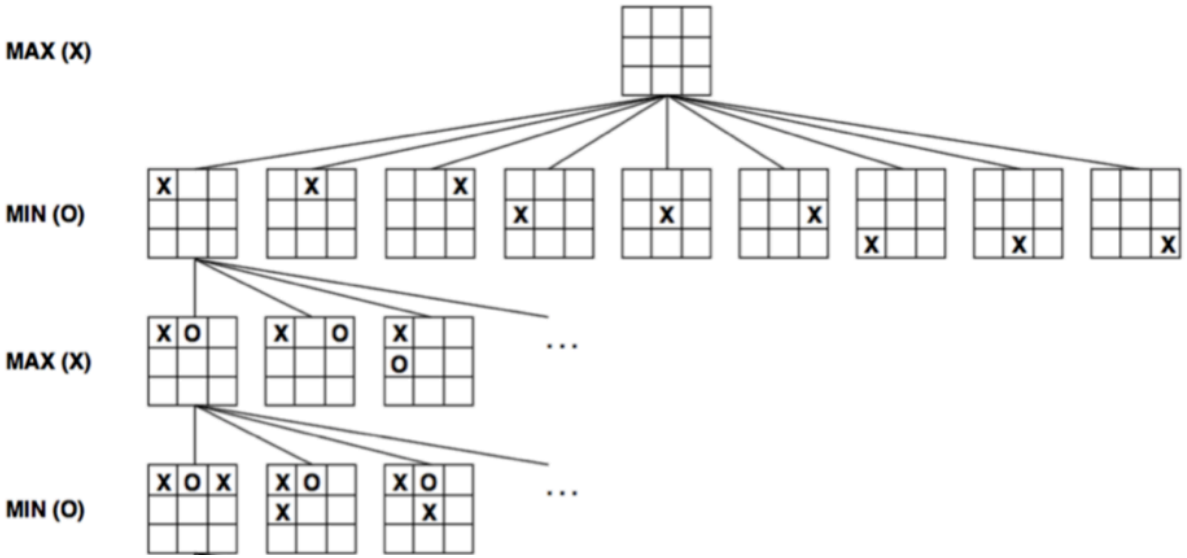
CS143: Artificial Intelligence



Adversarial Search

Drake UNIVERSITY

Adversarial Search

- "Adversarial Search" concern problems known as **games**:
  - eg, checkers, chess, go, backgammon, etc.

- We focus in particular on games that involve turn-taking and are **zero-sum**:
  - means the total payoff to players is the same for each game
  - eg, given player A vs. player B, the outcomes include:
  - **A wins (1), A loses (-1), or tie (0)**
  - A wins (1,0); B wins (0,1), or tie (0.5,0.5)

Games

- Games are interesting because they are *too hard* to solve optimally
  - eg, the search tree for a chess game involves 10^{154} possible nodes
- Consider two players: MIN and MAX
  - MAX moves first;
  - MIN moves next;
  - MAX moves again;
  - MIN follows up;
  - …
  - …
  - etc

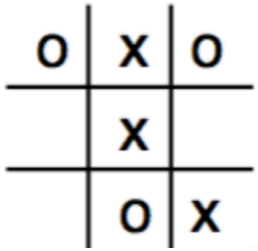
Games

- The choice of possible moves for both players gives rise to a **game tree**

- A **utility function** determines a numeric value for a game that ends in a *terminal state*

  - MAX wins (1), MIN loses (-1), or tie (0)
  - MAX wins (1,0); MIN wins (0,1), or tie (0.5,0.5)

