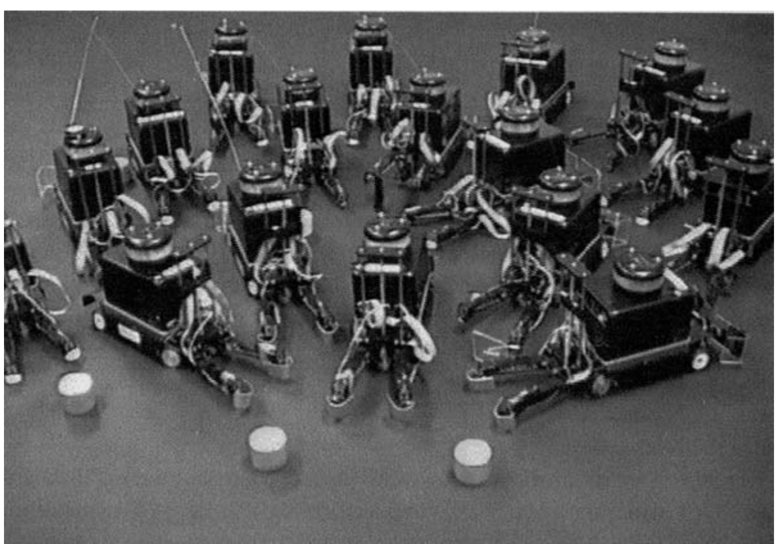
A Primer on Distributed Intelligence

March 26, 2007
Prof. Lynne Parker



Two Types of "Intelligence" in Multi-Agent Systems

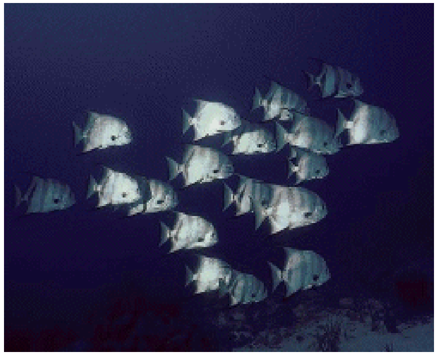
Distributed Intelligence

Cooperative:

Agents work together toward shared goal

Competitive:

Agents have different goals and compete against each other



Part I: Cooperative Intelligence

- Today's focus: *cooperative motions*
  - Specifically:
    - Following/Swarming/Flocking/Schooling
    - Formations

Following / Swarming / Flocking / Schooling

- Natural flocks consist of two balanced, opposing behaviors:
  - Desire to stay close to flock
  - Desire to avoid collisions with flock

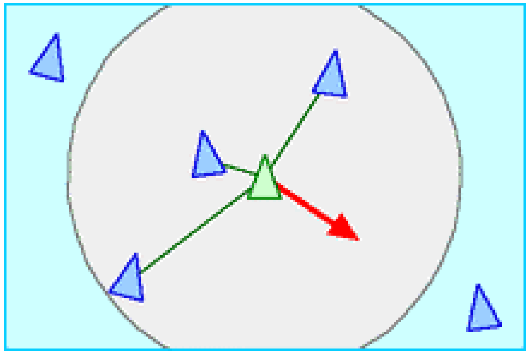
- Why desire to stay close to flock?
  - In natural systems:
    - Protection from predators
    - Statistically improving survival of gene pool from predator attacks
    - Profit from a larger effective search pattern for food
    - Advantages for social and mating activities

Craig Reynolds (1987) Developed Boids

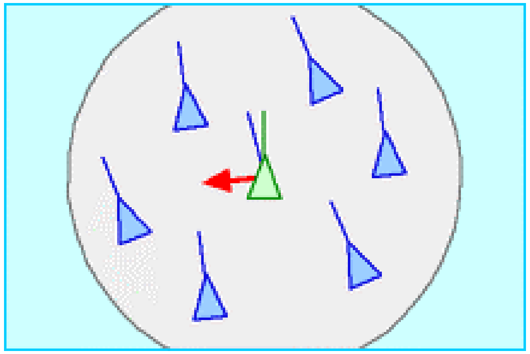
- "Flocks, Herds, and Schools: A Distributed Behavioral Model", Craig Reynolds, *Computer Graphics*, 21(4), July 1987, pgs. 25-34.

Simulated boid flock avoiding cylindrical obstacles

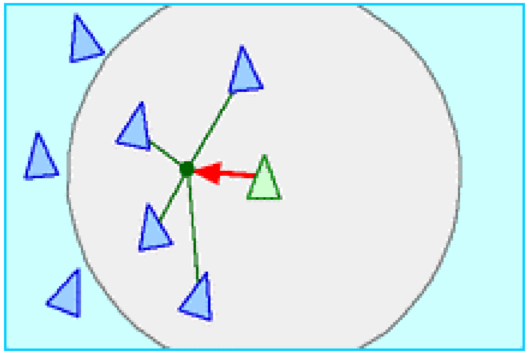
How do Boids work?



Separation: steer to avoid crowding local flockmates



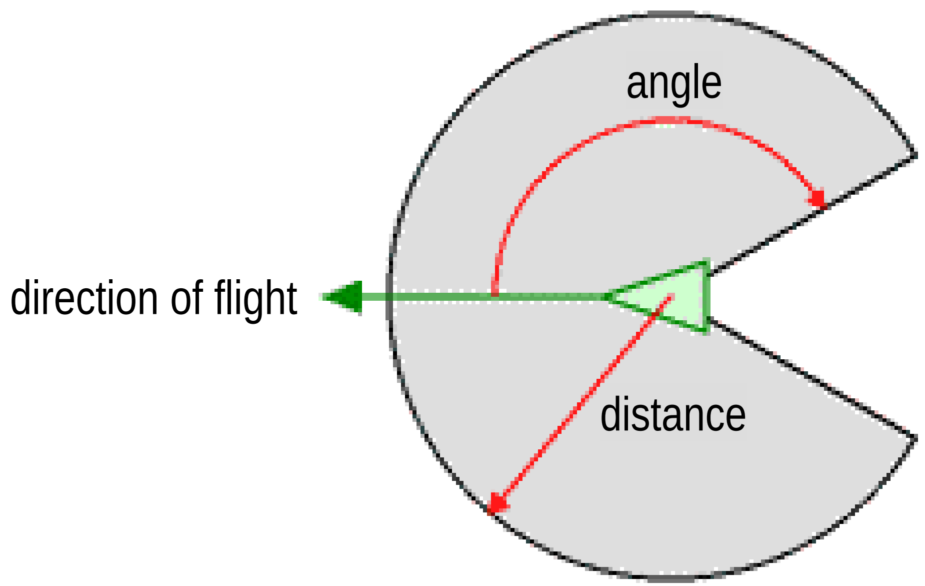
Alignment: steer towards average heading of local flockmates



Cohesion: steer to move toward the average position of local flockmates

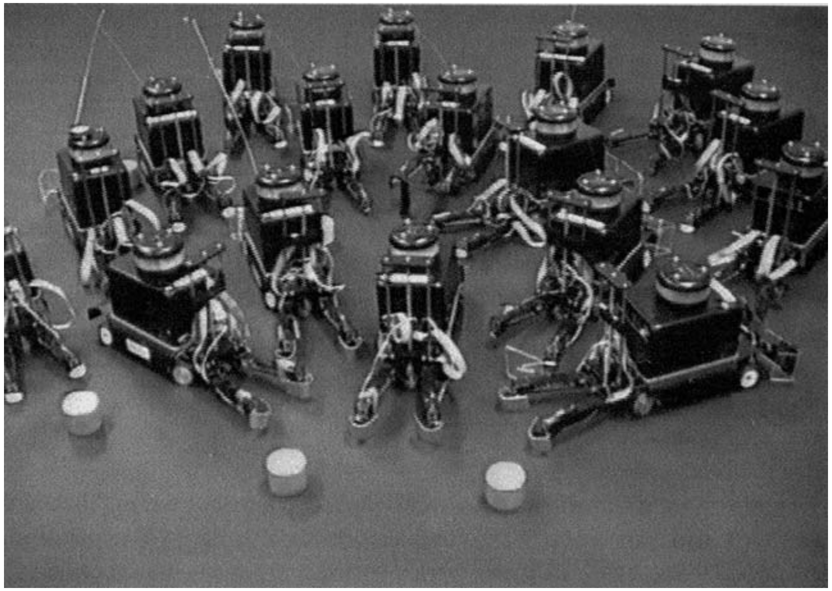
Reynold's Boid Flocks

- Boid neighborhood characterized by:
  - Distance (measured from center of boid)
  - Angle (measured from direction of flight)



Translating these Behaviors to Code on Robots

- Work of Mataric, 1994
- General Idea:
  - Use "local" control laws to generate desired "global" behavior
- The Robots:
  - 12" long
  - 4 wheels
  - Bump sensors around body
  - Radio system for:
    - Localization
    - Communication
    - Data collection
    - "Kin" recognition

