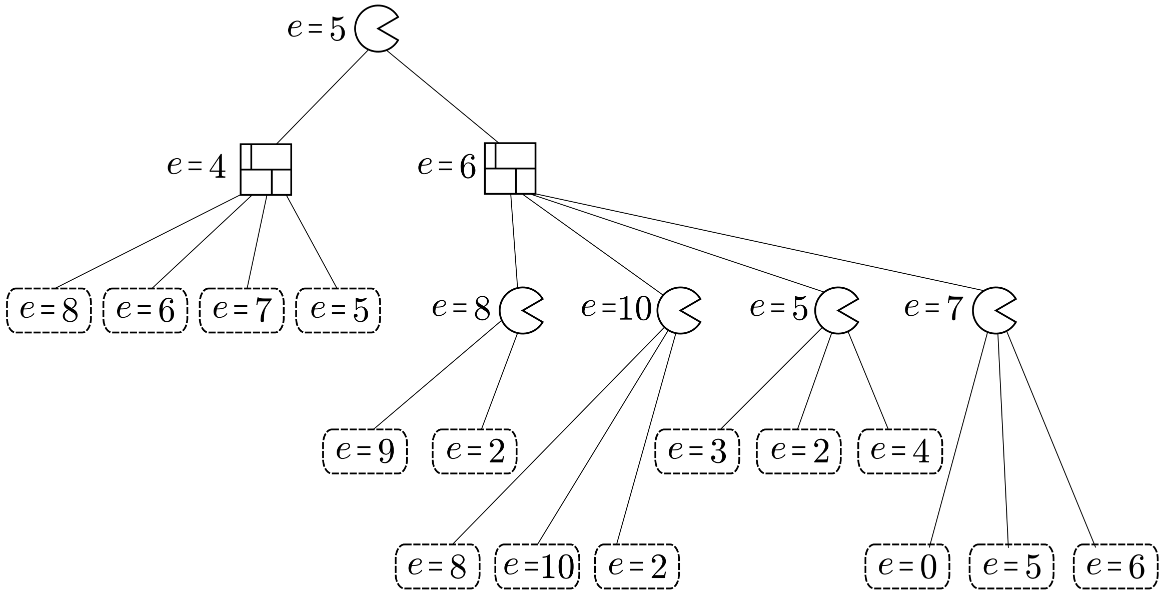
Fill in these boxes:

(1)

(2)

The same game tree is shown below, with the evaluation function applied to *internal* as well as leaf nodes.

**(g)** In the game tree below cross off any nodes that can be pruned assuming the special property holds true. If not sure you correctly formalized into pseudo-code your intuition on how to exploit the special property for improved pruning, make sure to annotate your pruned nodes with a brief explanation of why each of them was pruned.



e=5
e=4
e=6
e=8
e=6
e=7
e=5
e=8
e=10
e=5
e=7
e=9
e=2
e=3
e=2
e=4
e=8
e=10
e=2
e=0
e=5
e=6