National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 4: Adversarial Search

Issued: February 25, 2020 Due: Week 6 in the tutorial class

Important Instructions:

- *Your solutions for this tutorial must be TYPE-WRITTEN.*
- *Make TWO copies of your solutions: one for you and one to be SUBMITTED TO THE TUTOR IN CLASS. Your submission in your respective tutorial class will be used to indicate your CLASS ATTENDANCE. Late submission will NOT be entertained.*
- *YOUR SOLUTION TO QUESTION* 3 *will be GRADED for this tutorial.*
- *You may discuss the content of the questions with your classmates. But everyone should work out and write up ALL the solutions by yourself.*

An extensive form game is defined by V a set of nodes and E a set of directed edges, defining a tree. The root of the tree is the node r. Let $V_{\max}$ be the set of nodes controlled by the MAX player and $V_{\min}$ be the set of nodes controlled by the MIN player. We often refer to the MAX player as player 1, and to the MIN player as player 2. Furthermore, let $desc(v)$ be the descendants of a node v in the search tree. A *strategy* for the MAX player is a mapping $s_1 : V_{\max} \to V$; similarly, a strategy for the MIN player is a mapping $s_2 : V_{\min} \to V$. In both cases, $s_i(v) \in chld(v)$ is the choice of child node that will be taken at node v. We let $\mathcal{S}_1, \mathcal{S}_2$ be the set of strategies for the MAX and MIN player, respectively.

The leaves of the minimax tree are *payoff nodes*. There is a payoff $a(v)$ associated with each payoff node v, where $a(v)$ is the amount that the MAX player receives and $-a(v)$ is the amount that the MIN player receives if the node v is reached. The MAX player wants to maximize their payoff, and the MIN player wants to minimize the amount that they pay. More formally, the utility of the MAX player from v is $u_{\max}(v) = a(v)$ and the utility of the MIN player is $u_{\min}(v) = -a(v)$. The utility of a player from a pair of strategies $s_1 \in \mathcal{S}_1, s_2 \in \mathcal{S}_2$ is simply the utility they receive by the leaf node reached when the strategy pair (s_1, s_2) is played.

Definition 1 (Nash Equilibrium)**.** A pair of strategies $s_1^* \in \mathcal{S}_1, s_2^* \in \mathcal{S}_2$ for the MAX player and the MIN player, respectively, is a *Nash equilibrium* if no player can get a strictly higher utility by switching their strategy. In other words:

$$\forall s \in \mathcal{S}_1 : u_1(s_1^*, s_2^*) \geq u_1(s, s_2^*);$$
$$\forall s' \in \mathcal{S}_2 : u_2(s_1^*, s_2^*) \geq u_2(s_1^*, s')$$

Definition 2 (Subgame)**.** Given an extensive form game $\langle V, E, r, V_{\max}, V_{\min}, \vec{a}\rangle$, a subgame is a subtree of the original game, defined by some arbitrary node v set to be the root node, and all of its descendants (i.e. its children, its children's children etc.), denoted by $desc(v)$. Leaf nodes and node control are the same as in the original extensive form game.

Definition 3 (Subgame-Perfect Nash Equilibrium (SPNE))**.** A pair of strategies $s_1^* \in \mathcal{S}_1, s_2^* \in \mathcal{S}_2$ is a sub-perfect Nash equilibrium if it is a Nash equilibrium for any subtree of the original game tree.

1. Consider Figure 5.1 in the AIMA 3rd edition textbook (reproduced in Figure 1). The Tic-Tac-Toe search space can actually be reduced by means of symmetry. This is done by eliminating those states which become identical with an earlier state after a symmetry operation (e.g. rotation). The following diagram shows a reduced state space for the first three levels with the player making the first move using "x" and the opponent making the next move with "o". Assume that the following heuristic evaluation function is used at each leaf node n:

$$Eval(n) = P(n) - O(n)$$

where $P(n)$ is the number of winning lines for the player while $O(n)$ is the number of winning lines for the opponent. A winning line for the player is a line (horizontal, vertical or diagonal) that either contains nothing or "x". For the opponent, it is either nothing or "o" in the winning line. Thus, for the leftmost leaf node in Figure 2, $Eval(n) = 6 - 5 = 1$.

