National University of Singapore
School of Computing
CS3243 Introduction to AI

Tutorial 4: Adversarial Search

Issued: February 3, 2021 Discussion in: Week 6

Important Instructions:

- **Assignment 4** consists of **Question 3** from this tutorial.
- Your solutions for this tutorial must be TYPE-WRITTEN.
- You are to submit your solutions on LumiNUS by **Week 5, Saturday, 2359 hours**.
- Refer to LumiNUS for submission guidelines

Note: you may discuss the content of the questions with your classmates (outside your group). But each group should work out and write up ALL the solutions individually. If caught plagiarising, you may be awarded an F Grade for the module.

Please refer to the **Appendix** for defintions of:

- **Nash Equilibrium**
- **Subgame-Perfect Nash Equilibrium**

---

1. Prove that the `MINIMAX` algorithm shown in class outputs a subgame-perfect Nash equilibrium.

2. 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, $Eval(n) = 6 - 5 = 1$.

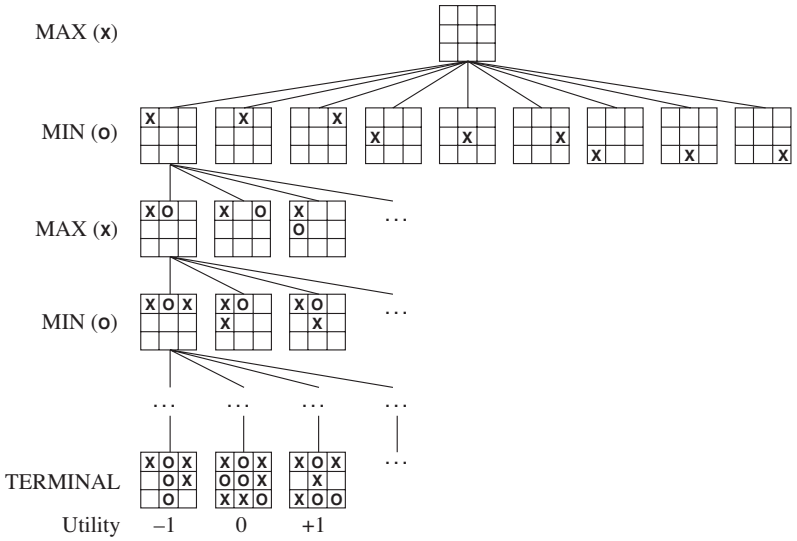

Figure 1: Search space for Tic-Tac-Toe.

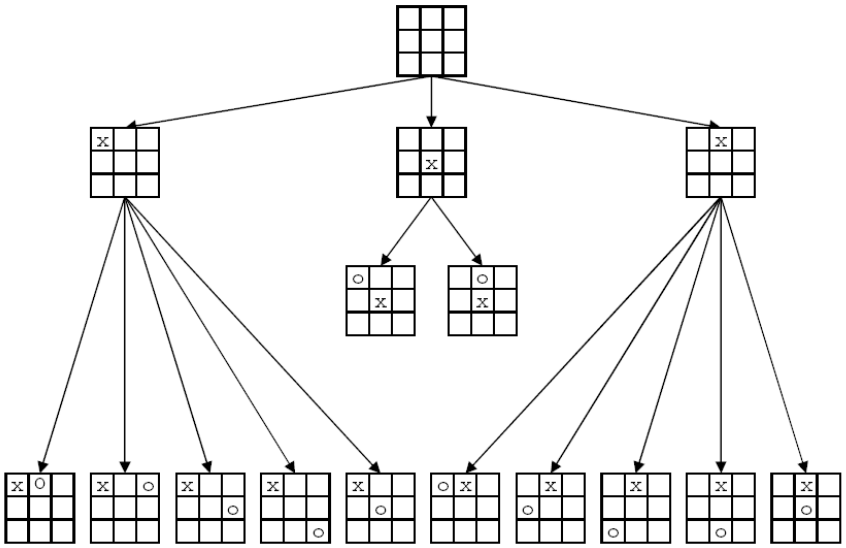

Figure 2: 2-ply deep search space

(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).

