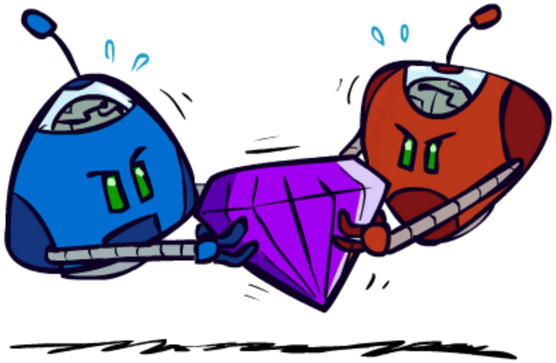
CMSC 471: Games

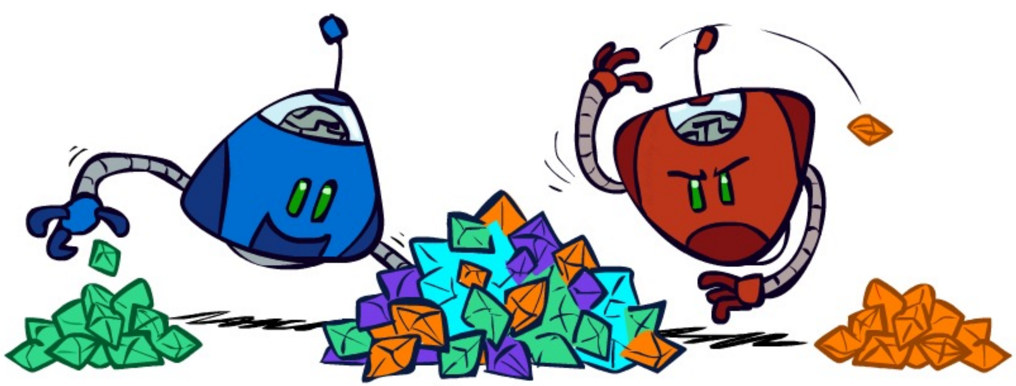
KMA Solaiman

ksolaima@purdue.edu

Slides courtesy Tim Finin and Frank Ferarro. Some material adopted from notes by Andreas Geyer-Schulz and Chuck Dyer,
Some materials adopted from slides by Dan Klein and Pieter Abbeel at UC Berkeley [http://ai.berkeley.edu]