   (a) Use the minimax algorithm to determine the first move of the player, searching 2-ply deep search space shown in Figure 2.

   (b) Assume that the "x" player will now make his second move after his opponent has placed an "o". Complete the following minimax tree in Figure 3 by filling the remaining blank boards at the leaf nodes. Compute the evaluation function for each of the filled leaf nodes and determine the second move of the "x" player (searching 2-ply deep).

Solution:

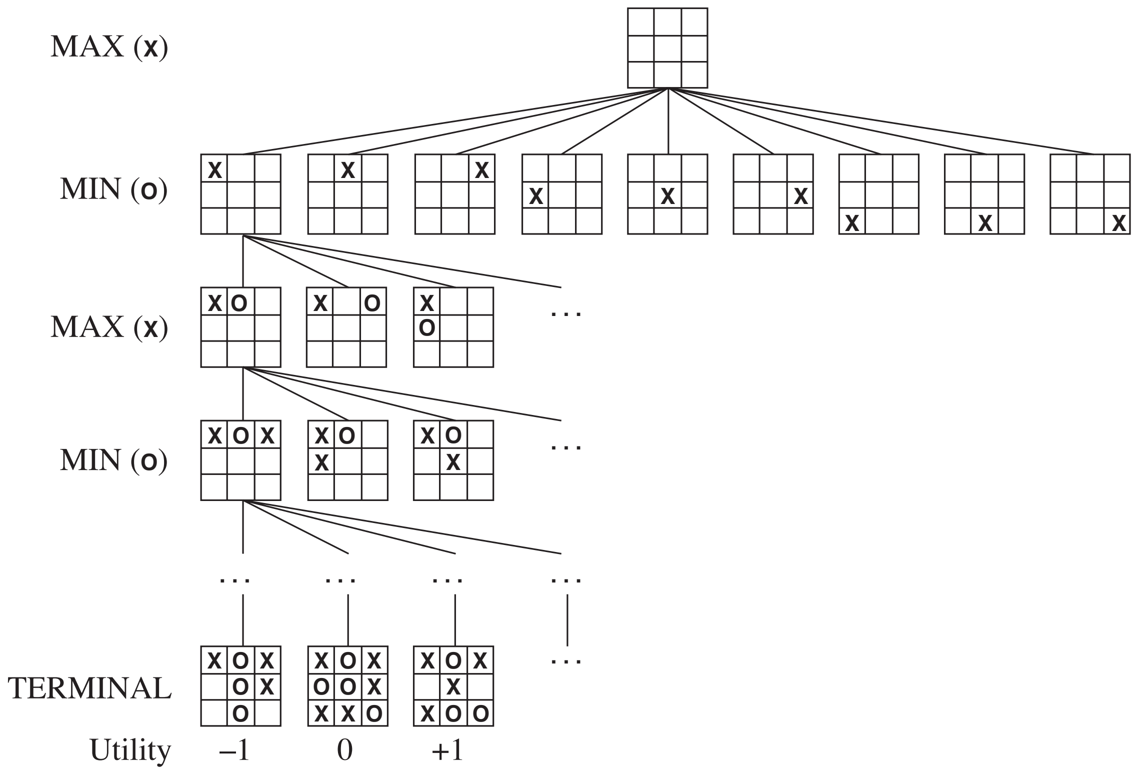

Figure 1: Search space for Tic-Tac-Toe.