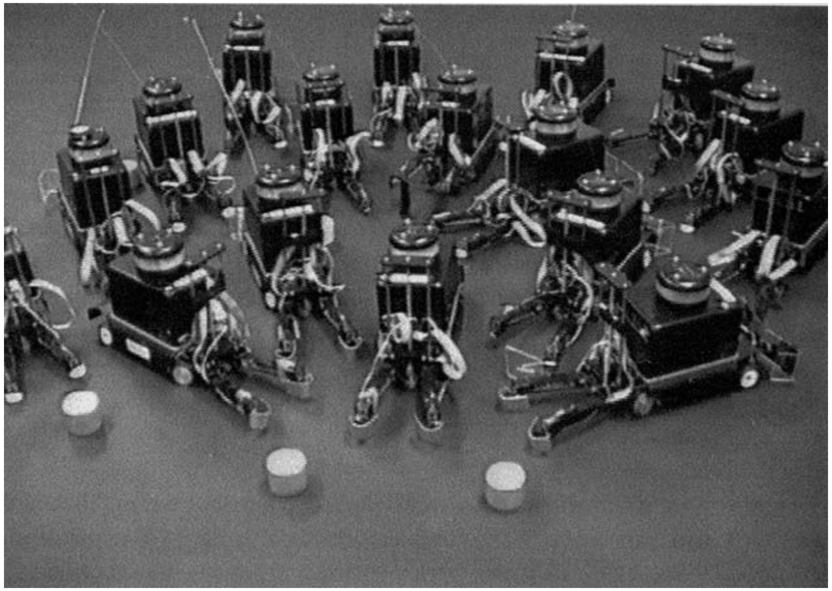


The Nerd Herd: Mataric, MIT, 1994

The Nerd Herd Approach

- Fundamental principle: Define *basis behaviors* as general building blocks for synthesizing group behavior

- Set of basis behaviors proposed:
  - Avoidance
  - Save-wandering
  - Following
  - Aggregation
  - Dispersion
  - Homing



- Combine basis behaviors into higher-level group behaviors:
  - Flocking
  - Foraging

Safe-Wandering Algorithm

- **Avoid-Kin:**
  - Whenever an agent is within d_avoid
    - If the nearest agent is on the left
      - Turn right
      - Otherwise, turn left
- **Avoid-Everything-Else**
  - Whenever an obstacle is within d_avoid
    - If obstacle is on right only, turn left
    - If obstacle is on left only, turn right
    - After 3 consecutive identical turns, backup and turn
    - If an obstacle is on both sides, stop and wait.
    - If an obstacle persists on both sides, turn randomly and back up
- **Move-Around:**
  - Otherwise move forward by d_forward, turn randomly

Following Algorithm

Follow:

- **Whenever an agent is within d_follow**
  - **If an agent is on the right only, turn right**
  - **If an agent is on the left only, turn left**

If sufficient robot density, safe_wandering + follow yield more complex behaviors:

- e.g., osmotropotaxic behavior of ants: unidirectional lanes

Dispersion Algorithm

Dispersion:

- Whenever one or more agents are within d_disperse
  - Move away from Centroid_disperse

Aggregation Algorithm

Aggregate:

- Whenever nearest agent is outside d_aggregate
  - Turn toward the local centroid_aggregate, go.
- Otherwise, stop.

Homing Algorithm

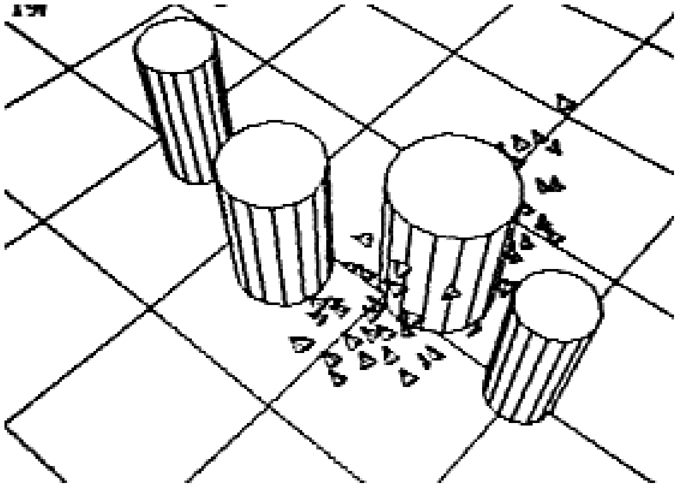
Home:

- Whenever at home
  - Stop
- Otherwise, turn toward home, go.

Generating Flocking Through Behavior Combinations

- **Flock:**
  - **Sum weighted outputs from Safe-Wander, Disperse, Aggregate, and Home**

In general, flocking should allow agents to move around obstacles:



Work of Reynolds (1987)

Boids Movie
"Stanley and Stella in Breaking the Ice"

QuickTime™ and a
Sorenson Video decompressor
are needed to see this picture.

http://odyssey3d.stores.yahoo.net/comanclascli2.html

For lots more information on Boids (Flocks, Herds, Schools …)

- Great web site:

  http://www.red3d.com/cwr/boids/

  Contains lots of pointers to related literature on this topic

Formations

Key Issues:

- What is desired formation?
- How do robots determine their desired position in the formation?
- How do robots determine their actual position in the formation?
- How do robots move to ensure that formation is maintained?
- What should robots do if there are obstacles?
- How do we evaluate robot formation performance?

Example Movies of Column Formation-Keeping

Parker, 1995

Parker et al., 2001

Issue in Formation Keeping: Local vs. Global Control

- Local control laws:
  - No robot has all pertinent information
  - Appealing because of their simplicity and potential to generate globally emergent functionality
  - But, may be difficult to design to achieve desired group behavior

- Global control laws:
  - Centralized controller (or all robots) possess all pertinent information
  - Generally allow more coherent cooperation
  - But, usually increases inter-agent communication

Let's look at approach of Balch (1998)

"Behavior-Based Formation Control for Multiagent Robot Teams", by Tucker Balch, Ronald C. Arkin

Published in:

IEEE Transactions on Robotics and Automation

December, 1998.

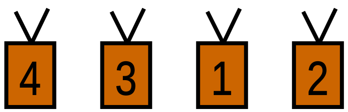
Available online at:

http://www.cs.cmu.edu/~trb/papers/formjour.ps.Z

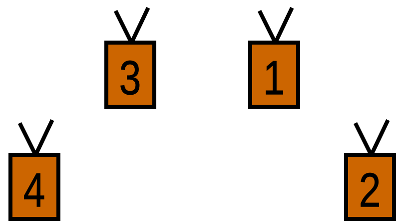
Balch's Formation Types and Position Determination

Formations:

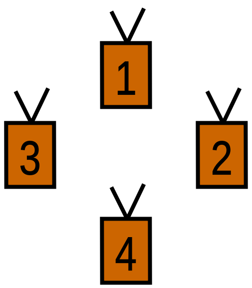
Line



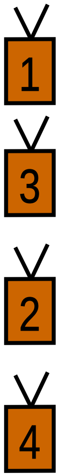
Wedge



Diamond

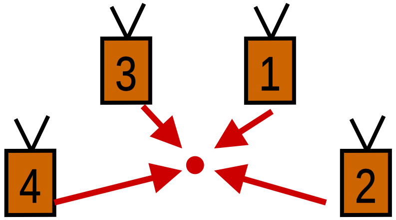


Column

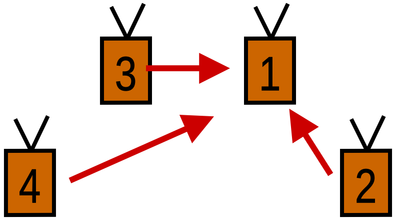


Position Determination (i.e., figuring out where robot should be):

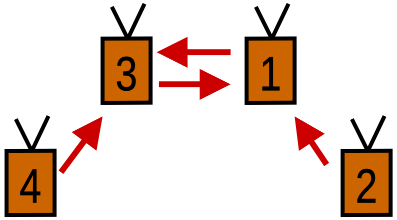
Unit-center-referenced



Leader-referenced



Neighbor-referenced



Requirements of Formation Techniques

- Unit-center approach:
  - Requires transmitter and receiver for all robots
  - Requires protocol for exchanging position information
  - Places heavy demand on passive sensor systems: each robot has to track 3 other robots that may be spread across a very large field of view
- Leader-referenced approach:
  - Requires only one transmitter for leader and one receiver for each follower robot
  - Thus, has reduced communications bandwidth
  - Require tracking only one robot
  - However, leader may be too far away to sense
  - Local interactions among robots may make little sense, if they aren't paying attention to each other
- Neighbor-referenced approach:
  - Requires tracking only one other robot
  - However, less information on global formation requirements ➔ could be more formation error

Basic Behaviors of Formation-Keeping Robot

Combine behaviors:

- move-to-goal
- avoid-static-obstacle
- avoid-robot
- maintain-formation

Result:

Robot moves to a goal location while remaining in formation and avoiding obstacles and collisions with other robots