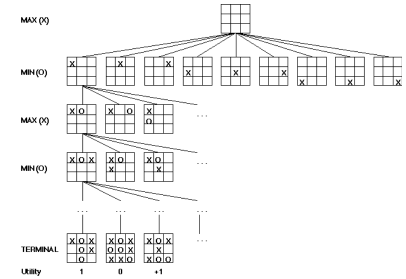
Zero-Sum Games





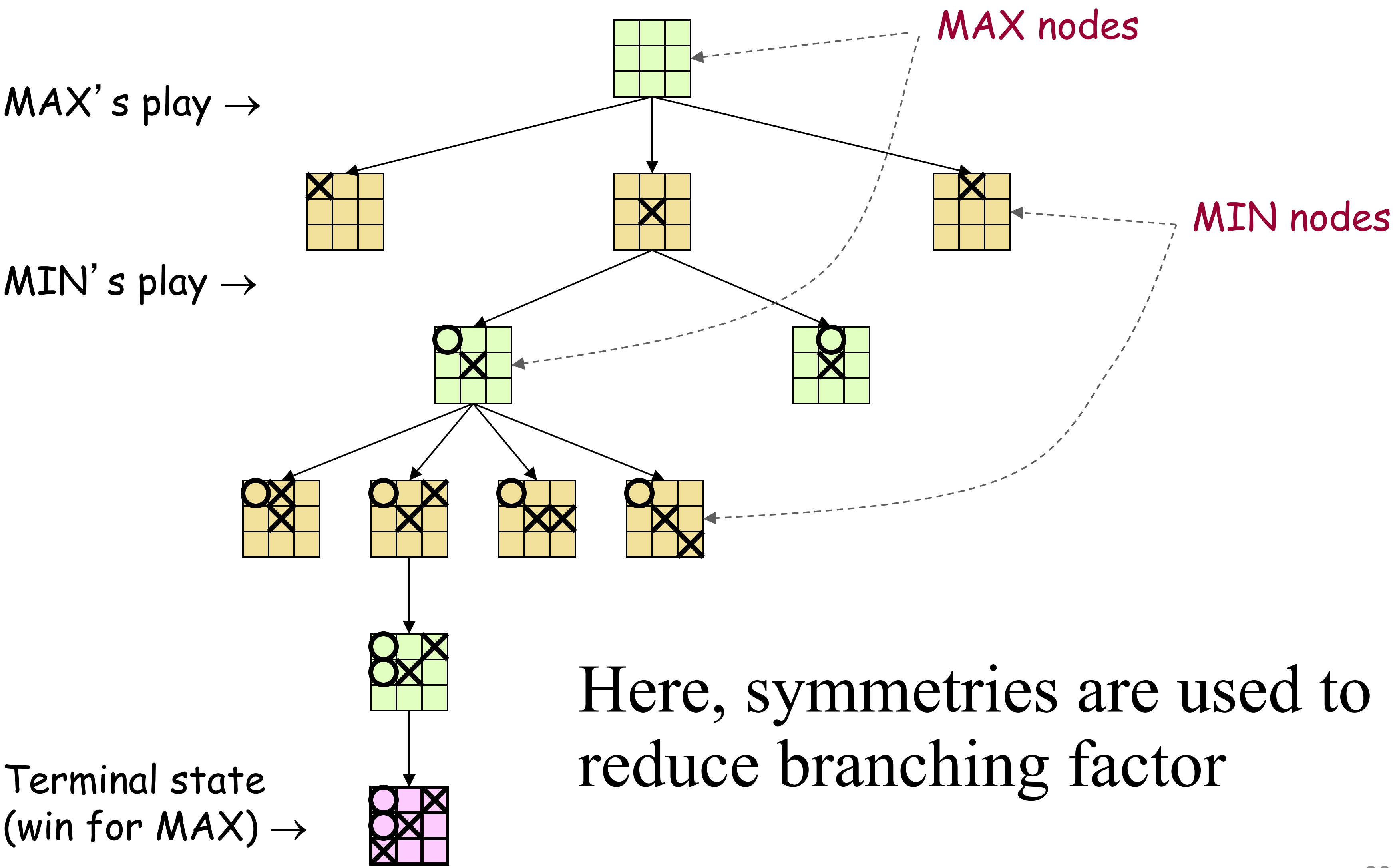
- Zero-Sum Games
  - Agents have opposite utilities (values on outcomes)
  - **Lets us think of a single value that one maximizes and the other minimizes**
  - Adversarial, pure competition
- General Games
  - Agents have independent utilities (values on outcomes)
  - Cooperation, indifference, competition, and more are all possible
  - More later on non-zero-sum games

Game trees



- Problem spaces for typical games are trees
- Root node is current board configuration; player must decide best single move to make next
- **Static evaluator function** rates board position **f(board):**real, > 0 for me; < 0 for opponent
- Arcs represent possible legal moves for a player
- If **my turn** to move, then root is labeled a "**MAX**" node; otherwise it's a "**MIN**" node
- Each tree level's nodes are all MAX or all MIN; nodes at level i are of opposite kind from those at level i+1