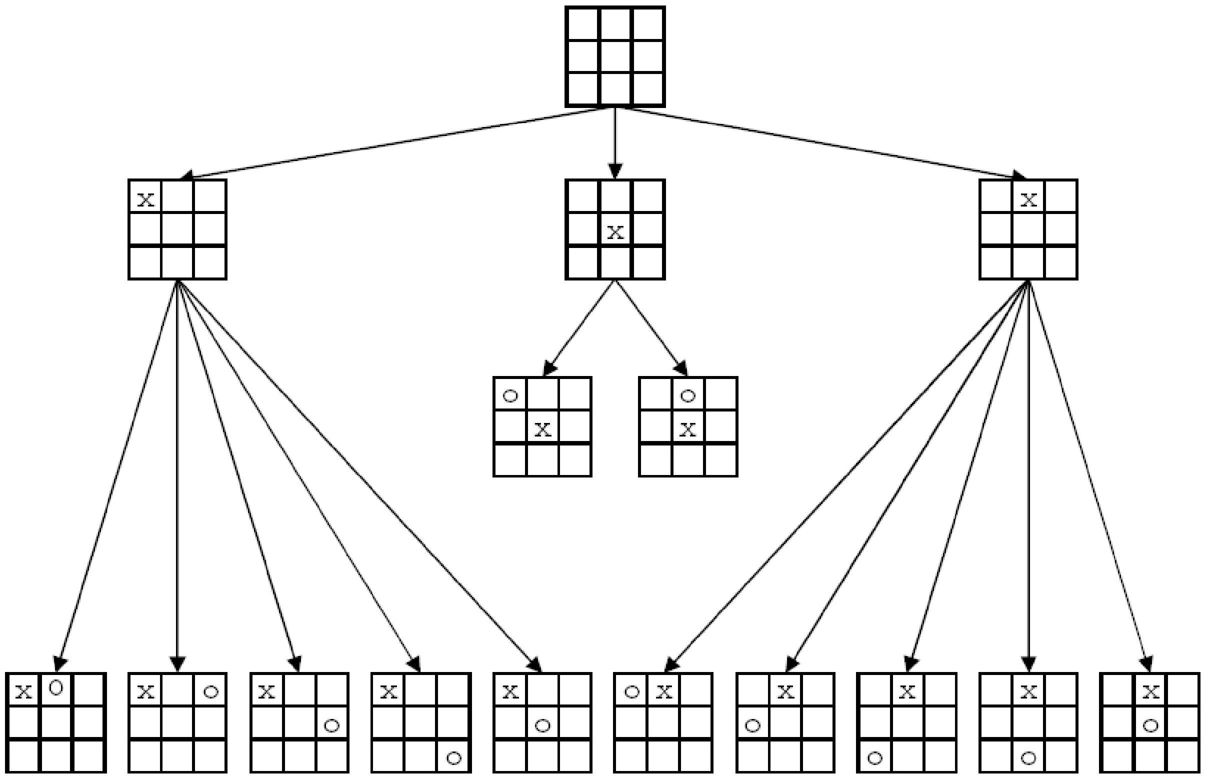

Figure 2: 2-ply deep search space

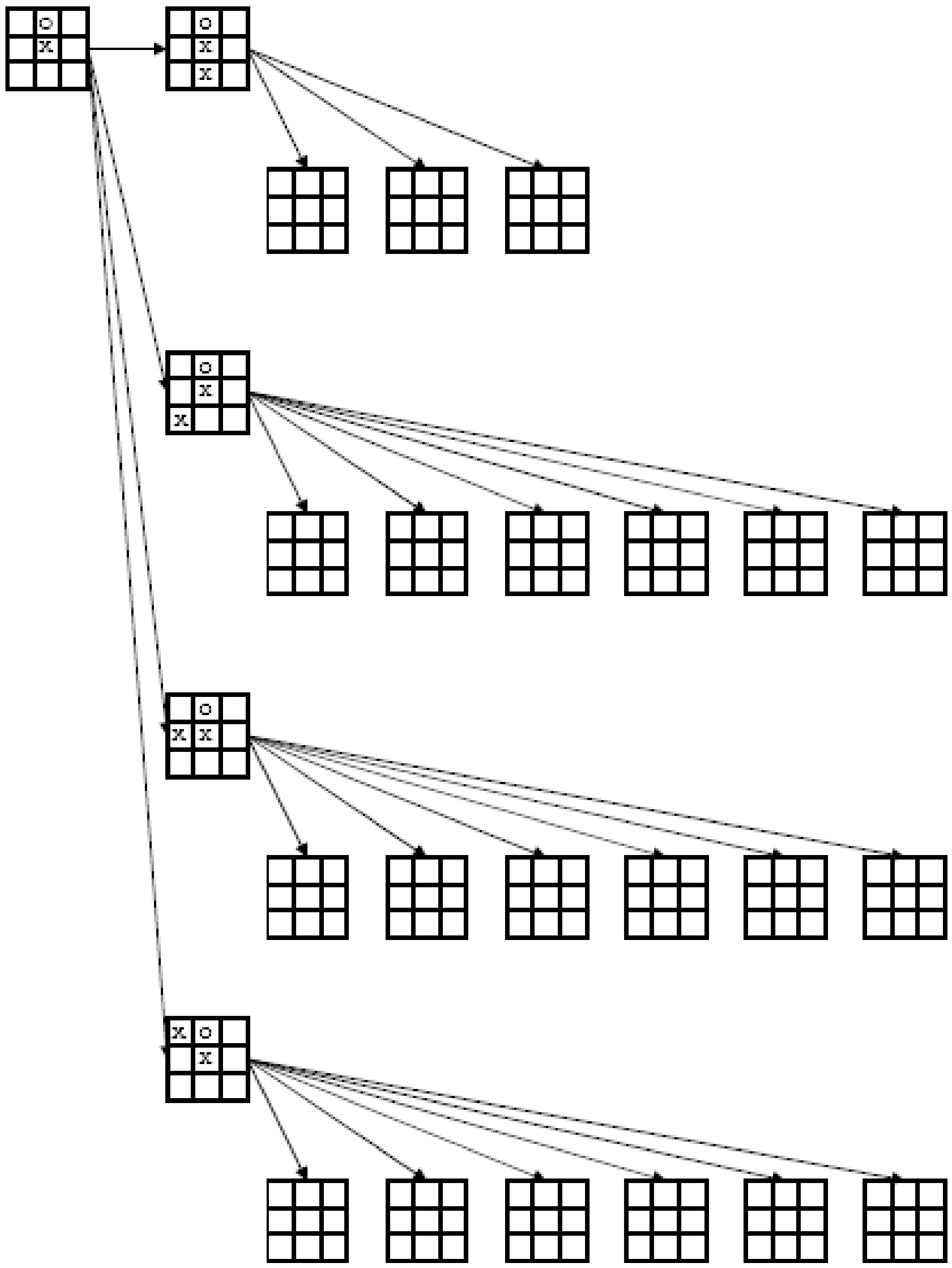

Figure 3: 3-ply deep search space

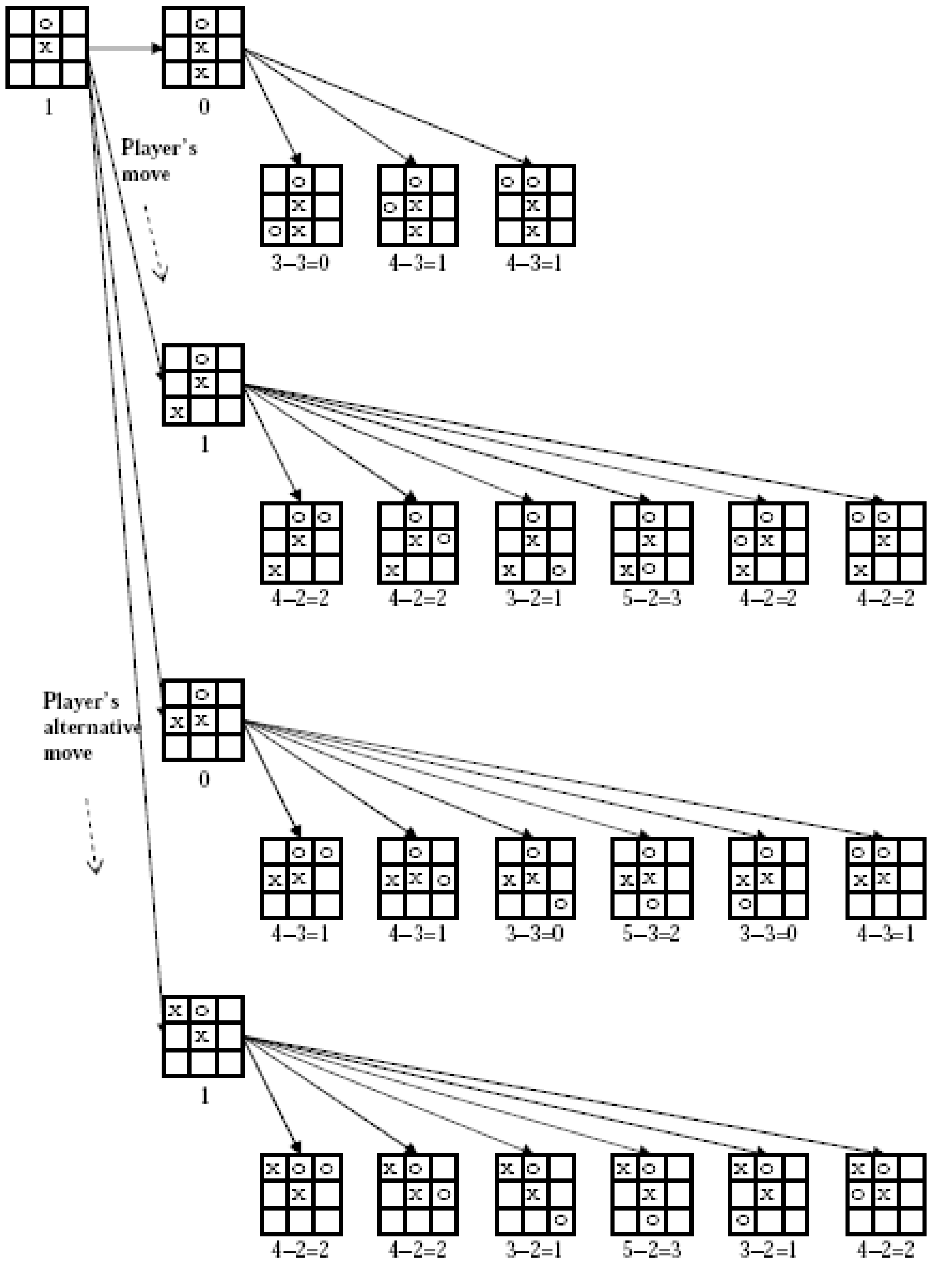

Figure 4: 3-ply deep search space

2. (**Past-year exam question**) Consider the minimax search tree shown in the solution box below. In the figure, black nodes are controlled by the MAX player, and white nodes are controlled by the MIN player. Payments (terminal nodes) are squares; the number within denotes the amount that the MIN player pays to the MAX player (an amount of 0 means that MIN pays nothing to MAX). Naturally, MAX wants to maximize the amount they receive, and MIN wants to minimize the amount they pay.

   Suppose that we use the α-β pruning algorithm, given in Figure 5.7 of AIMA 3rd edition (reproduced in Figure 5). Consider the minimax tree given in Figure 6.

   (a) **Assume that we iterate over nodes from right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