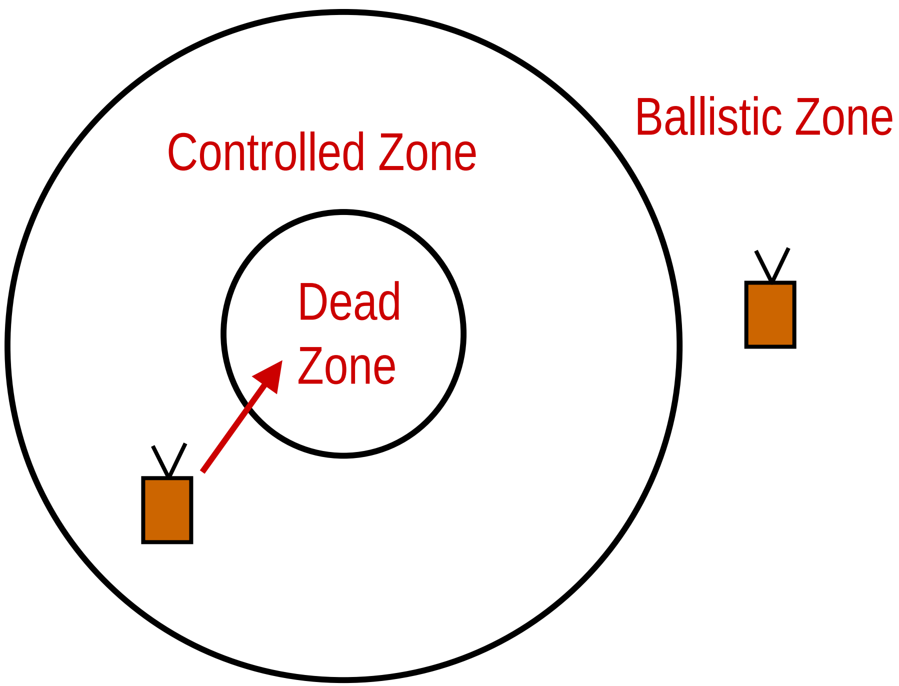
Maintain-formation

- Perceptual Function: detect-formation-position
  - Determine robot's desired location
  - Determine robot's relative position in the overall formation
  - Determine other robots' positions
- Motor output (in form of motion vector).
  - Direction: always in the direction of the desired formation position.
  - Magnitude: depends on how far the robot is away from the desired position.

Output Vector Magnitude Calculation

- Dead zone:
  - Robot is within acceptable positional tolerance.
  - Output vector magnitude is always 0.
- Controlled zone:
  - Robot is somewhat out of position.
  - Output vector magnitude decreases linearly from a maximum at zone's furthest edge to 0 at the inner edge.
  - Directional component: points toward dead zone's center.
- Ballistic zone:
  - Output vector magnitude is set to its maximum
  - Directional component points

Magnitudes:



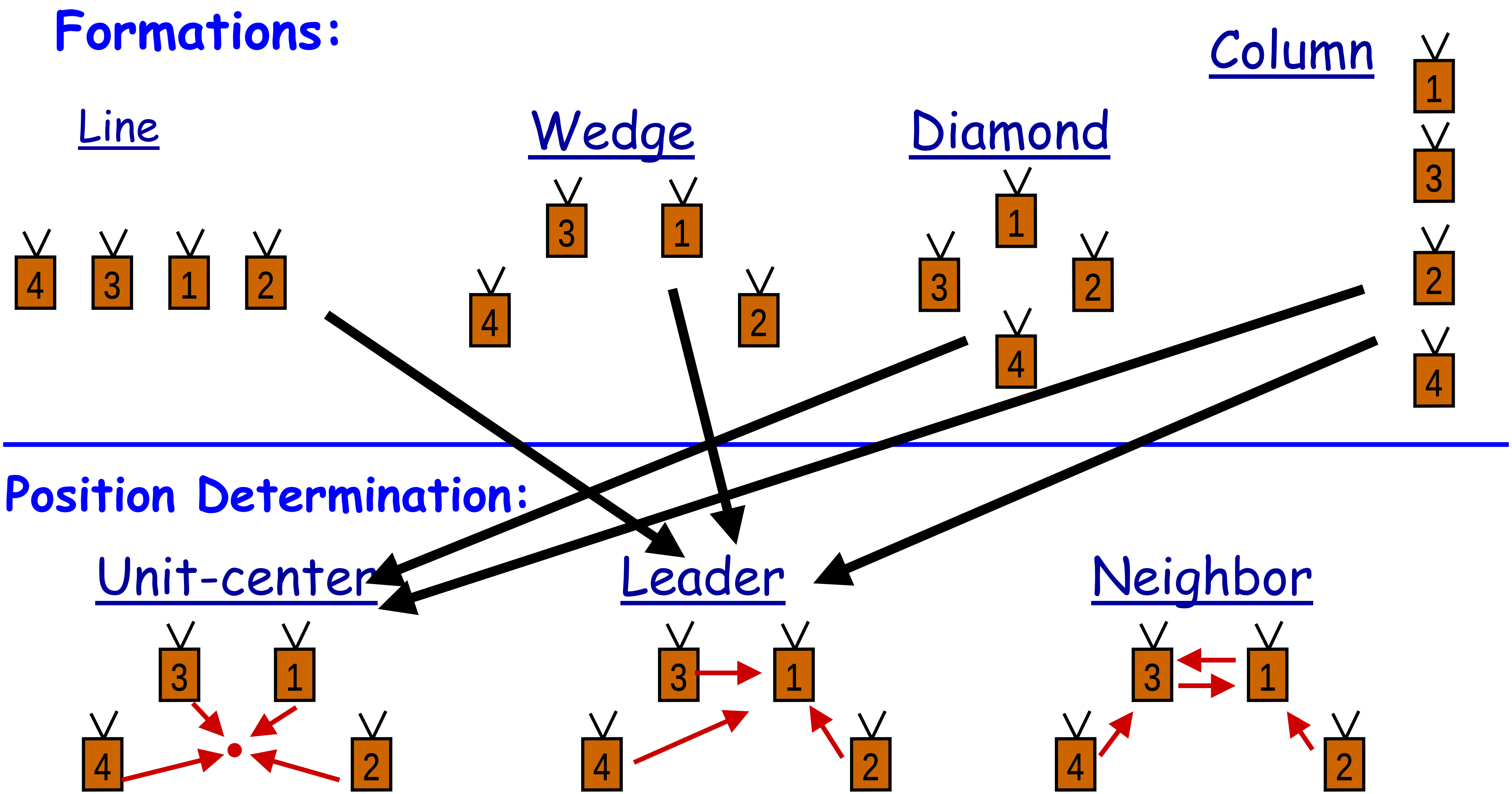
When there are obstacles…

- To avoid obstacles like barriers, choices are:
  - Move as an unit around the barrier
  - Divide into subgroups
  - Depends on the relative strengths of behaviors (gain)

Balch's Formation Results

- For 90 degree turns:
  - Diamond formation best with unit-center-reference
  - Wedge, line formations best with leader-reference
- For obstacle-rich environments:
  - Column formation best with either unit-center or leader-reference
- Most cases:
  - Unit-center better than leader-center
  - Except:
    - If using human leader, not reasonable to expect to use unit-center
    - Unit-center requires transmitter and receiver for all robots, whereas leader-center only requires transmitter at leader plus receivers for all robots
    - Passive sensors are difficult to use for unit-center

Summary of Formation Approaches: Which is best when?



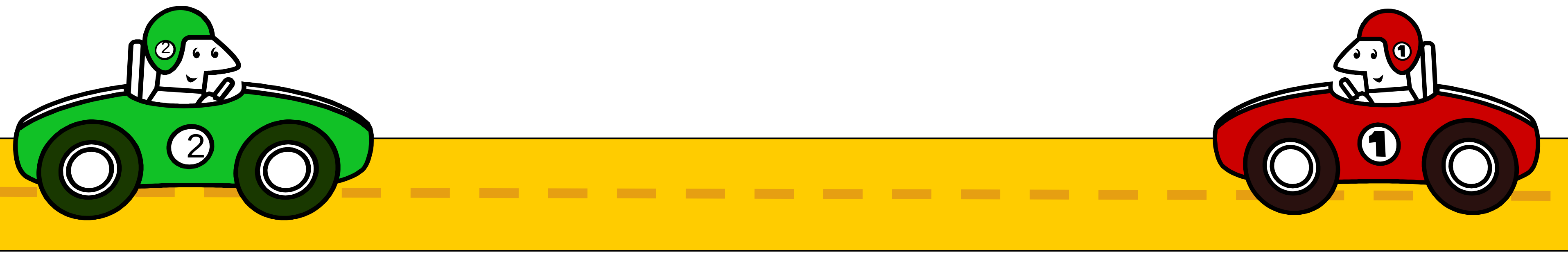
Part II: Competitive Agents

- Can take lots of forms

- Today: Focus on *Game Theory*

What is Game Theory About?

- Analysis of situations where conflict of interests are present



- Game of Chicken
  - driver who steers away looses
- What should drivers do?
- Goal is to prescribe how conflicts can be resolved

What is a Game?

- Various types of games exist (e.g. card, board, sport, war, etc.)

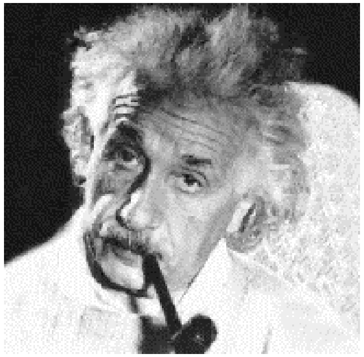
- Game Theory deals with games having the following properties:
  - Two or more *players*
  - Choice of action involves a *strategy*
  - One or more *outcomes*
  - Outcome depends on the chosen strategies: i.e., *strategic interaction*

- Rules out:
  - Games of pure chance
  - Games without strategic interaction

Five Elements of a Game

1. Set of Players
2. Set of Actions
3. Set of Strategies
4. Set of Outcomes
5. Payoff or Utility

Assumed Rationality



- We assume players are *rational*

- That is, players try to maximize their payoffs, irrespective of what the other players are doing.