Game Tree for Tic-Tac-Toe



Here, symmetries are used to reduce branching factor

Minimax Algorithm

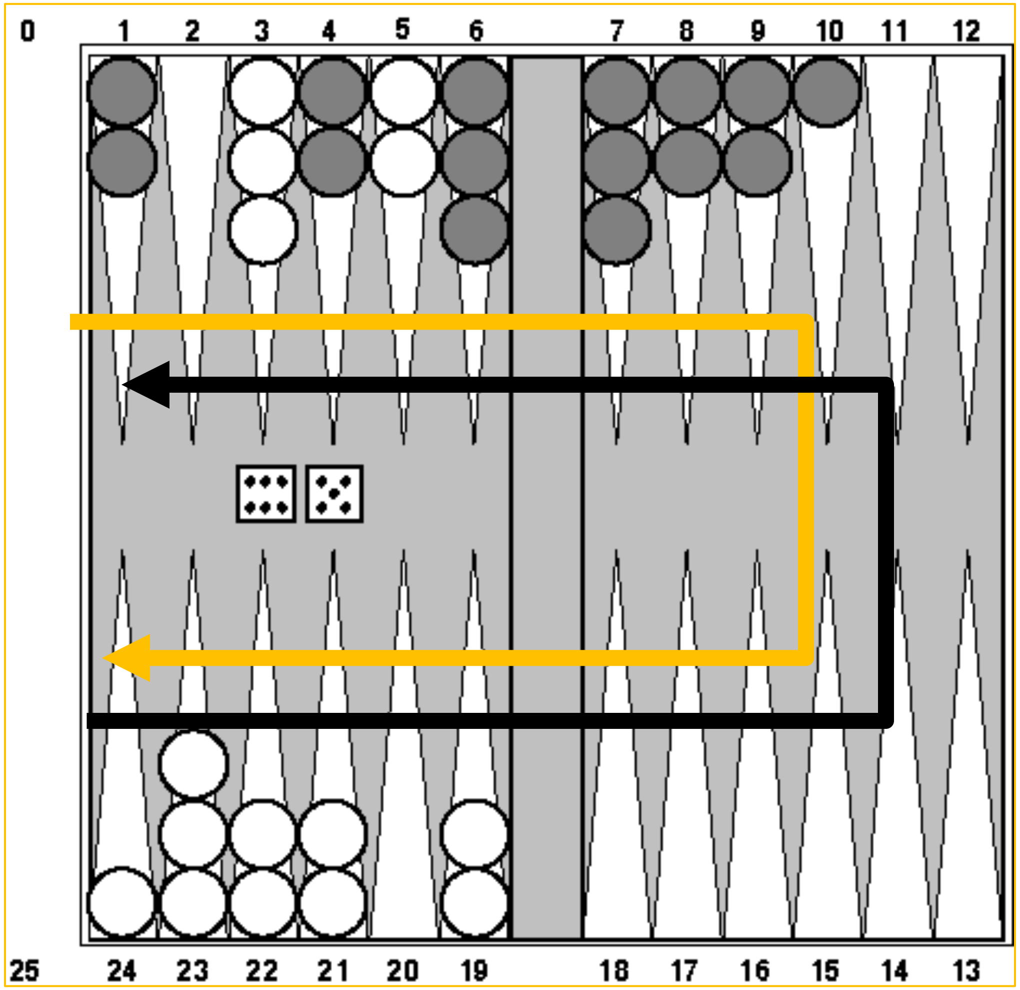
1. Create MAX node with current board configuration
2. *Expand nodes to some* ***depth*** *(a.k.a.* ***plys****) of* ***lookahead*** *in game*
3. Apply evaluation function at each **leaf** node
4. ***Back up*** values for each non-leaf node until value is computed for the root node
   - At MIN nodes: value is **minimum** of children's values
   - At MAX nodes: value is **maximum** of children's values
5. Choose move to child node whose backed-up value determined value at root

Stochastic Games

- In real life, unpredictable external events can put us into unforeseen situations
- Many games introduce unpredictability through a random element, such as the throwing of dice
- These offer simple scenarios for problem solving with adversaries and uncertainty