   (b) Does you answer change if we iterate over nodes from **left to right**?

```
function ALPHA-BETA-SEARCH(state) returns an action
  $v \leftarrow$ MAX-VALUE(state, $-\infty, +\infty$)
  return the action in ACTIONS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure 5: Alpha-beta pruning algorithm (note that $s = state$).

Solution: We refer to the players' actions as R (right) M (middle) and L (left). We let $\alpha(x)$ be the least payoff that MAX can guarantee at node x, and $\beta(x)$ be the maximal payoff MIN has to pay at node x. Let us consider a run of the α-β pruning algorithm here. First, the MAX player will explore R, followed by the MIN player exploring R. At this point, $\beta(u_2) = 10$. Next, MIN plays M and explores v_4; MAX explores R and observes a value

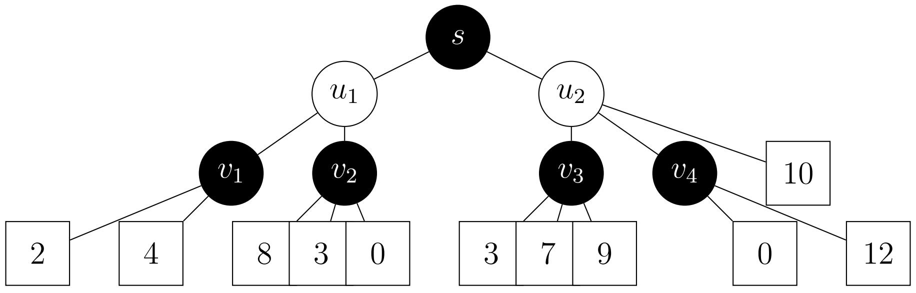

Figure 6: A minimax search tree.

of 12 (so $\alpha(v_4) = 12$); since this value is greater than $\beta(u_2)$ we know that choosing M at u_2 yields a lower value than playing R, the edge between v_4 and 0 is pruned. Next, MIN explores L (the node v_3), seeing the values 9, 7 and 3 in this order; this yields $\alpha(v_3) = 9$), and we update $\beta(u_2) = 9$, and subsequently $\alpha(s) = 9$ (in other words, if MAX chooses R at s they can get a payoff of at least 9).

Next, MAX explores L at s, MIN explores R at u_1, and MAX explores R, followed by M and L, at v_2. This yields $\alpha(v_2) = 8$, which means that $\beta(u_1) = 8$ which is strictly smaller than $\alpha(s)$. What this immediately implies is that the action L is strictly worse for MAX than the action R, and there is no point in continuing to explore its subtree: if MAX chooses L then MIN can choose R and obtain a better outcome (for MIN). At this point, there is no more need to explore the action L at u_1 (and its subtree) and the algorithm stops: the optimal action sequence is R at s, L at u_2 and R at v_3, with MIN paying 9 to MAX.

Next, we assume that agents explore actions from left to right. The MAX player first explores (s, u_1), followed by the MIN player exploring (u_1, v_1), and the MAX player checking $(v_1, 2)$; then we check $(v_1, 4)$ and $\alpha(v_1) = 4$ which updates $\beta(u_1) = 4$ as well (in the worst case the MIN player loses 2 here by playing L). Then we check (u_1, v_2) and $(v_2, 8)$. At this point we stop - $\alpha(v_2) = 8$ but $\beta(u_1) = 4$! We prune both $(v_2, 3)$ and $(v_2, 0)$. Onto checking (s, u_2): we next check (u_2, v_3) and $(v_3, 3)$, $(v_3, 7)$ and $(v_3, 9)$, yielding $\alpha(v_3) = \beta(u_2) = 9$. Choosing (u_2, v_4) next, we check both $(v_4, 0)$ and $(v_4, 12)$ to get $\alpha(v_4) = 12$ and $\beta(u_2)$ remains at 9. Finally, we check $(u_2, 10)$ and nothing changes.

3. Consider the following game: we have an attacker looking at three targets: t_1, t_2 and t_3. A defender must choose which of the two targets it will guard; however, the attacker has an advantage: it can observe what the defender is doing before it chooses its move. If an attacker successfully attacks it receives a payoff of 1 and the defender gets a payoff of -1.

   (a) Model this problem as a minimax search problem. Draw out the search tree. What is the defender's payoff in this game?

   (b) Can the defender do better by randomizing? What is the defender's optimal strategy? Prove your claim.

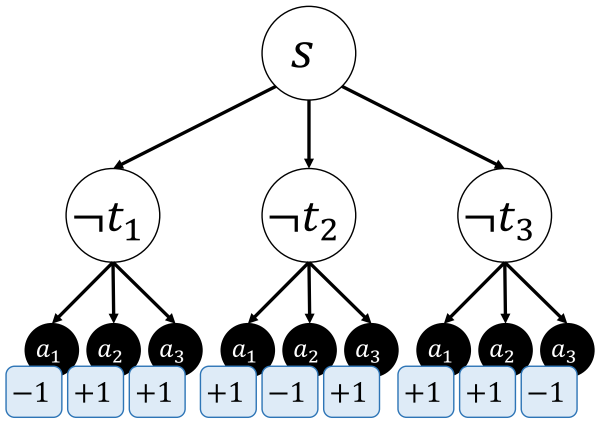

Figure 7: Defender/Attacker game tree. An action by the defender is which node it is *not* defending, given by $\neg t_i$.