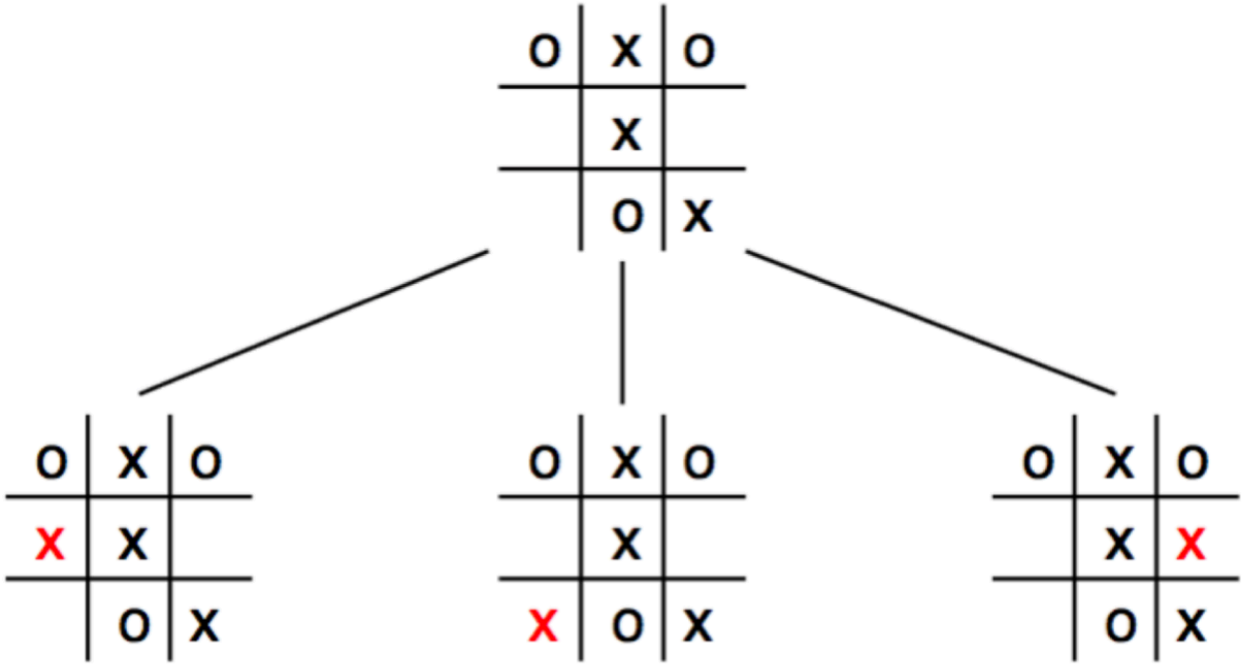
Game Tree Example



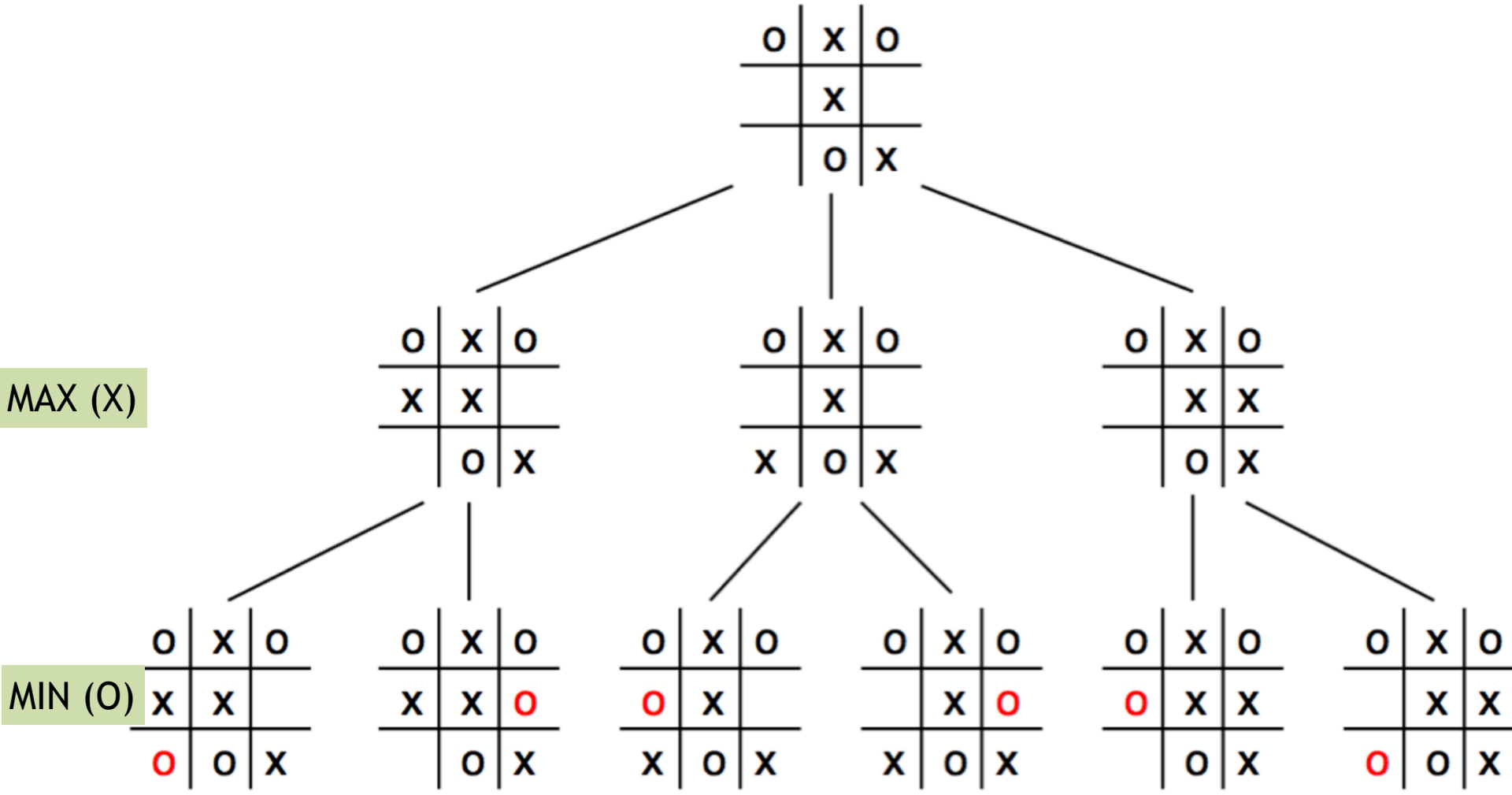
MAX (X)

Game Tree Example

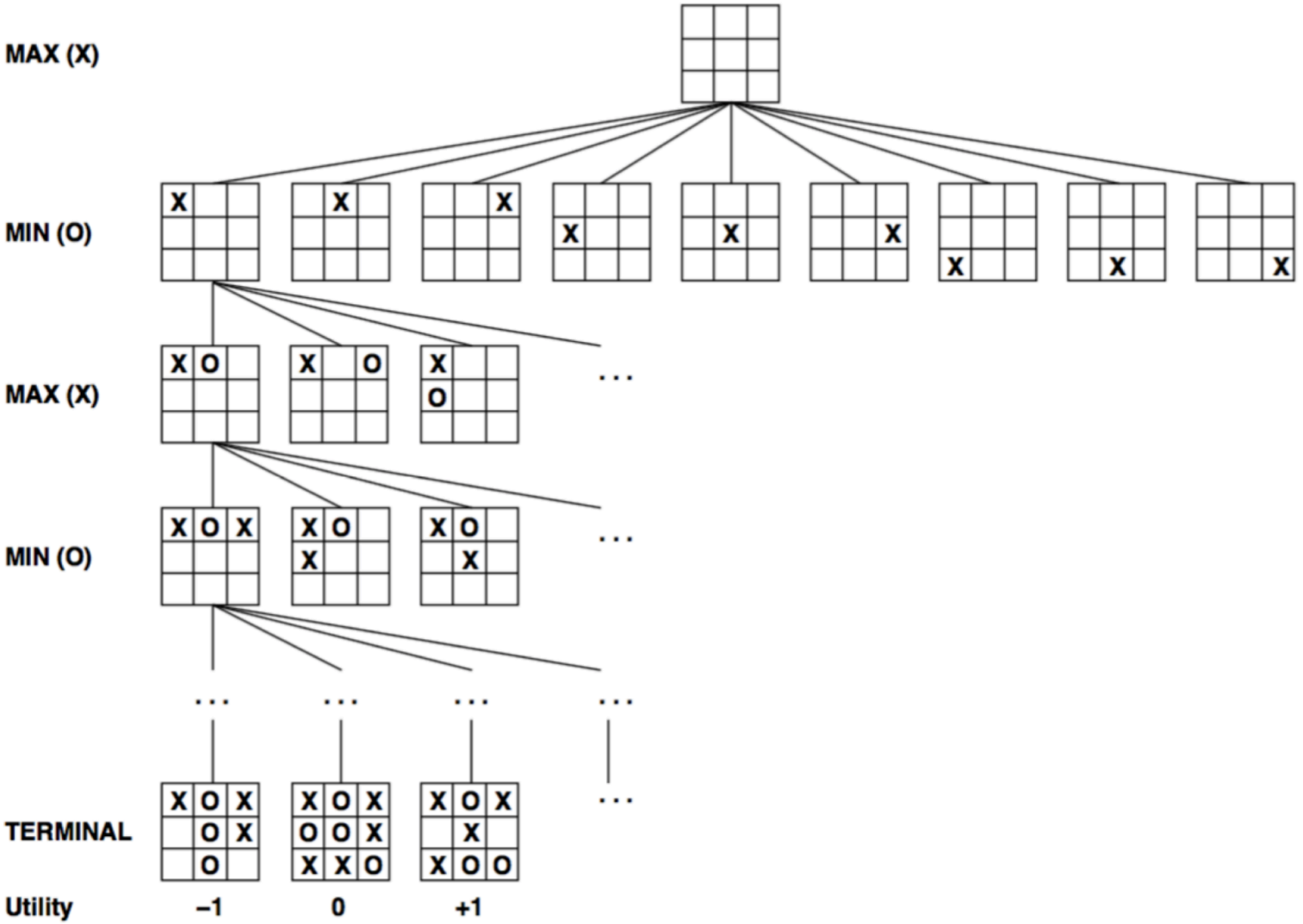
MAX (X)