Example: Backgammon

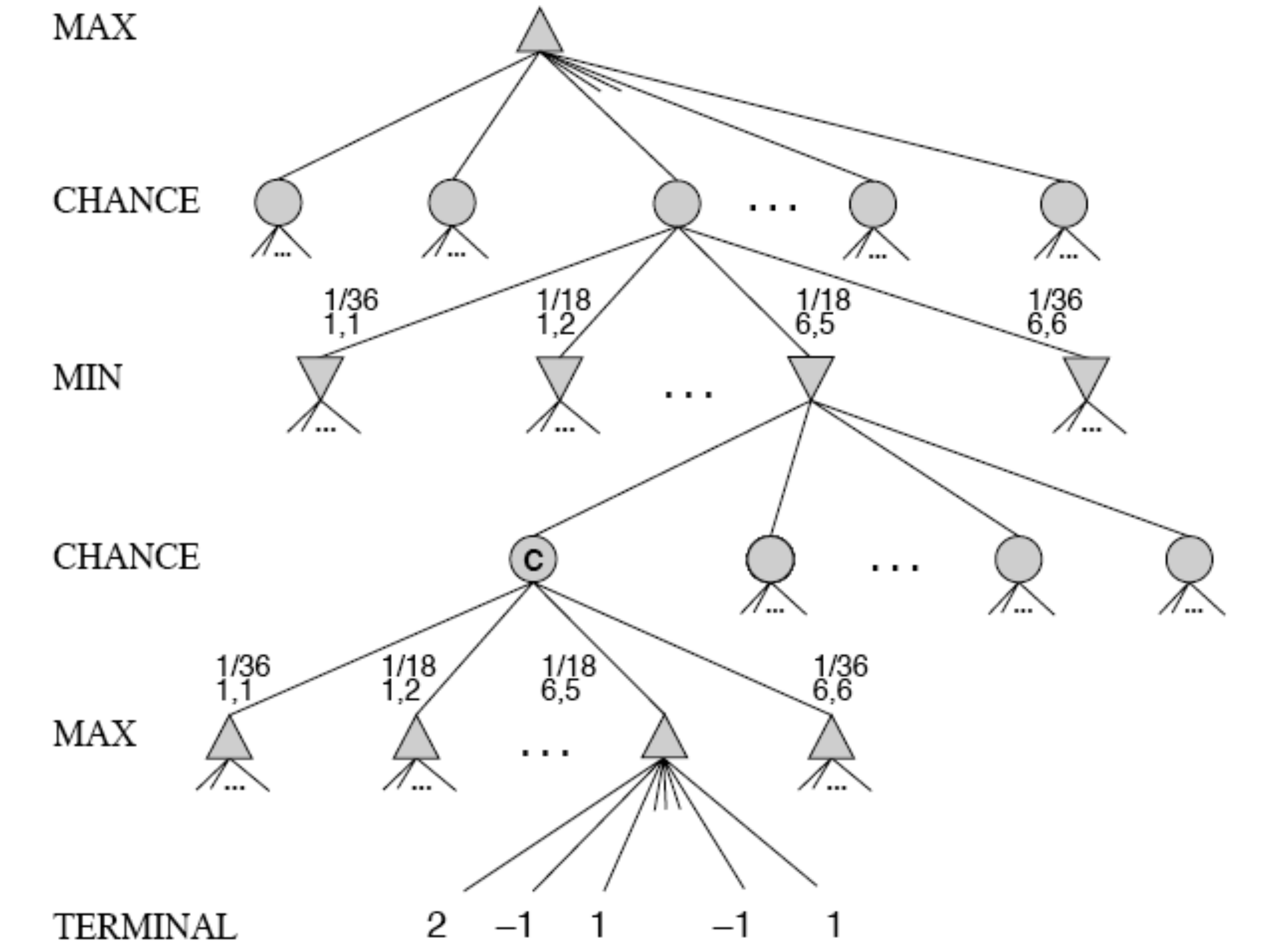
- Popular two-player game with uncertainty
- Players roll dice to determine what moves can be made
- White has just rolled 5 & 6, giving four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16
- Good for exploring decision making in adversarial problems involving skill **and** luck