Example: The Prisoners' Dilemma (PD) Game

- **Players:**

  2 Prisoners

- **Actions:**

  Prisoner 1: Confess, Deny

  Prisoner 2: Confess, Deny

- **Strategies:**

  Choose action simultaneously, without knowing each other's actions.

- **Outcomes:**

  Quantified in prison years

- **Payoff:**

  Fewer years == Better payoff

Types of Games

- Sequential vs. Simultaneous moves
- Single Play vs. Iterated
- Zero vs. non-zero sum
-
- Perfect vs. Imperfect information
- Cooperative vs. conflict
- Deterministic vs. chance

Representation of Games
Matrix Form

- A *matrix* which shows the players, strategies, and payoffs.
- Presumed that players act simultaneously.
- Prisoner's Dilemma example:

	P2 Confess	*P2 Deny*
P1 Confess	5, 5	0, 10
P1 Deny	10, 0	1, 1

General Matrix Representation of a Game

Strategy set for Player 2

Strategy set for Player 1

		Player 2		
		A	B	C
Player 1	A	(2, 2)	(0, 0)	(-2, -1)
	B	(-5, 1)	(3, 4)	(3, -1)

Payoff to Player 1

Payoff to Player 2

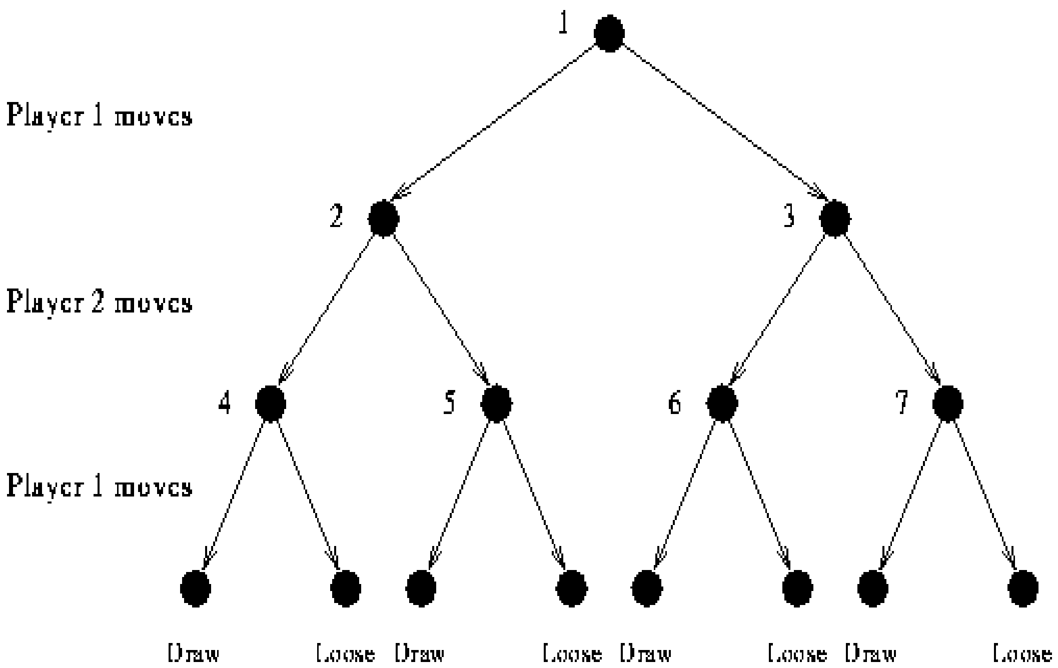
- Simultaneous play
  - players analyze the game and write their strategy on a paper
- Combination of strategies determines payoff

How to Solve?

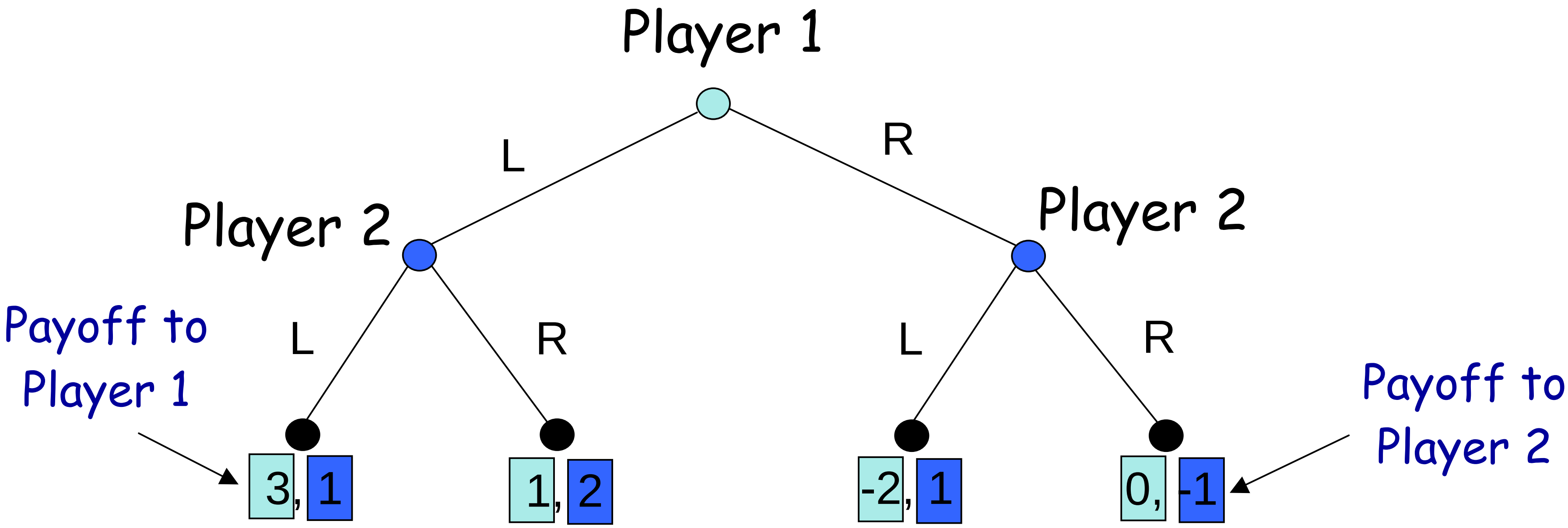
- Use concepts of:
  - Dominated strategy removal
  - Saddle points
  - Pareto optimality
  - …
- Too much to get into today
- Instead:
  - Convert matrix to game tree
  - Assume iterated decisions (instead of simultaneous)
    - That is, players take turns making decision
  - Make use of mini-max algorithm

Game Trees

- Non-leaf nodes:
  - Represent decision-point for one of the players
- Edges:
  - Represent available choices of actions
- Leaf nodes:
  - State payoffs for each player