Game Tree Example



Game Tree Example



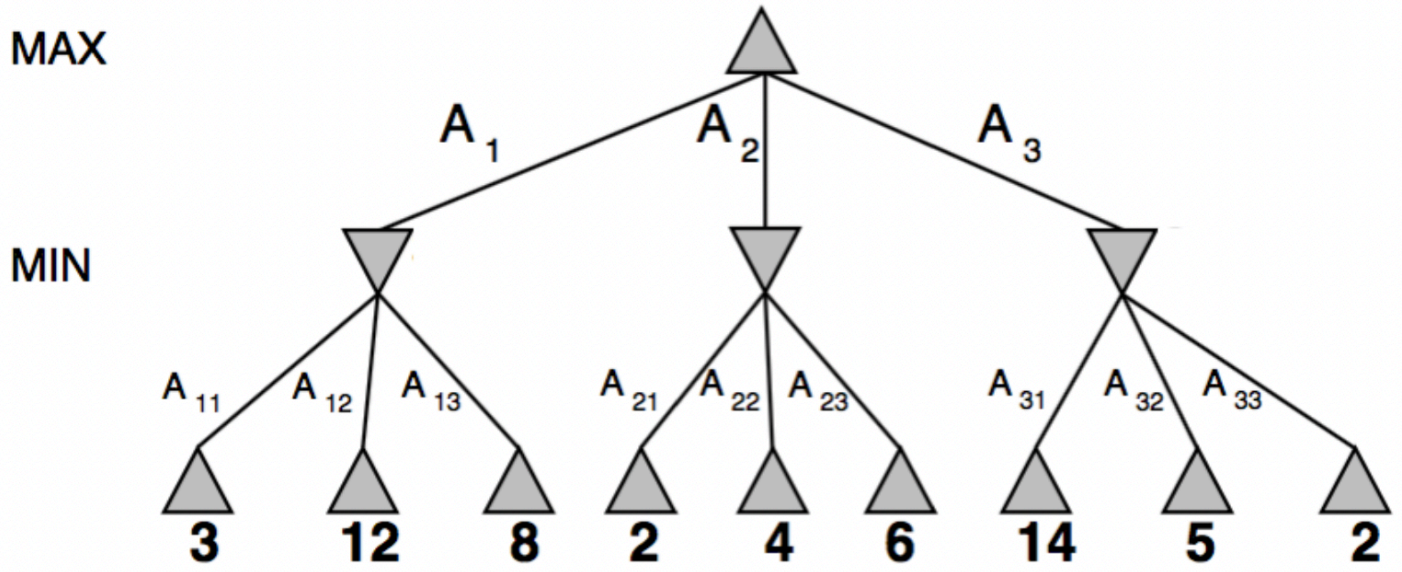
The MINIMAX algorithm

Contingent Strategies

- MAX would like to play according to a given strategy
- However, MIN has something to say about how the game unfolds
- Thus MAX must play according to a **contingent strategy**, which takes into account MIN's possible responses to MAX's moves

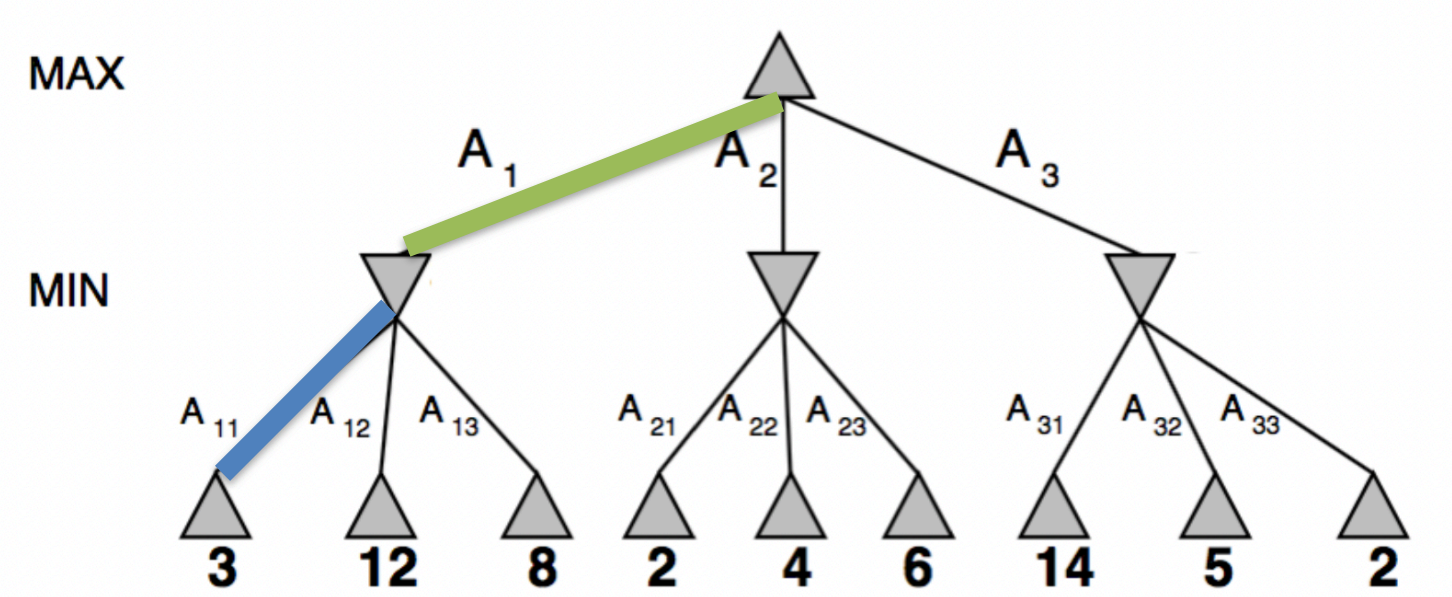
A Simple Game

- Suppose MAX and MIN play a game in which
  - MAX is trying to getting the highest possible score
  - while MIN is trying to make MAX's score as low as possible