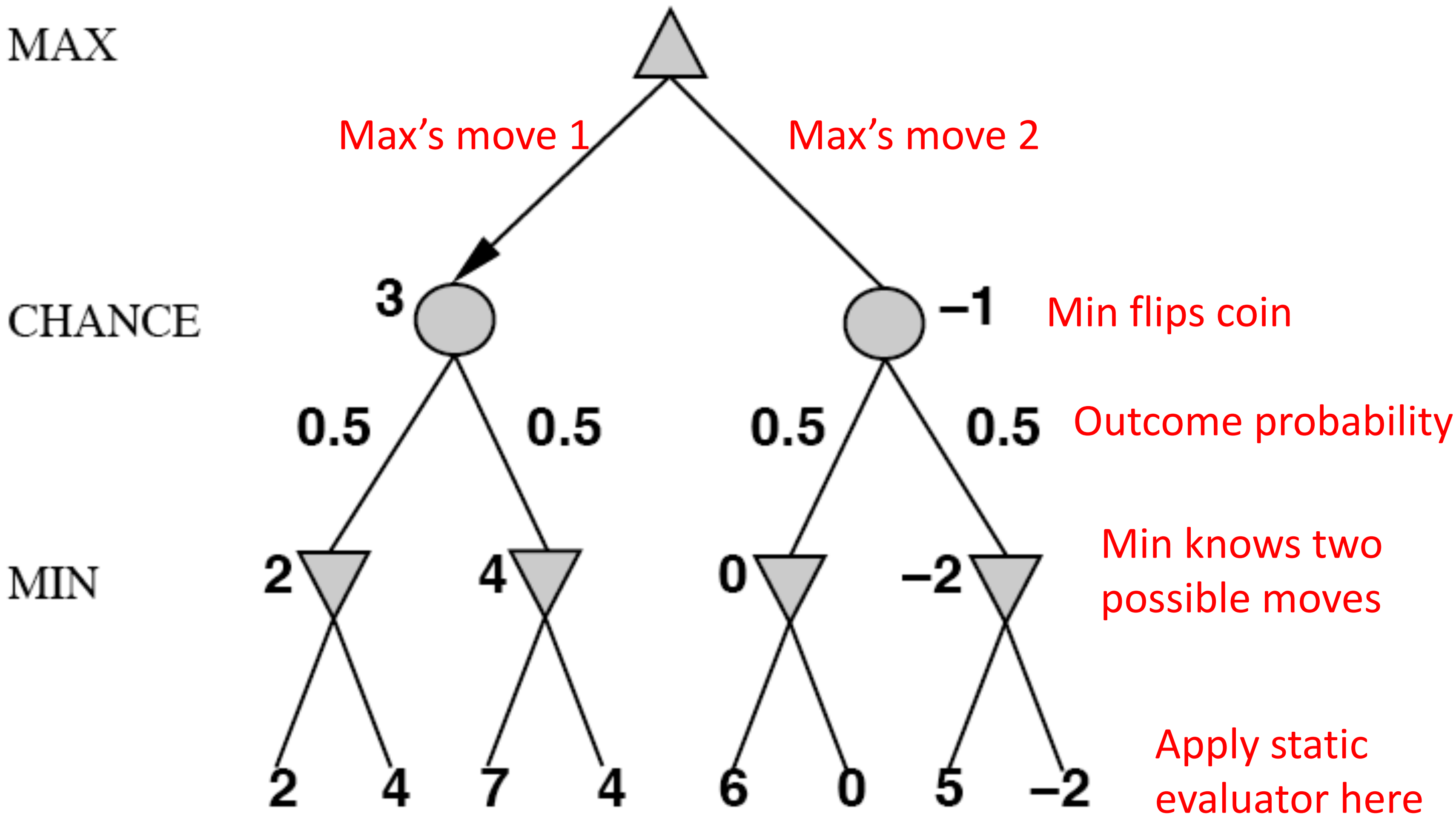
Why can't we use MiniMax?

- Before a player chooses a move, she rolls dice and only then knows exactly what moves are possible
- The immediate outcome of each move is also known
- But she does not know what moves she or her opponent will have available in the future
- Need to adapt MiniMax to handle this