Game Tree Example



- Strategy set for Player 1: {L, R}

- Strategy for Player 2: __, __

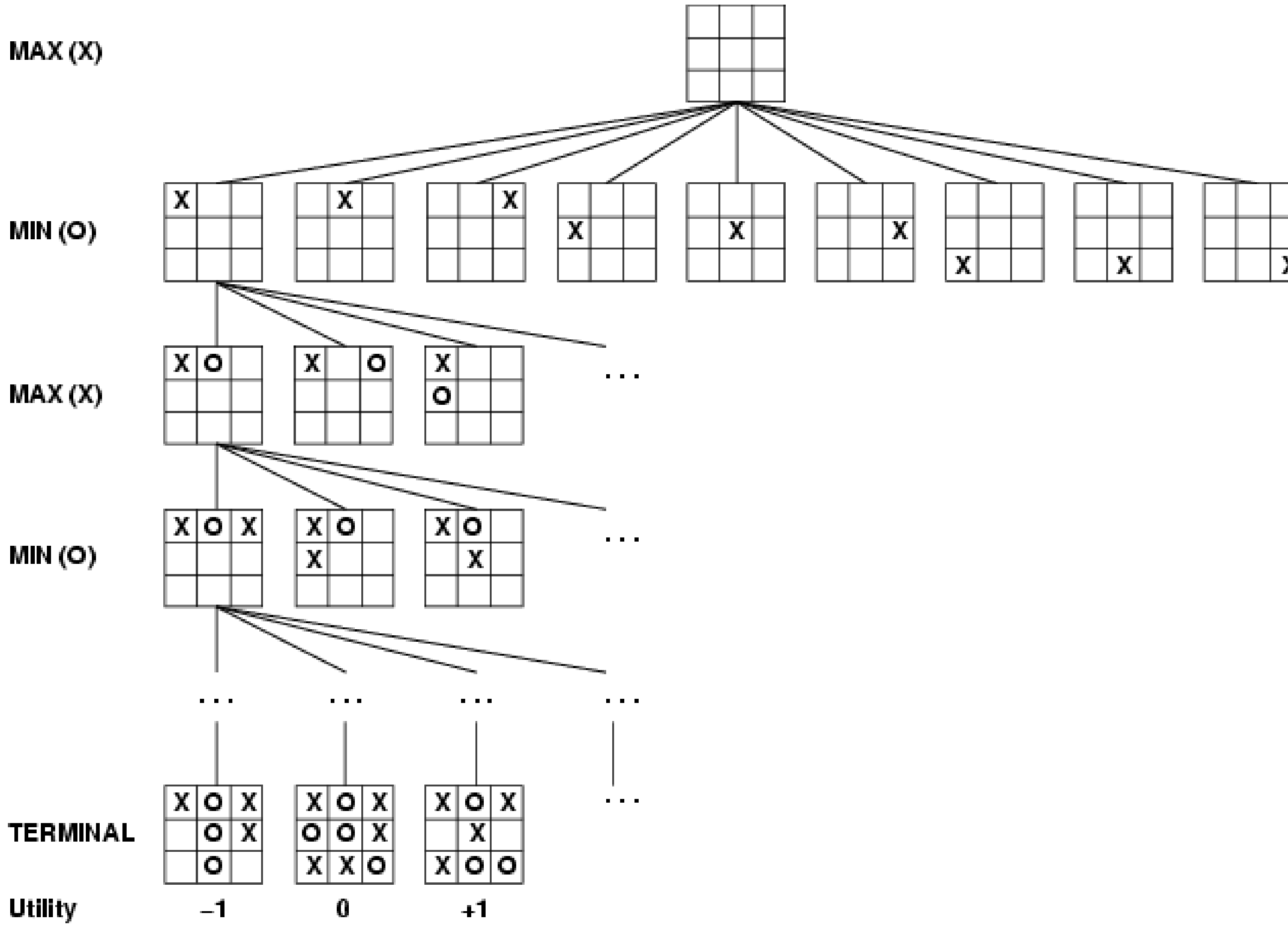
what to do when P1 plays L

what to do when P1 plays R

- Strategy set for Player 2: {LL, LR, RL, RR}

Game Tree Applied to Noughts and Crosses (i.e., Tic-tac-toe)

- 2-player
- Deterministic
- Turn taking



Minimax Algorithm

- Minimax algorithm
  - Perfect for deterministic, 2-player game
  - One opponent tries to maximize score (Max)
  - One opponent tries to minimize score (Min)
  - Goal: move to position of highest minimax value
  - Identify best achievable payoff against best play

The Mini-Max Algorithm Approach

Algorithm approach:

1. Generate game tree completely

2. Determine utility of each terminal state

3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level

4. At the root node use <u>minimax decision</u> to select the move with the max (of the min) utility value

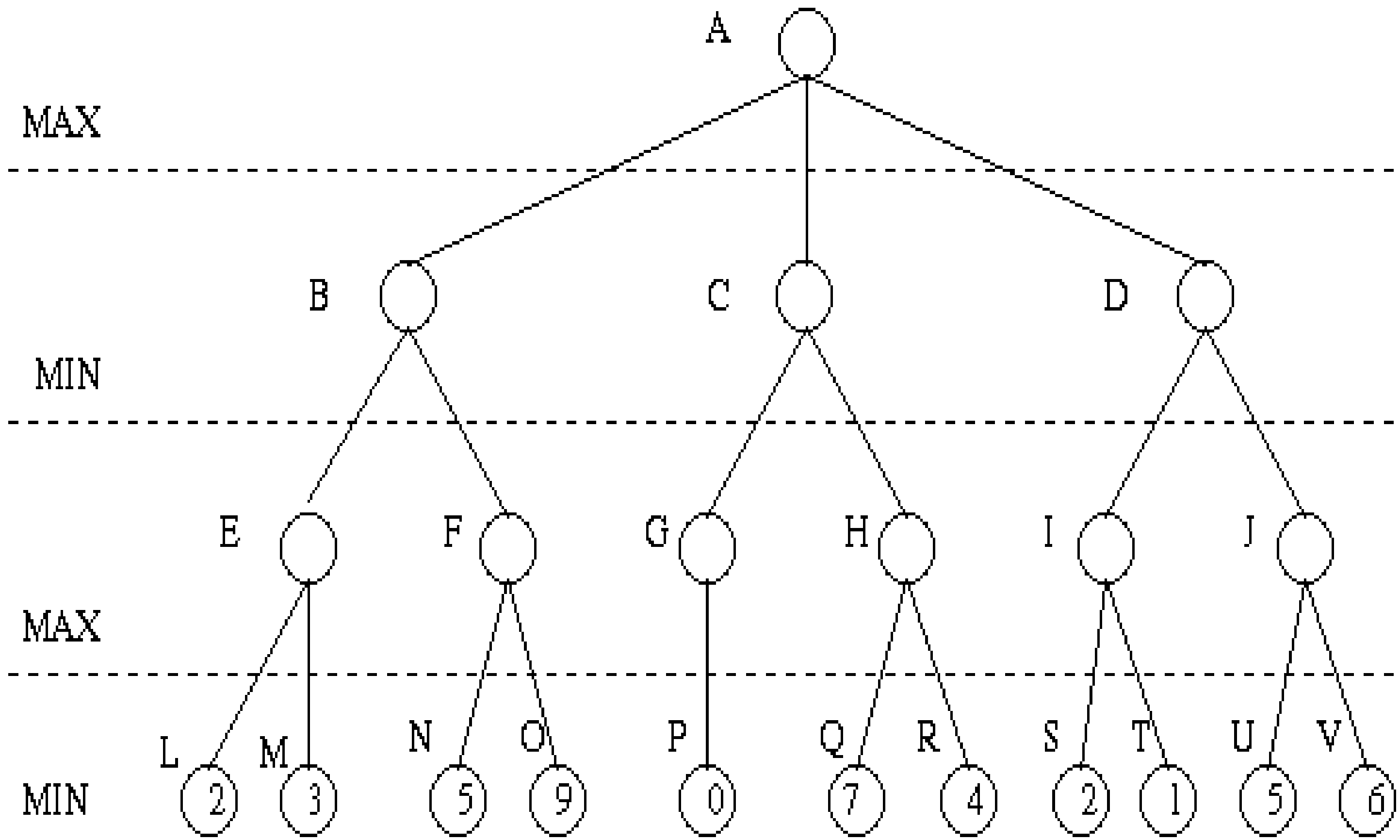
Minimax Algorithm Code: Recursive implementation

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v
```

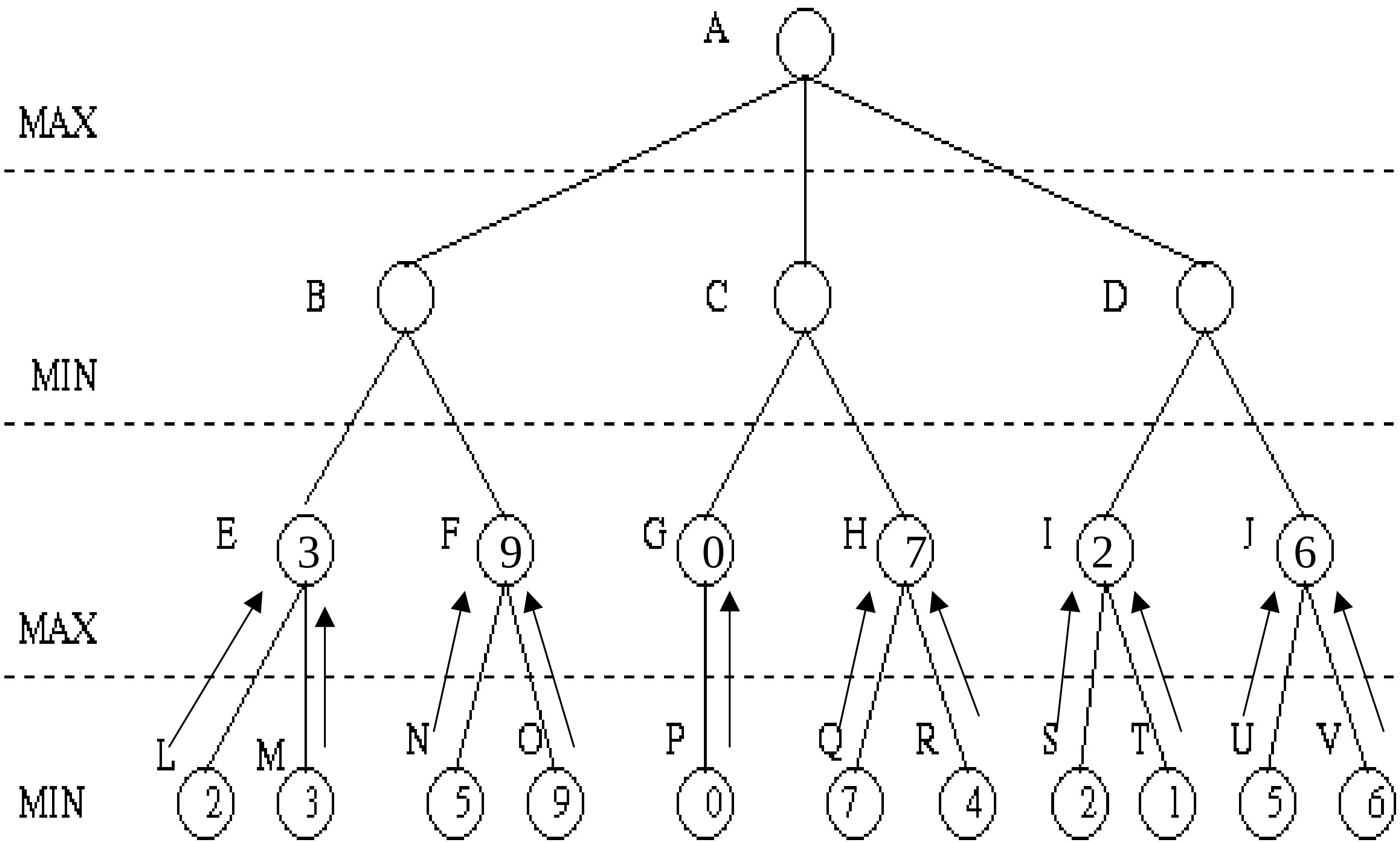
```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