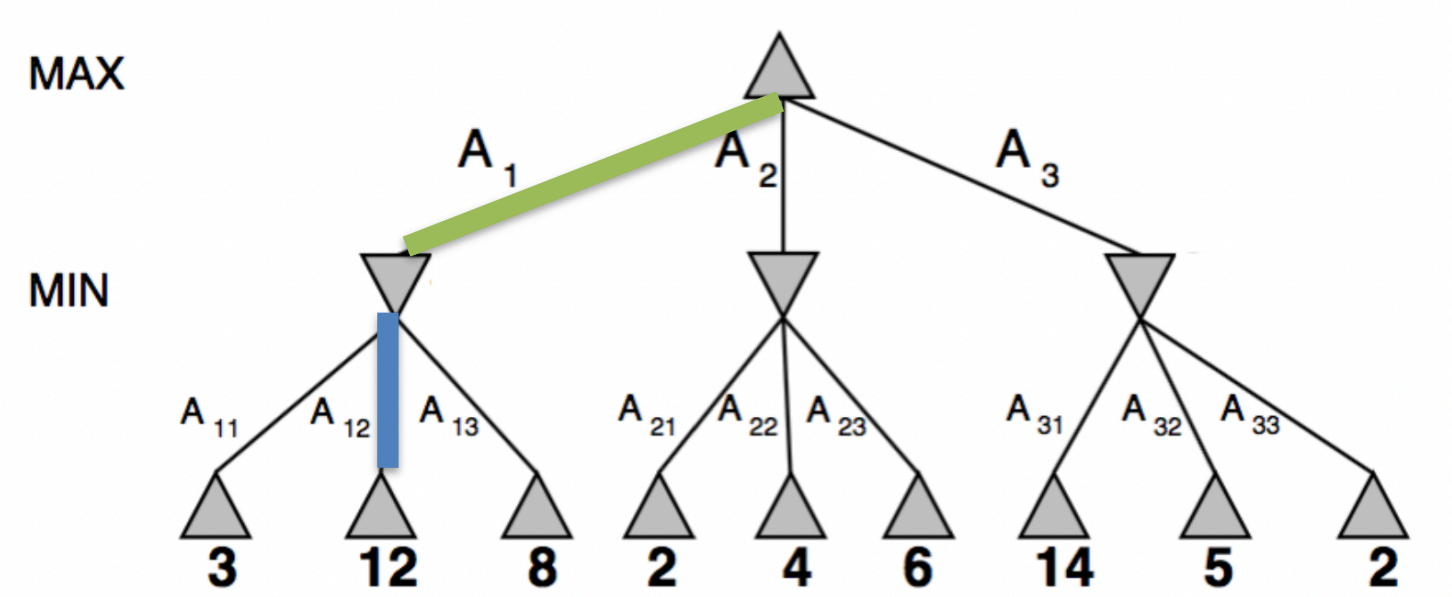
A Simple Game

- MAX can choose among A_1, A_2, and A_3
- Once MAX makes a choice, say A_i, MIN chooses between A_{i1}, A_{i2} and A_{i3}



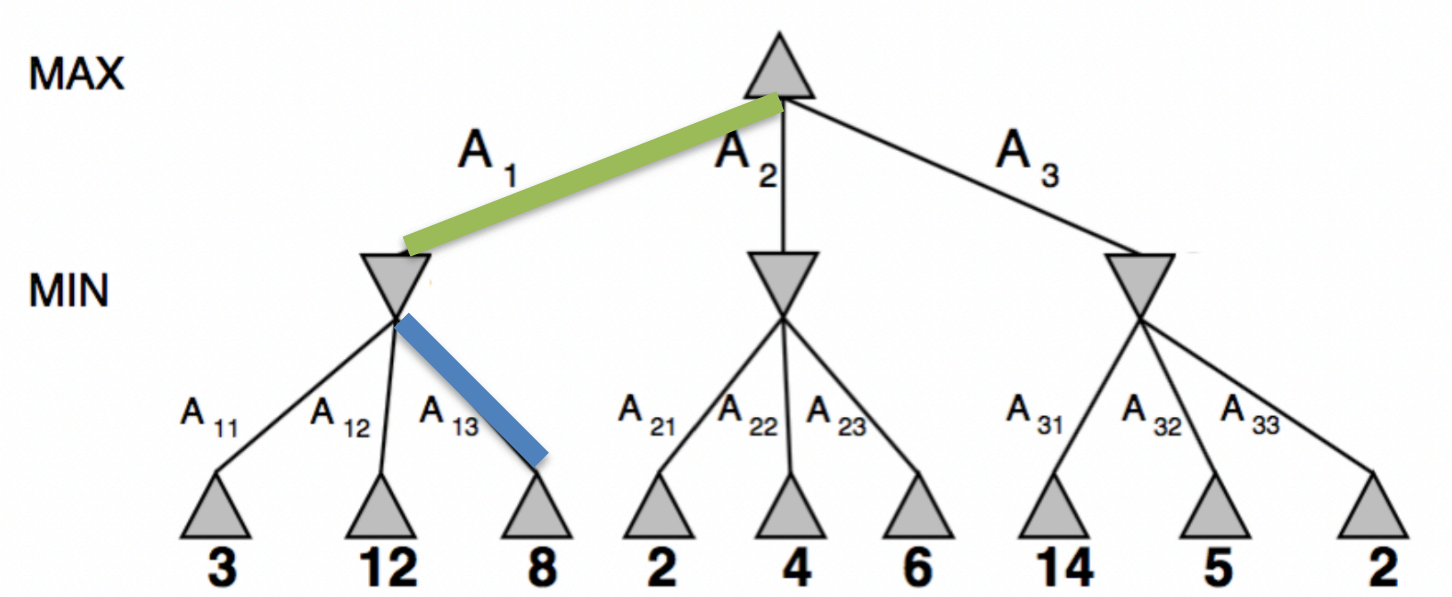
A Simple Game

- MAX can choose among A_1, A_2, and A_3
- Once MAX makes a choice, say A_i, MIN chooses between A_{i1}, A_{i2} and A_{i3}



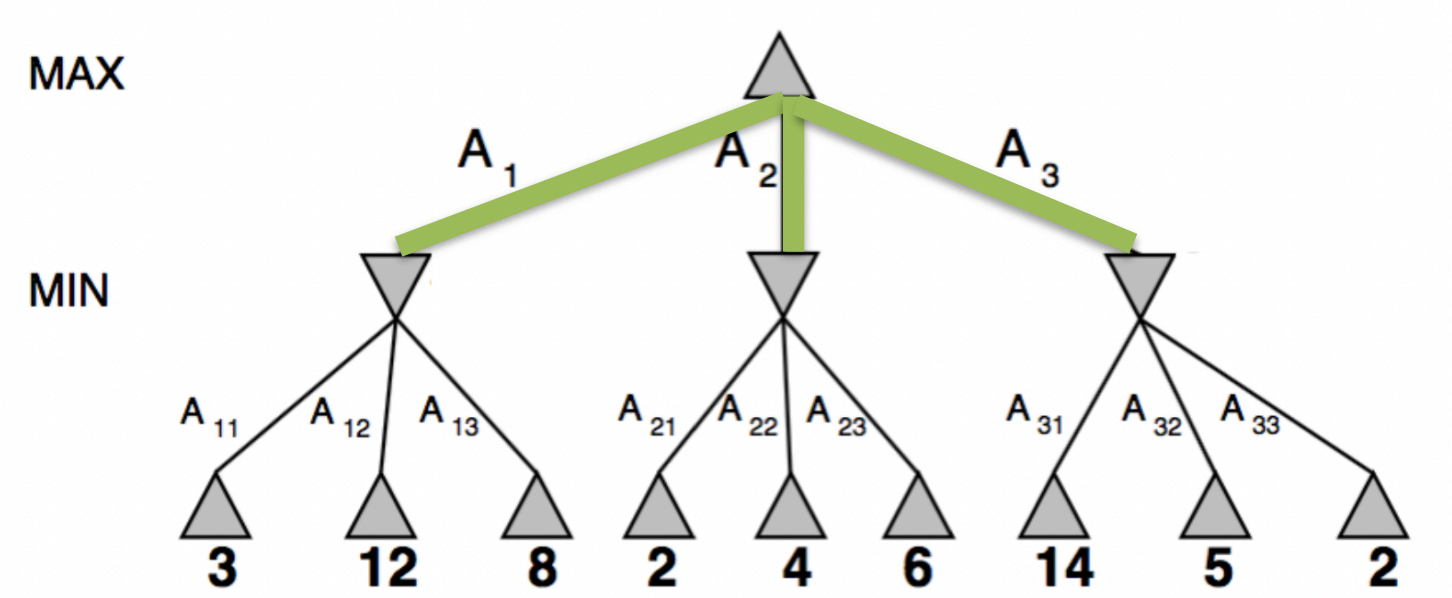
A Simple Game

- MAX can choose among A_1, A_2, and A_3
- Once MAX makes a choice, say A_i, MIN chooses between A_{i1}, A_{i2} and A_{i3}



A Simple Game

- What should MAX choose (among A_1, A_2, and A_3)?