MiniMax trees with Chance Nodes



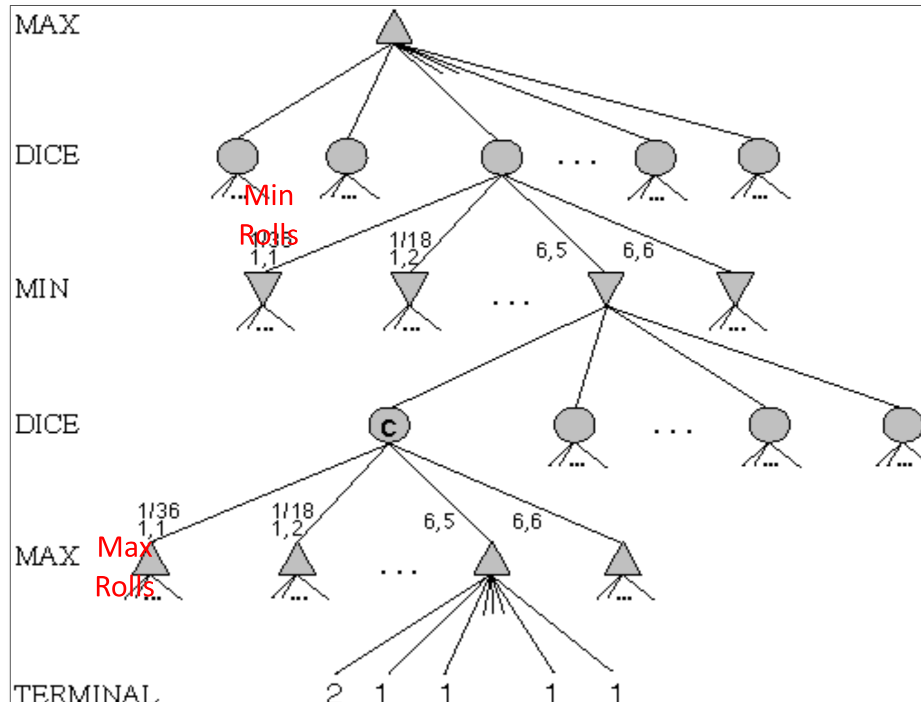
Understanding the notation



Board state includes chance outcome determining available moves

Game trees with chance nodes

- Chance nodes (circles) represent random events
- For random event with N outcomes, chance node has N children, each with a probability
- 2 dice: 21 distinct outcomes
- Use minimax to compute values for MAX and MIN nodes
- Use expected values for chance nodes



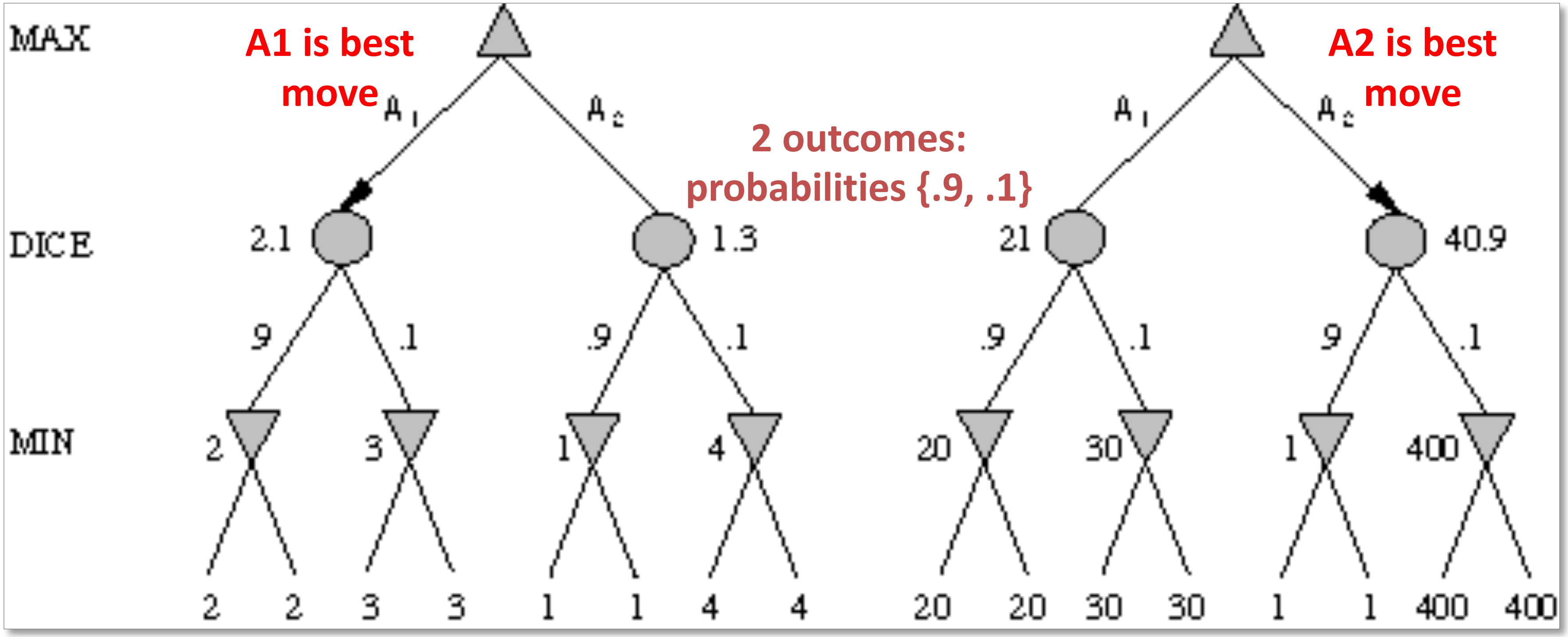
- Chance nodes over max node: expectimax(C) = $\sum_i(P(d_i)*\text{maxval}(i))$
- Chance nodes over min node: expectimin(C) = $\sum_i(P(d_i)*\text{minval}(i))$