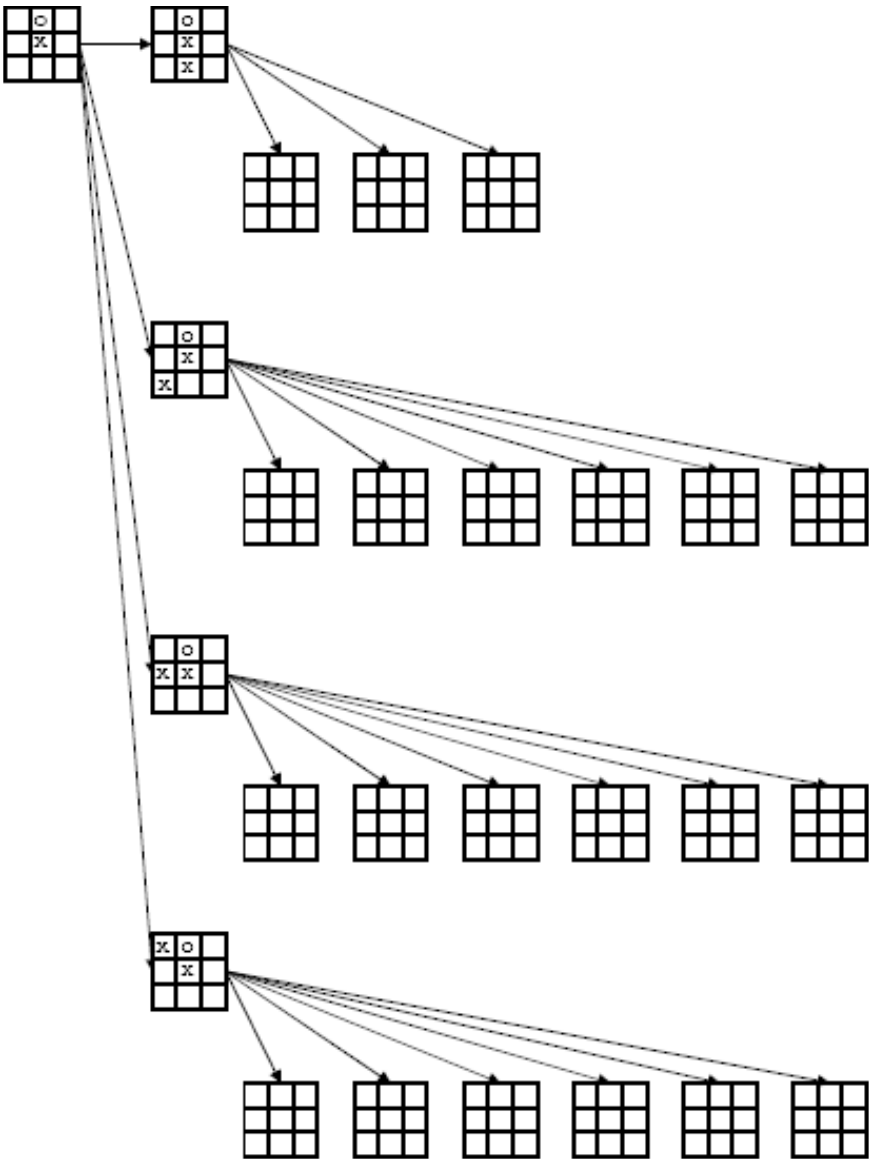

Figure 3: 3-ply deep search space

3. (**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 4).

   Consider the minimax tree given in Figure 5.

   (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) Show that the pruning is different when we instead iterate over nodes from **left to right**. Your answer should clearly indicate all nodes that are pruned under the new traversal ordering.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

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

Figure 4: α-β `PRUNING` algorithm (note that $s = state$).

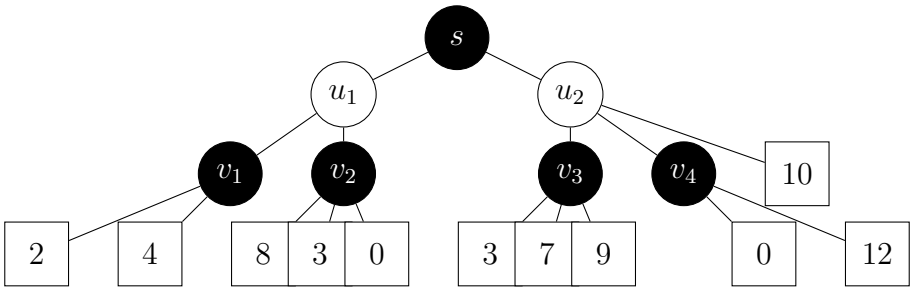

Figure 5: A minimax search tree.

4. 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.

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.

Appendix: Nash Equilibrium and Subgame-Perfect Nash Equilibrium

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 S_1, 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 $s_1 \in S_1$, $s_2 \in S_2$ 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 $s_1^* \in S_1$, $s_2^* \in S_2$ 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:

$$\begin{aligned} \forall s \in S_1 : \quad & u_1(s_1^*, s_2^*) \geq u_1(s, s_2^*); \\ \forall s' \in S_2 : \quad & u_2(s_1^*, s_2^*) \geq u_2(s_1^*, s') \end{aligned}$$

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 $s_1^* \in S_1$, $s_2^* \in S_2$ is a sub-perfect Nash equilibrium if it is a Nash equilibrium for any subtree of the original game tree.