Solution: The search tree is given in Figure 7 Note that the attacker can choose an action that guarantees it a payoff of 1 no matter what the defender does. However, when the defender randomizes the attacker's benefit becomes significantly smaller. By choosing the target not to defend with probability $\frac{1}{3}$ its expected loss becomes $\frac{1}{3}$. Note that when the number of targets is n, this probability is $\frac{1}{n}$.

4. Prove that the minimax algorithm shown in class outputs a subgame-perfect Nash equilibrium.

Solution: The proof is by induction on the height of the game tree, h. If $h = 1$ then the source node is a terminal node and there is nothing to prove. Suppose that for all trees at height $h - 1$ the minimax algorithm computes a subgame-perfect Nash equilibrium, and consider a node v at height h. We need to show that the minimax algorithm strategy will be a Nash equilibrium for the subtree induced by the node v. A bit of notation: given a node u, let $T(u)$ be the subtree rooted at u, and let $C(u)$ be the children of u. Assume that v is a MAX node (if it's a MIN node the proof is similar). The minimax algorithm has, by assumption, selected strategies in the subtrees rooted in the children of v such that no player can improve their welfare by switching strategies (they are Nash equilibria). Let s_1^* be the strategy outputted by the minimax algorithm for the MAX player, and let s_2^* be the strategy outputted by the minimax algorithm for the MIN player. Let s_1 be some other strategy by player 1. We need to show that $u_1(v, s_1^*, s_2^*) \geq u_1(v, s_1, s_2^*)$, where $u_1(v, \cdot, \cdot)$ is the utility that player 1 receives from the game subtree rooted at the node v. Suppose that at node v the minimax algorithm chooses the action $c^* \in C(v)$. Then we know that for any other child $c \in C(v)$, $u_1(c^*, s_1^*, s_2^*) \geq u_1(c, s_1^*, s_2^*)$. Thus, by induction hypothesis, we have that for all $c \in C(v)$:

$$u_1(v, s_1^*, s_2^*) = u_1(c^*, s_1^*, s_2^*) \geq u_1(c, s_1^*, s_2^*) \geq u_1(c, s_1, s_2^*)$$

The last inequality holds by the induction hypothesis - the strategy pair s_1^*, s_2^* induces a Nash equilibrium on the subtree rooted at c. Our claim immediately follows: some child node c_0 is the one chosen by s_1, and the payoff from playing s_1 is thus $u_1(v, s_1, s_2^*) = u_1(c_0, s_1, s_2^*) \leq u_1(c_0, s_1^*, s_2^*) \leq u_1(c^*, s_1^*, s_2^*) = u_1(v, s_1^*, s_2^*)$.

5. Prove that α-β pruning does not remove any strategies that are played in a Nash equilibrium of an extensive form game; can α-β pruning be used to find a subgame-perfect Nash equilibrium? Prove or provide a counterexample.

Solution: Suppose α-β pruning removes a subtree, say a subtree rooted in a node u_1 that's the child of a node v. Suppose that v is a MAX node; then there is some other node u_2 who's a child of v and the payoff to the MAX player from choosing u_2 is at least as high as that of choosing u_1 (that's the only reason we'll prune u_1 in the first place). In other words, let x be the payoff that player 1 gets from choosing the node u_1 from v; then it must be the case that x is no more than the payoff to player 1 from playing u_2.

Therefore, pruning u_1 will never prune a strategy that's played in a Nash equilibrium of the game overall. However, since we do not explore the entire tree, we do not specify a complete strategy, and in particular, we cannot be specifying a sub-perfect Nash equilibrium strategy.