MINIMAX Algorithm

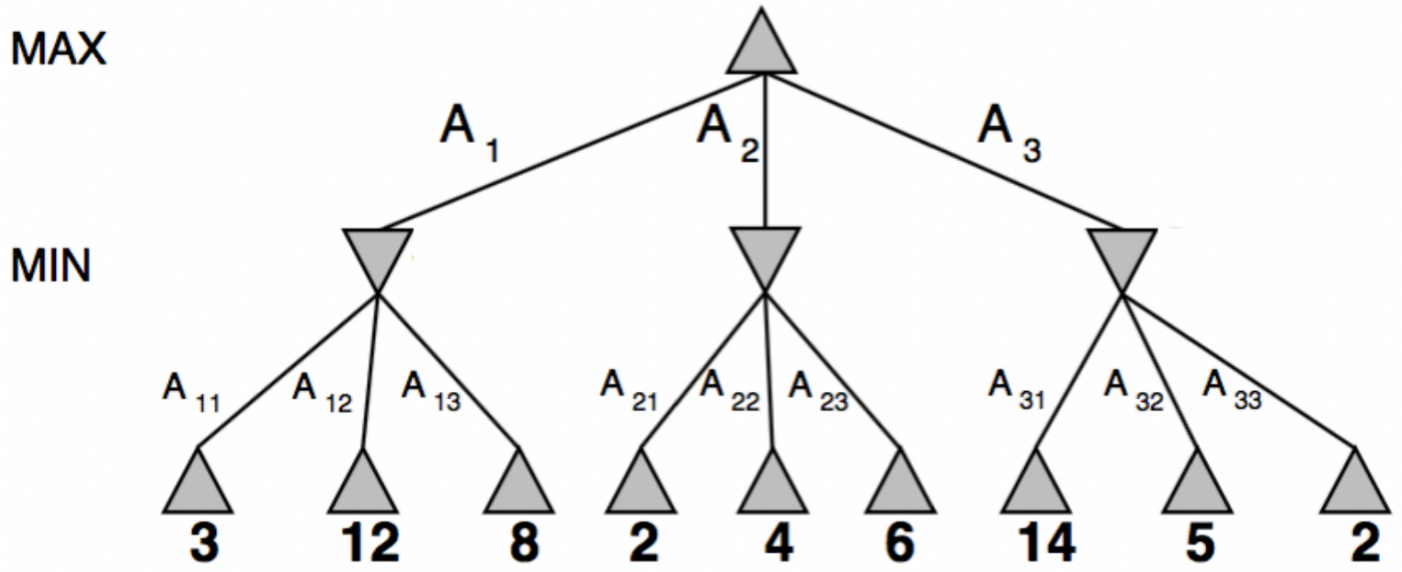
- Idea: At each turn, choose to move to the position with best **minimax** value
  - aim for the best achievable payoff against the best play of the opponent

- Evaluate the game tree's utility function value for all possible moves
  - start from the bottom-up