Impact on lookahead

- Dice rolls **increase branching factor**
  - There are 21 possible rolls with two dice
- Backgammon: ~20 legal moves for given roll

  ~6K with 1-1 roll (get to roll again!)
- At depth 4: 20 * (21 * 20)**3 ≈ 1.2B boards
- As depth increases, probability of reaching a given node shrinks
  - lookahead's value diminished and alpha-beta pruning is much less effective
- TDGammon used depth-2 search + good static evaluator to achieve world-champion level

Meaning of the evaluation function



- With probabilities & expected values we must be careful about meaning of values returned by static evaluator
- Relative-order preserving change of static evaluation values doesn't change minimax decision, but could here
- Linear transformations are OK

Games of imperfect information

- E.g. card games where opponent's initial hand unknown
  - Can calculate probability for each possible deal
  - Like having one big dice roll at beginning of game
- Possible approach: minimax over each action in each deal; choose action with highest expected value over all deals
- Special case: if action optimal for all deals, it's optimal
- GIB bridge program, approximates this idea by
  1. Generating 100 deals consistent with bidding
  2. Picking action that wins most tricks on average

High-Performance Game Programs

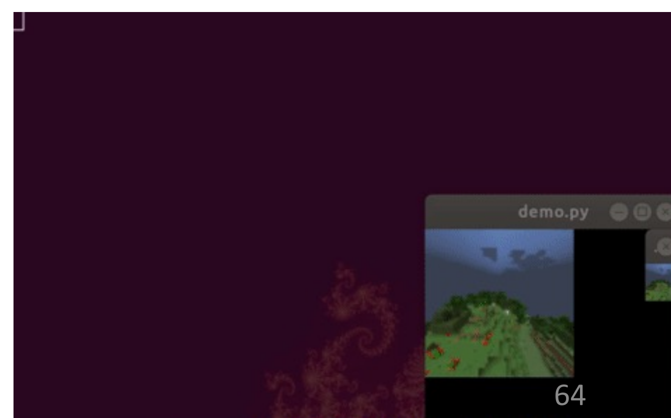
- Many programs based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...
- Chinook searched all checkers configurations with ≤ 8 pieces to create endgame database of 444 billion board configurations
- Methods general, but implementations improved via many specifically tuned-up enhancements (e.g., the evaluation functions)

Other Issues

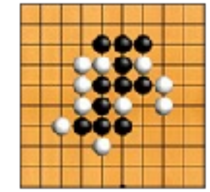
- Multi-player games, no alliances
  - E.g., many card games, like Hearts
- Multi-player games with alliances
  - E.g., Risk
  - More on this when we discuss game theory
  - Good model for a social animal like humans, where we must balance cooperation and competition

AI and video Games

- Many games include agents run by the game program as
  - Adversaries, in first person shooter games
  - Collaborators, in a virtual reality game
  - E.g.: AI bots in Fortnite Chapter 2
- Some games used as AI/ML challenges or learning environments
  - [MineRL](): train bots to play Minecraft
  - [MarioAI](): train bots for Super Mario Bros