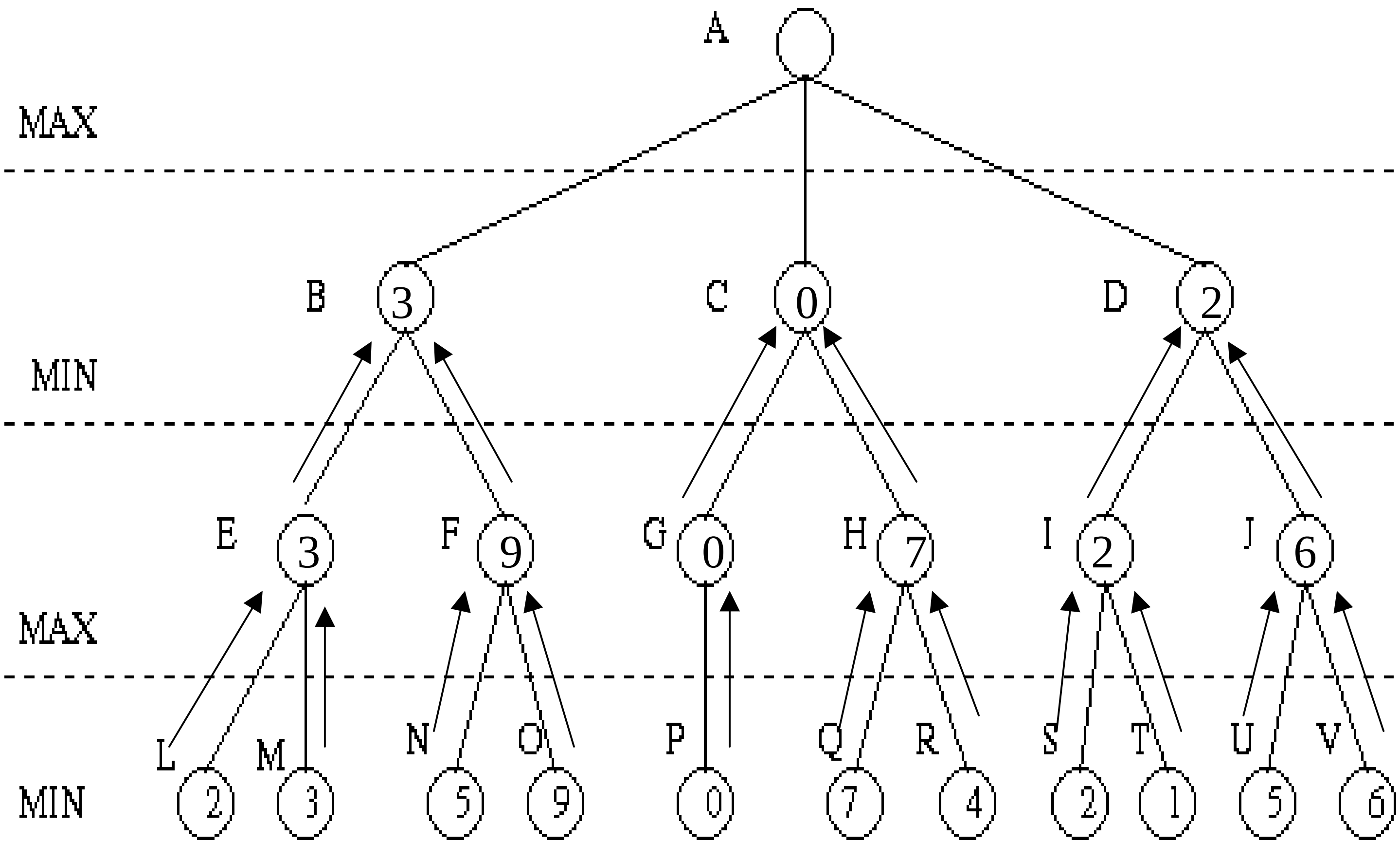
Minimax Algorithm (cont'd)



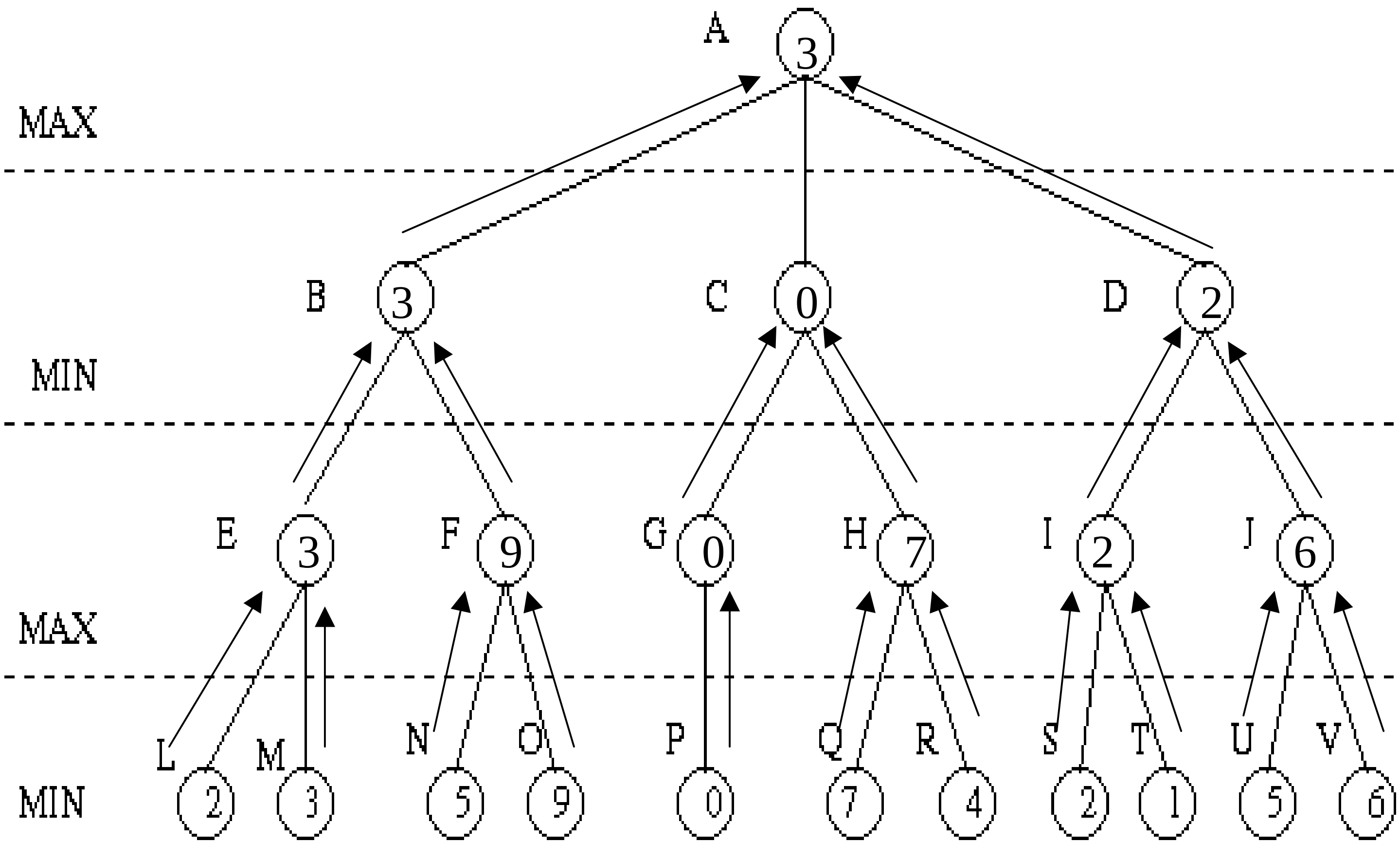
Minimax Algorithm (cont'd)



Minimax Algorithm (cont'd)



Minimax Algorithm (cont'd)



Minimax Algorithm (cont'd)

- Properties of minimax algorithm:
  - Complete? Yes (if tree is finite)
  - Optimal? Yes (against an optimal opponent)
  - Time complexity? $O(b^m)$
  - Space complexity? $O(bm)$ (depth-first exploration)

But we can do better…
Move evaluation without complete search

- Complete search is too complex and impractical

- **New** α-β Algorithm:

  - CUTOFF-TEST: cutoff test to replace the termination condition (e.g., deadline, depth-limit, etc.)