- This results in perfect play for deterministic, perfect-information games

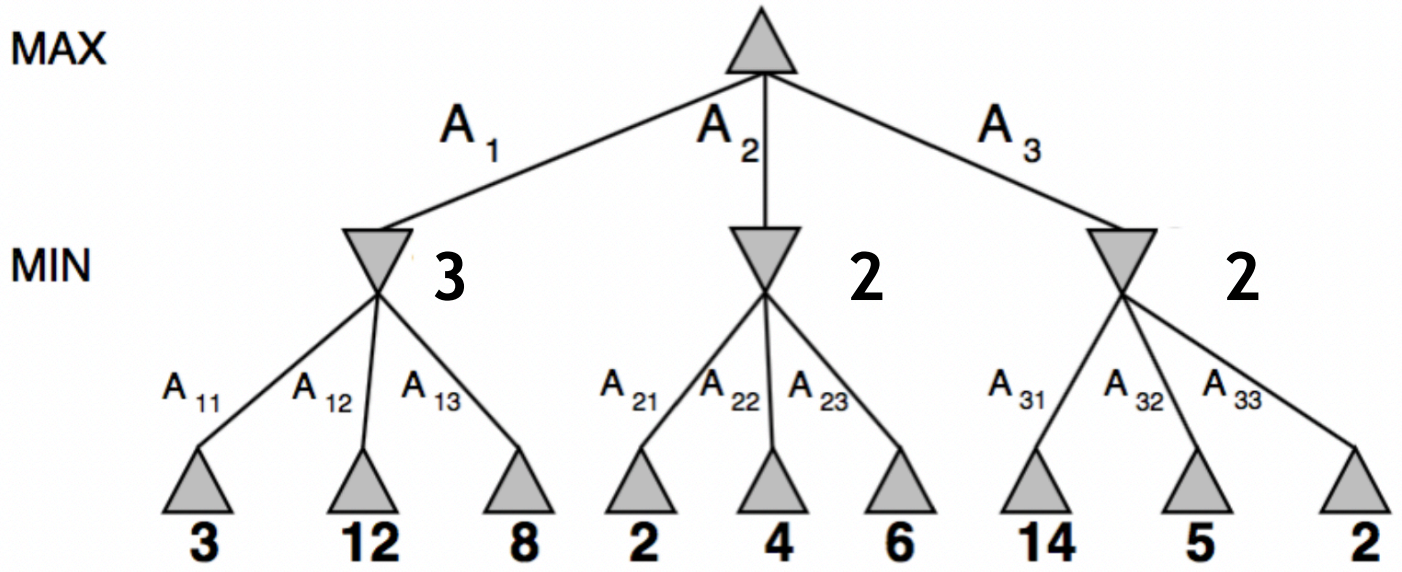
MINIMAX Algorithm

- Example: MAX and MIN play our simple game



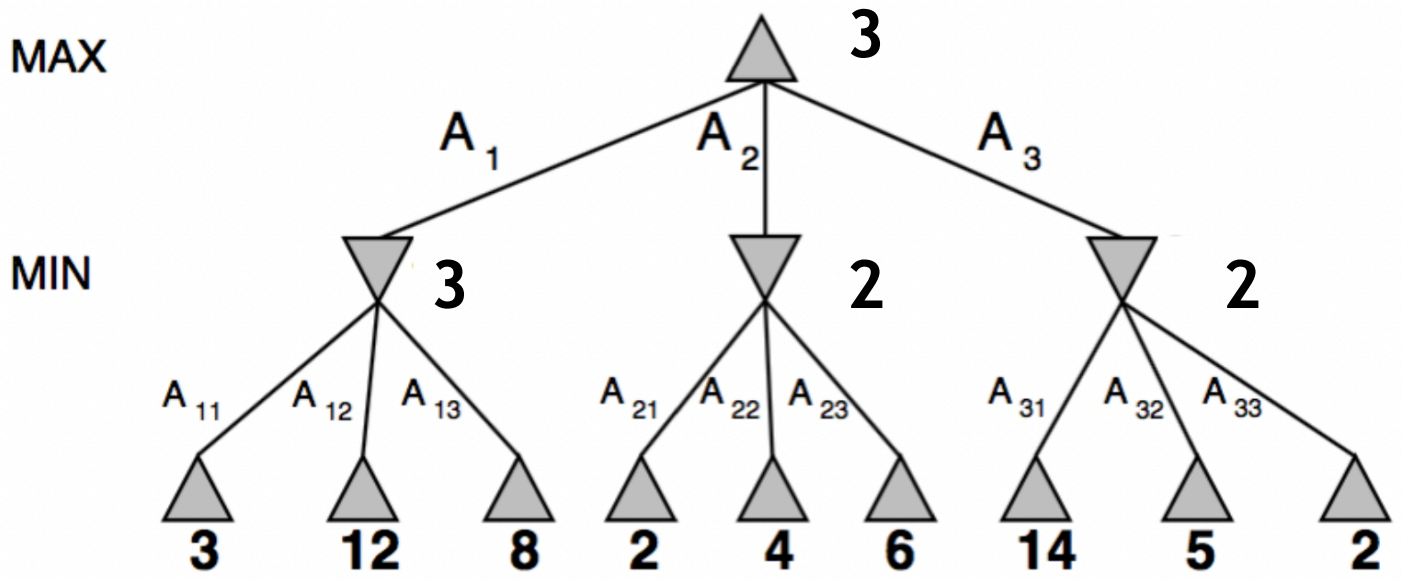
MINIMAX Algorithm

- Example: MAX and MIN play our simple game



MINIMAX Algorithm

- Example: MAX and MIN play our simple game



Assumption in MINIMAX Algorithm

- The minimax value of a node is MAX's utility of being in the corresponding state, but only assuming that both MAX and MIN play optimally from there to the end of the game
  - That is, MAX's strategy takes into account the worst-case scenario given by MIN's moves

- What happens if MIN does not play optimally?
  - One can show that MAX in fact does even better

Tile Sliding Game

- The two players, Player A and Player B, take turns
- Player A moves first

- Player B moves next

Tile Sliding Game

- During A's turn, they can do one of the followings
  - stay put
  - move right one space

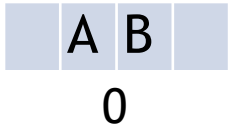
- During B's turn, they can do one of the followings
  - stay put
  - move left one space

- If the players occupy adjacent spaces, then the player may jump over the opponent to the next open space in the respective legal direction

Game Ending Conditions

- Condition 1: One of the players reaches the opposite end of the board (the player reaching the other end is the winner)
  - A wins if she reaches to the right-most position
  - B wins if he reaches to the left-most position

- Condition 2: Both players stay put in successive turns (this is a draw)



Game Ending Conditions

- Condition 3:
  The game reaches the following state:

+1

Then B stays put and A wins

or the game reaches the following state:

-1

Then A stays put and B wins

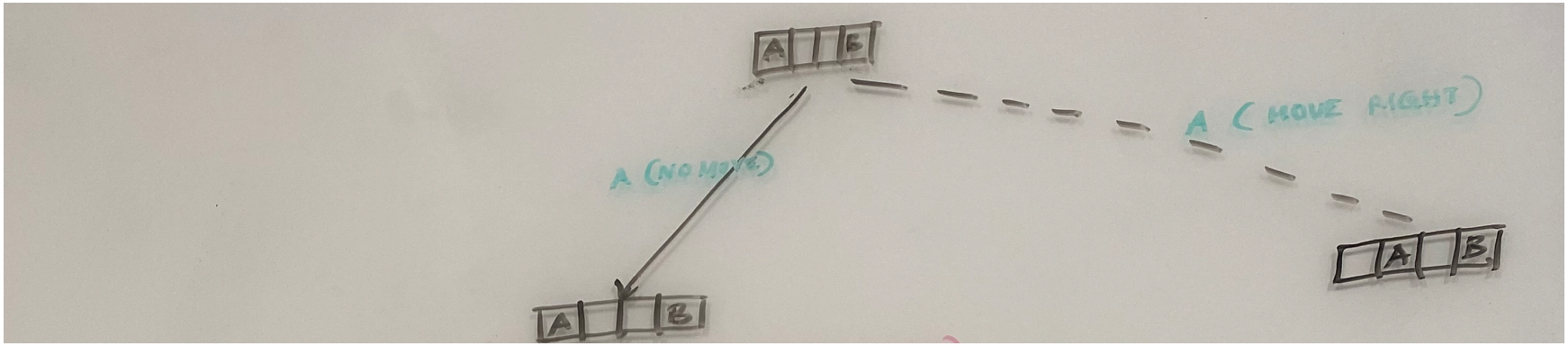
Class Activity

Tile Sliding Game Activity

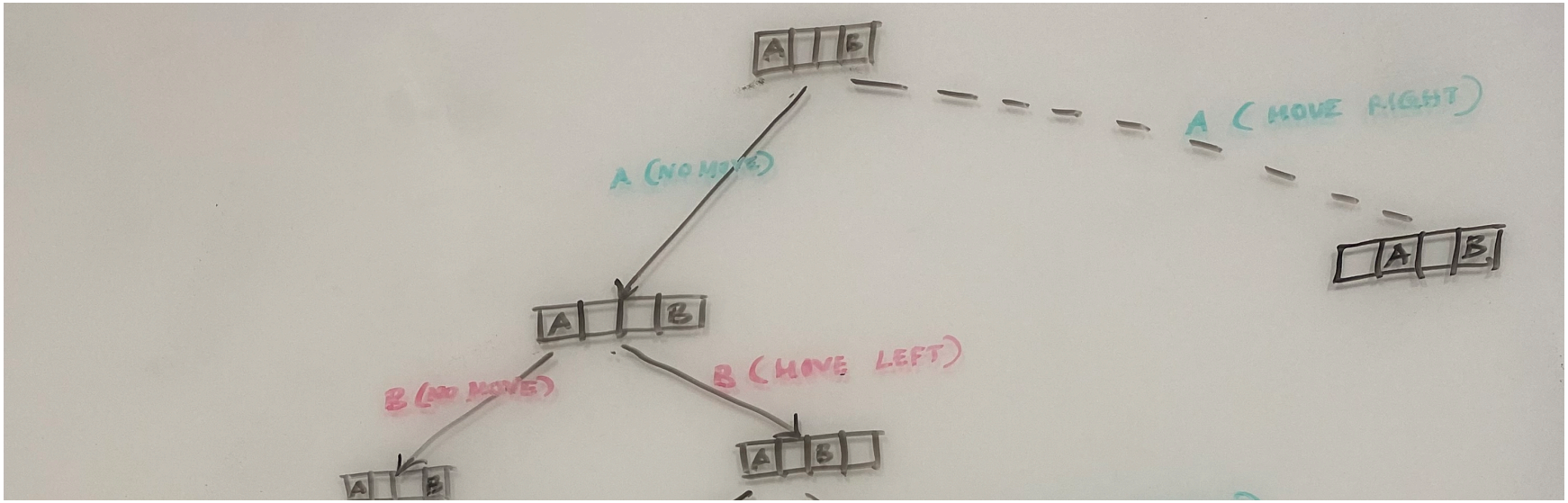
- From the above configuration, play a game with a partner and draw the game tree