AlphaGO



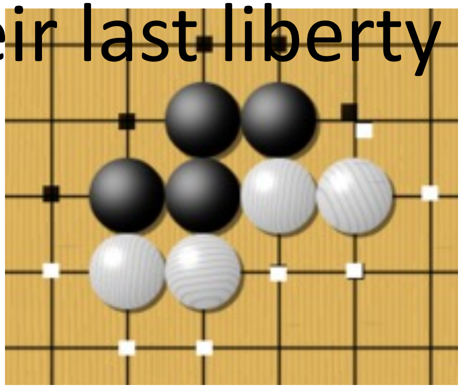
- Developed by Google's DeepMind
- Beat top-ranked human grandmasters in 2016
- Used Monte Carlo tree search over game tree

  expands search tree via random sampling of search space
- *Science* Breakthrough of the year runner-up

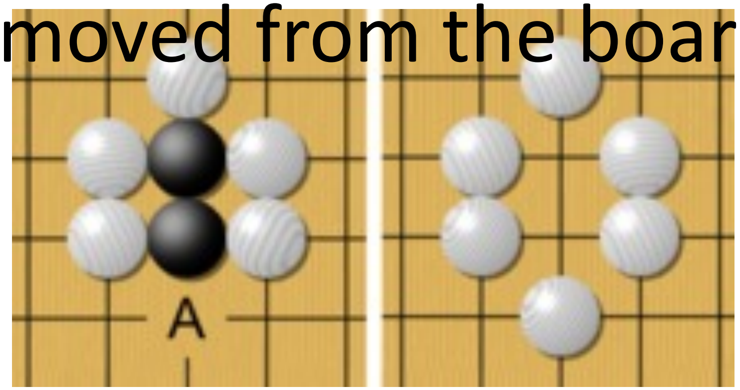
  Mastering the game of Go with deep neural networks and tree search, Silver et al., *Nature*, 529:484–489, 2016
- Match with grandmaster Lee Sedol in 2016 was subject of award-winning 2017 AlphaGo

Go - the game

- Played on 19x19 board; black vs. white stones
- Huge state space $O(b^d)$: chess:~35^{80}, go: ~250^{150}
- Rule: Stones on board must have an adjacent open point ("liberty") or be part of connected group with a liberty. Groups of stones losing their last liberty are removed from the board.



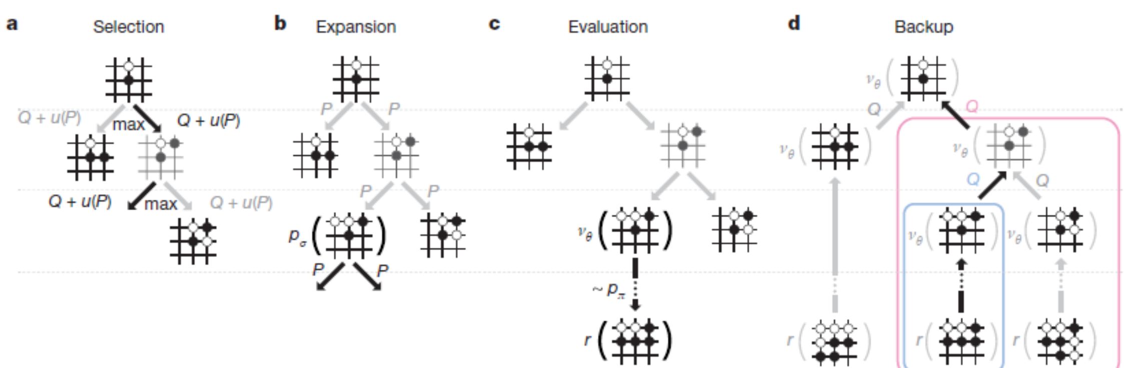
liberties



capture

AlphaGo implementation

- Trained deep neural networks (13 layers) to learn **value function** and **policy function**
- Performs Monte Carlo game search
  - –explore state space like minimax
  - –random "rollouts"
  - –simulate probable plays by opponent according to policy function



AlphaGo implementation

- Hardware: 1920 CPUs, 28O GPUs
- Neural networks trained in two phases over 4-6 weeks
- **Phase 1:** supervised learning from database of 30 million moves in games between two good human players
- **Phase 2:** play against versions of self using reinforcement learning to improve performance