  - EVAL: evaluation function to replace utility function (e.g., number of chess pieces taken)

More on the α-β algorithm

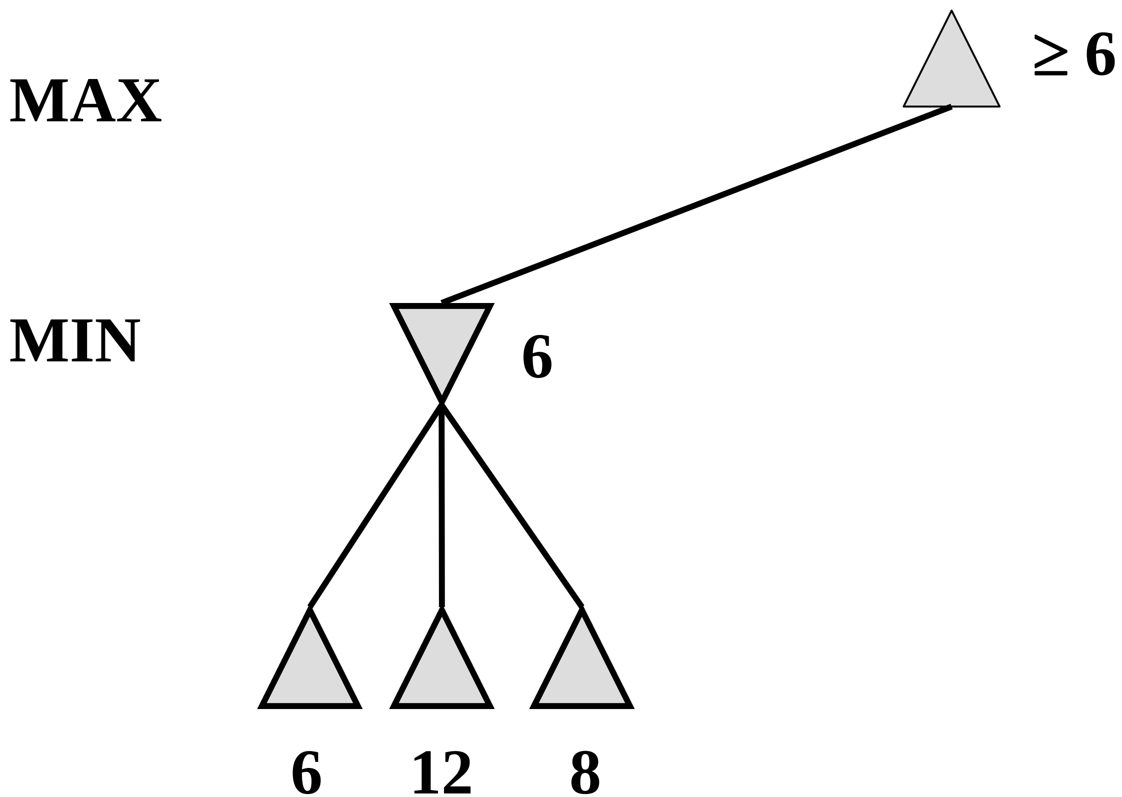
- Principle:
  - If a move is determined worse than another move already examined, then further examination deemed pointless

- Same basic idea as minimax, but prune (cut away) branches of the tree that we know will not contain the solution.

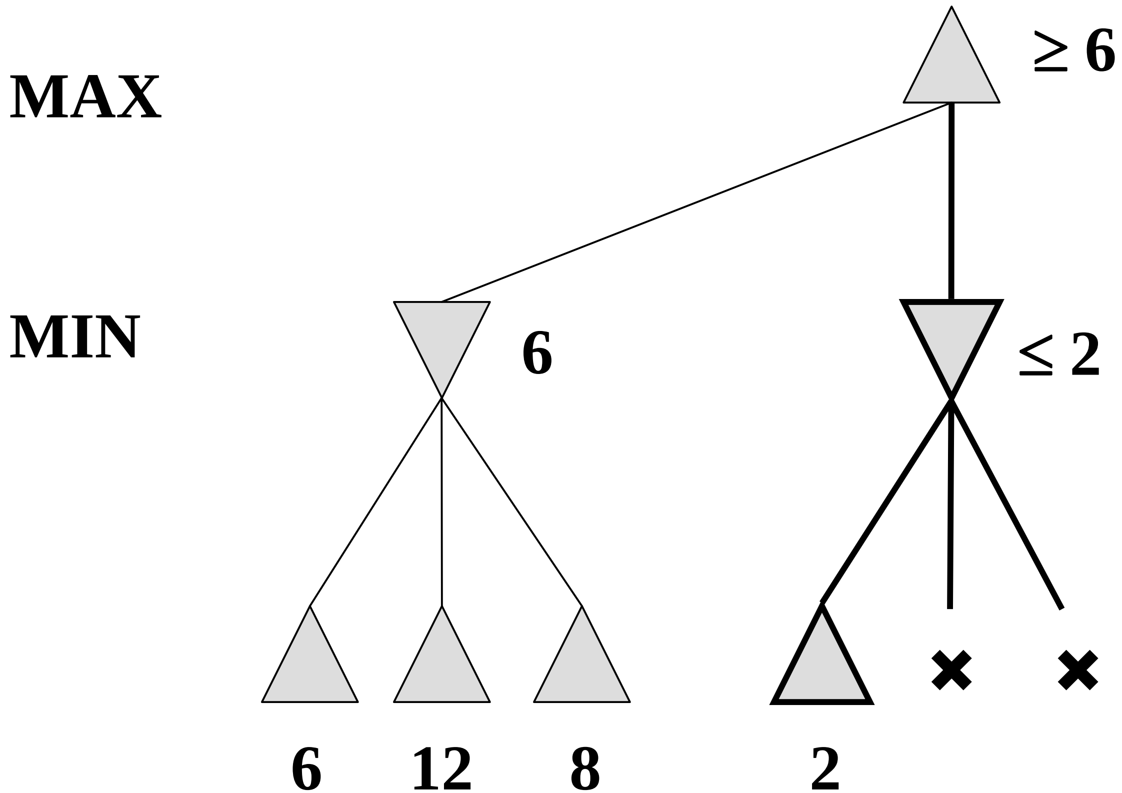
- Because minimax is depth-first, let's consider nodes along a given path in the tree. Then, as we go along this path, we keep track of:
  - α : Best choice so far for MAX
  - β : Best choice so far for MIN

- Does it work? Yes, in roughly cuts the branching factor from b to $\sqrt{b}$ resulting in twice as far look-ahead than pure minimax