Tile Sliding Game Activity

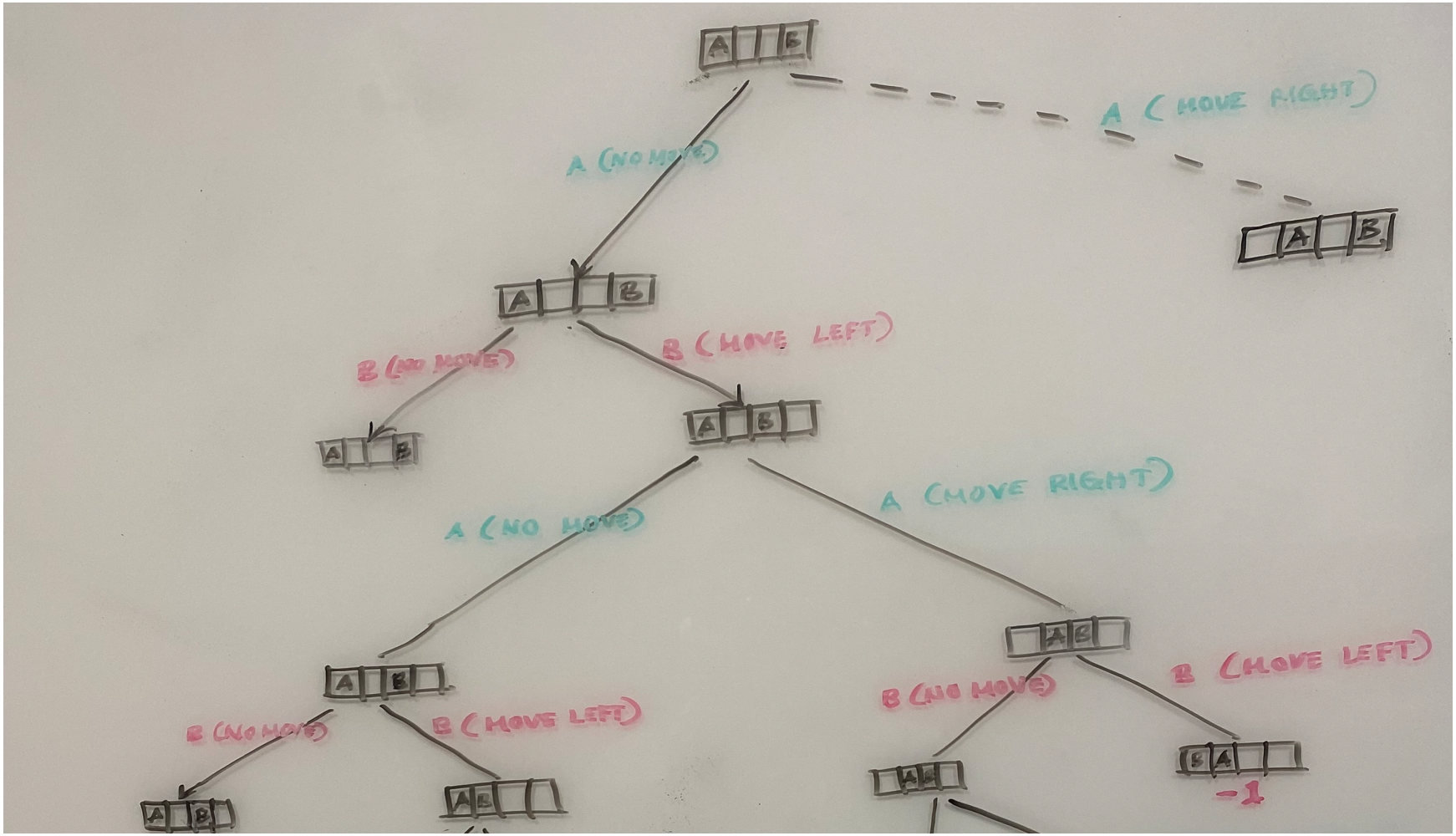
Tile Sliding Game Activity



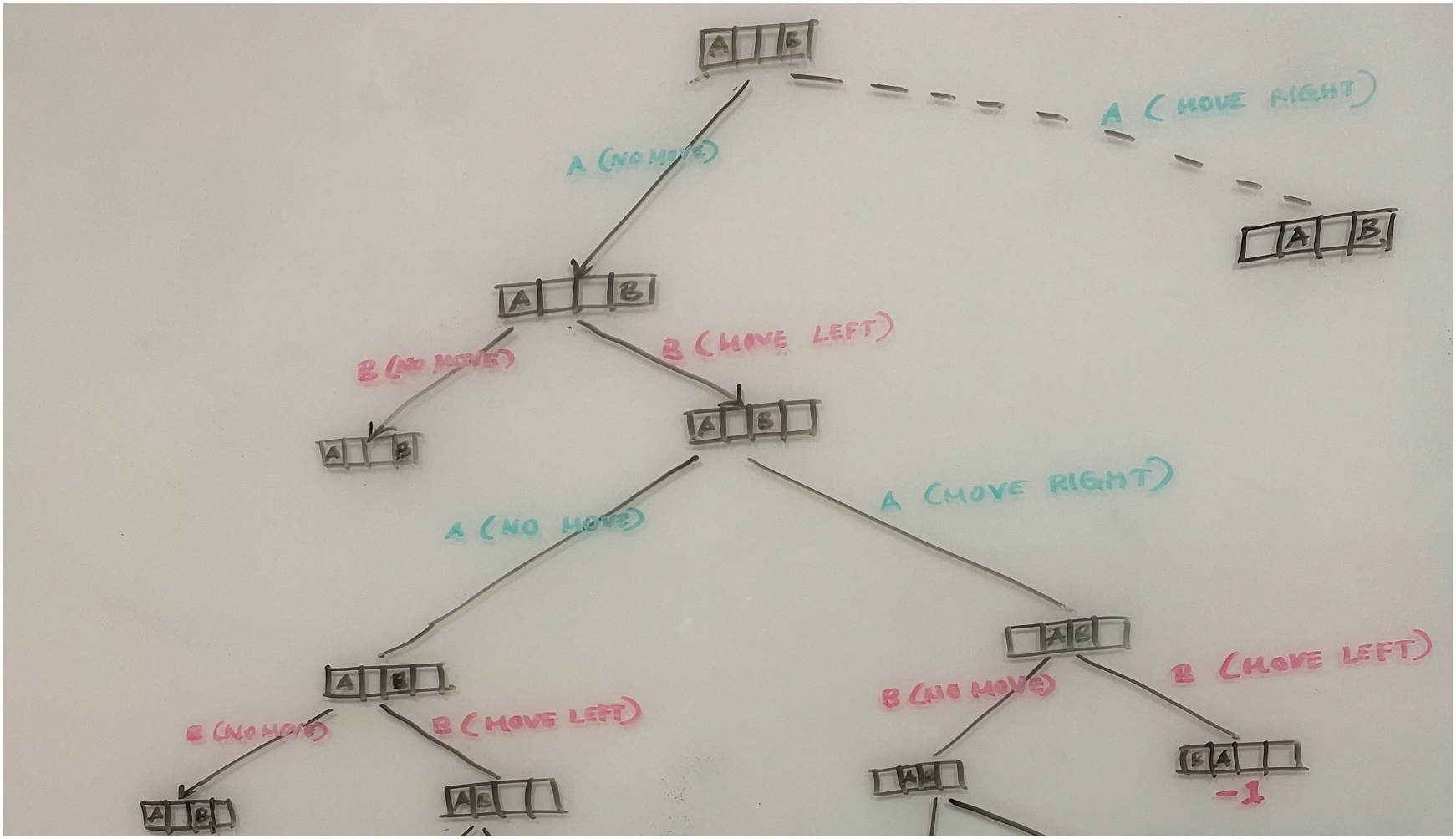
Tile Sliding Game Activity



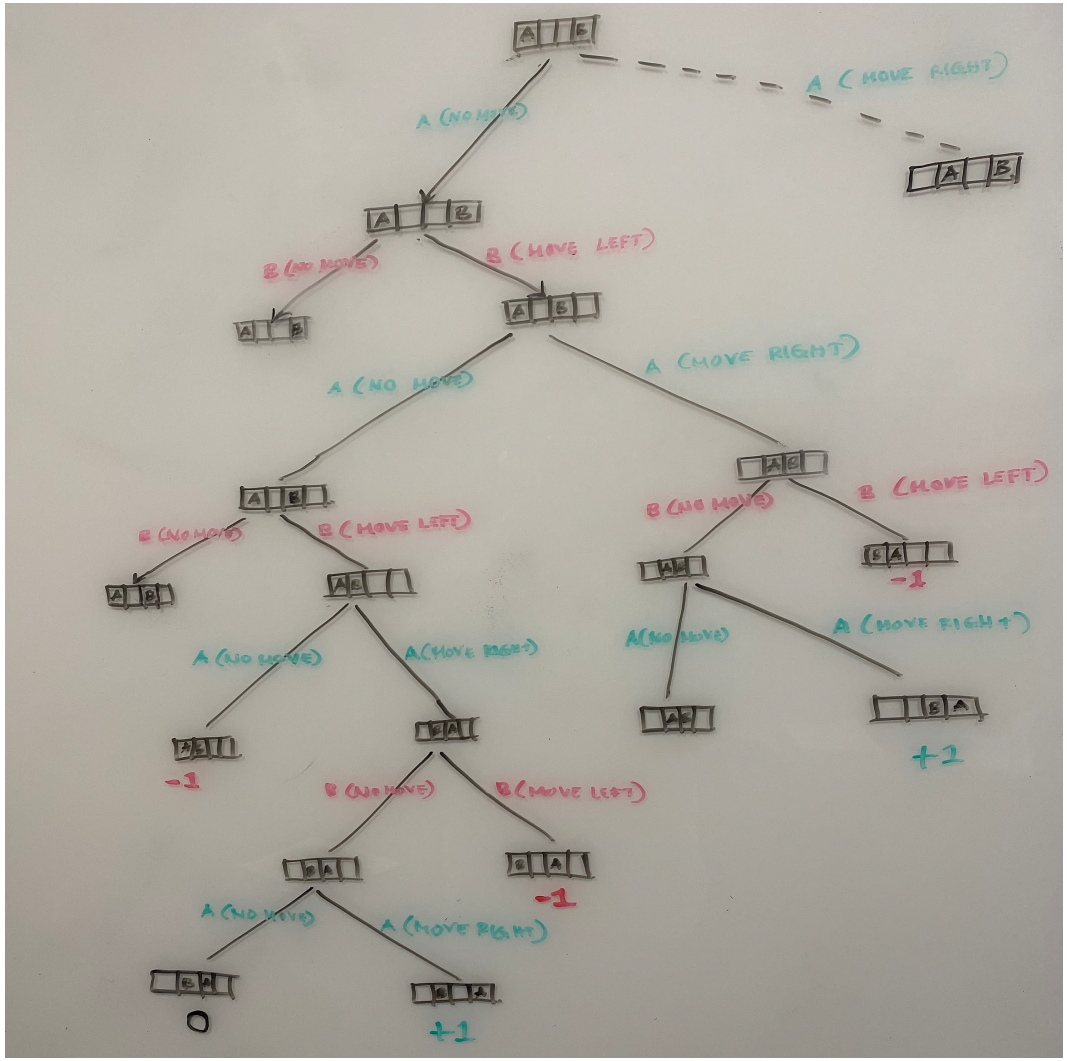
Tile Sliding Game Activity



Tile Sliding Game Activity



Tile Sliding Game Activity



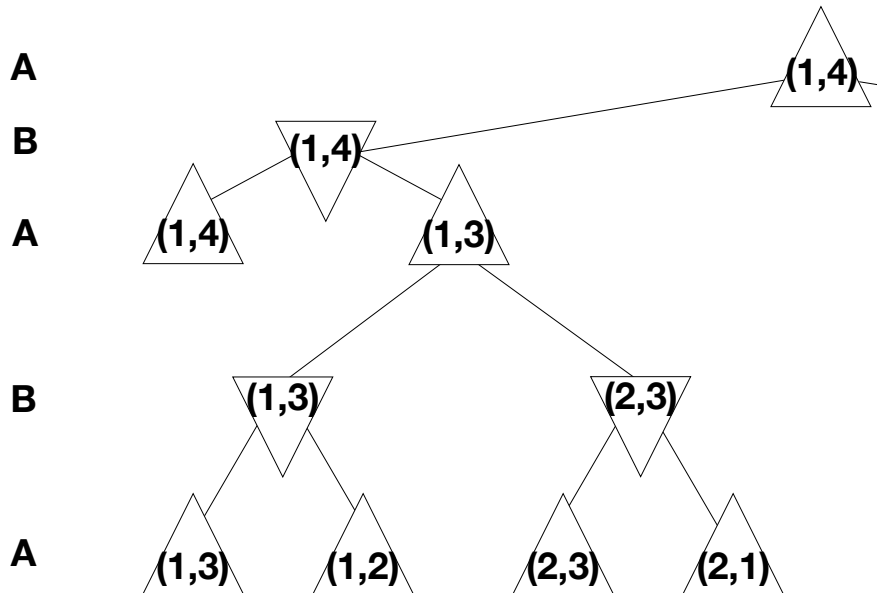
Tile Sliding Game Activity

- From the above configuration, play a game with a partner and draw the game tree
  - Can either player guarantee victory?
  - Can one of the players force a tie?
  - How do you tell this from the game tree?
  - What move should A make on her first move?

Tile Sliding Game Activity

- Simplified representation of the game states:
  - You can represent each state of this game as (n_A, n_B)
  - n_A, n_B are both numbers between 1 and 4
  - n_A is the position of the Player A
  - n_B is the position of the Player B

Class Activity



Properties of MINIMAX Algorithm

- Minimax performs a full D*epth-First Search (DFS)* of the game tree
- Suppose the maximum depth of the search tree is m and the number of choices at each step is at most b
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games. Exact solution completely infeasible

Issues

- The issue with minimax search is the number of game states it has to examine is exponential in the depth of the tree
- Can we feasibly look at every node in the tree?
- How do we find shortcuts to save computation?
- We will consider these questions going forward, but for the time being, we will consider examples of the minimax algorithm in action