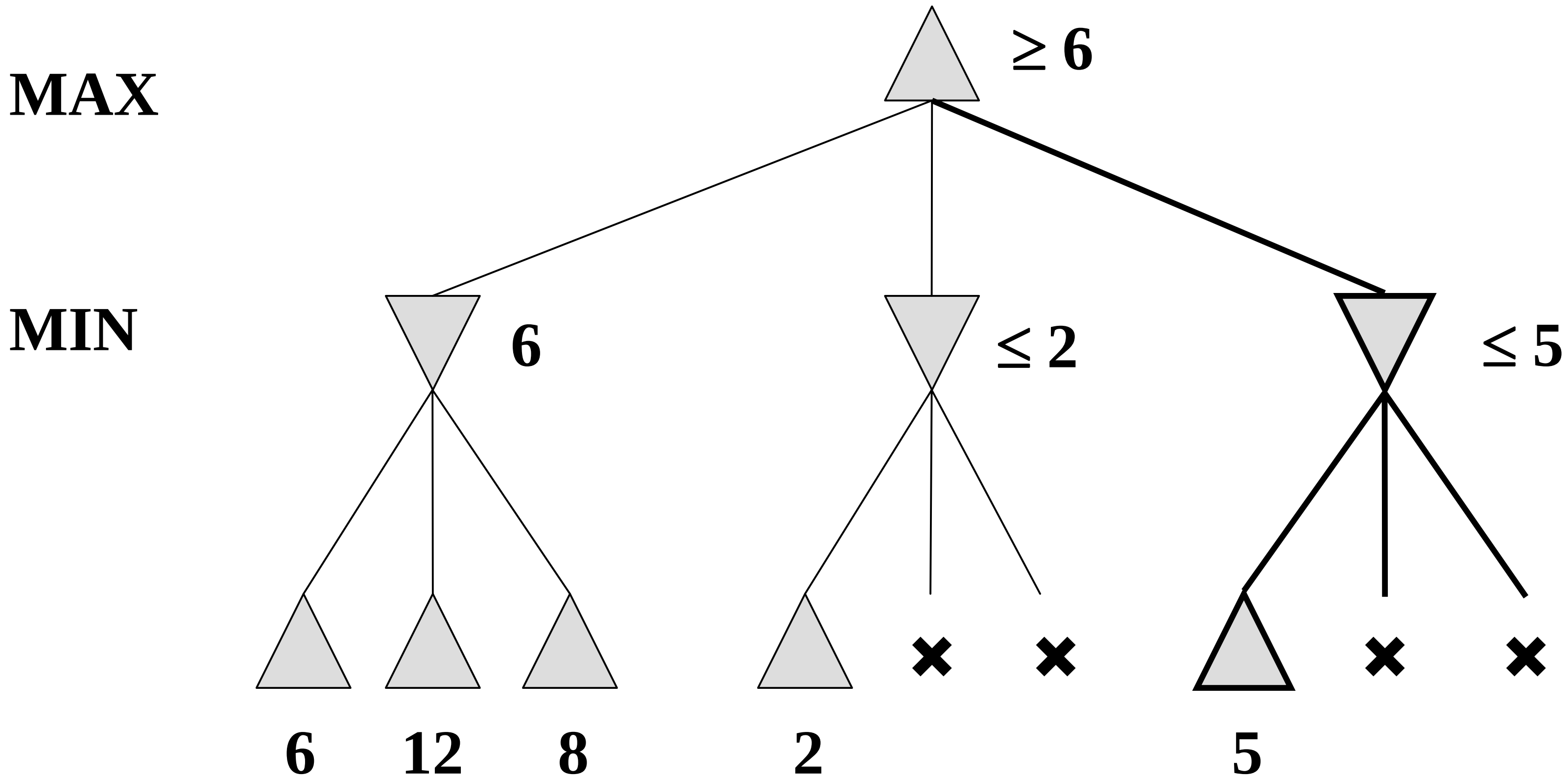
α–β pruning: example



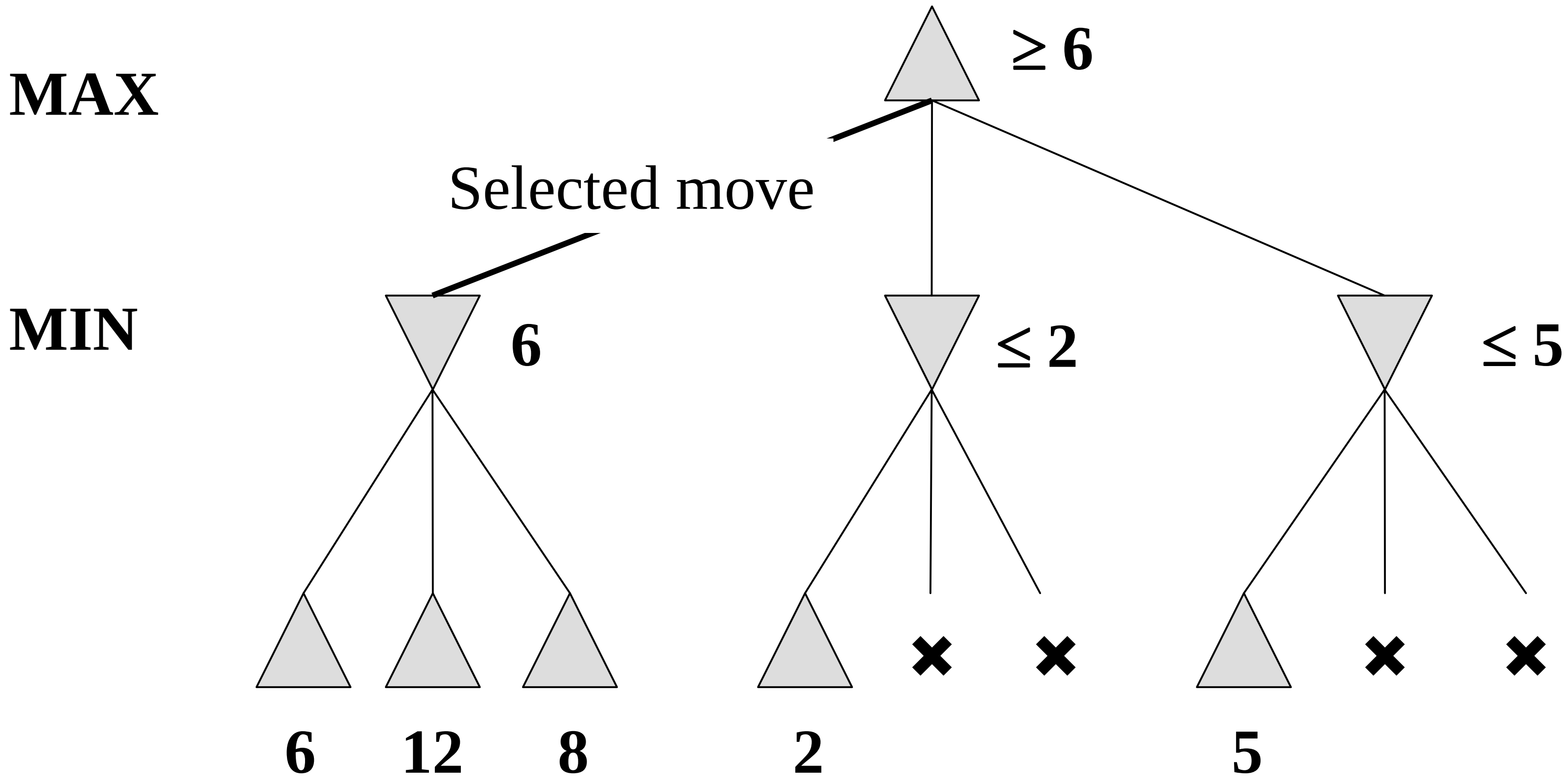
α–β pruning: example



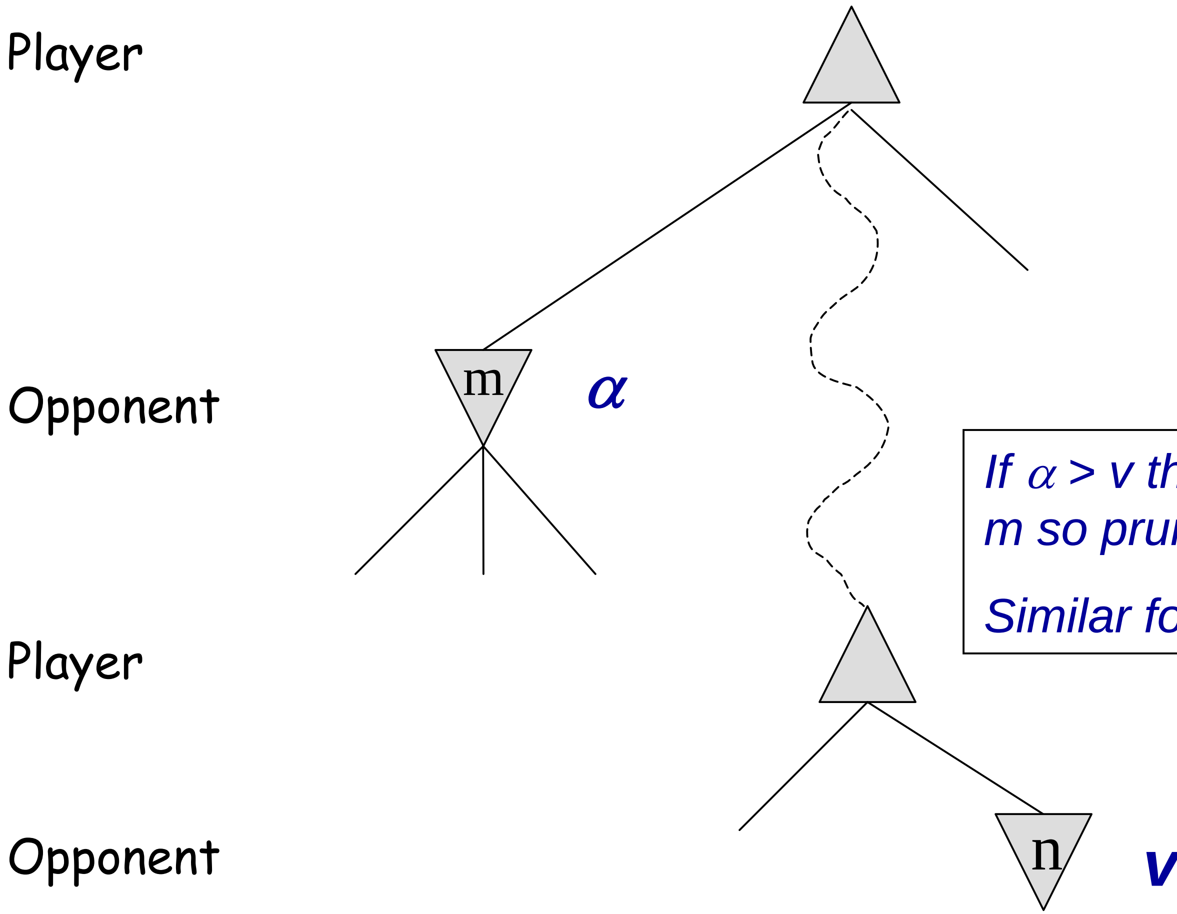
α–β pruning: example



α-β pruning: example



α-β pruning: General principle



If α > v then MAX will chose m so prune tree under n

Similar for β for MIN

The α-β algorithm

Basically MINIMAX + keep track of α, β + prune

```
function MAX-VALUE(state, game, α, β) returns the minimax value of state
   inputs: state, current state in game
           game, game description
           α, the best score for MAX along the path to state
           β, the best score for MIN along the path to state
   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
        α ← MAX(α, MIN-VALUE(s, game, α, β))
        if α ≥ β then return β
   end
   return α
```

```
function MIN-VALUE(state, game, α, β) returns the minimax value of state

   if CUTOFF-TEST(state) then return EVAL(state)
   for each s in SUCCESSORS(state) do
        β ← MIN(β, MAX-VALUE(s, game, α, β))
        if β ≤ α then return α
   end
   return β
```

Note: These are both Local variables. At the Start of the algorithm, We initialize them to ***α = -∞ and β = +∞***

Applet for experimenting with Minimax and Alpha-Beta

http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Summary: We've looked at two types of "Intelligence" in Multi-Agent Systems

Distributed Intelligence

Cooperative:

Agents work together toward shared goal

Techniques we've looked at:

- Following / Swarming / Flocking / Schooling
- Formations

Competitive:

Agents have different goals and compete against each other

Techniques we've looked at:

- Game Theory
- Minimax algorithm
- α-β algorithm