Adversarial Games Lectures — Contents
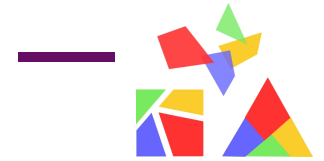
COMP2240

Artificial Intelligence

Lecture AG-1

Adversarial Games: Introduction

Outline

- Types and properties of games.

- Strategies.

- The basic idea of the MiniMax technique for move evaluation.

- Consideration of some particular games.

Games vs. search problems

Games have "Unpredictable" opponent.

$\Longrightarrow$ The solution is a contingency plan.

Moves have time limits.

$\Longrightarrow$ Unlikely to find optimal goal, must approximate.

Ideas to beat the problems of AI game playing:

- algorithm for perfect play (Von Neumann, 1944)
- finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
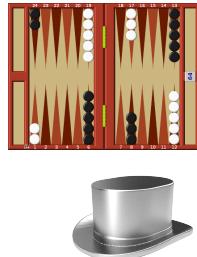- pruning to reduce costs (McCarthy, 1956)

History

- Computer considers possible lines of play (Babbage, 1846).

- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944).

- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950).

- First chess program (Turing, 1951).

- Machine learning to improve evaluation accuracy (Samuel, 1952–57).

- Pruning to allow deeper search (McCarthy, 1956).

Some Fundamental Types of Game

	deterministic	chance
perfect information	Chess Checkers, Go, Othello	Backgammon Monopoloy
imperfect information	Stratego	Bridge Poker Scrabble War

Other Important Game Properties

Number of players. (Assume we are dealing with 2 player games unless otherwise stated.)

Time limitations. (Either per move or for the whole game)

Modelling our Adversary.
Can we just consider the game state at each move, or do we need to consider the other player's strategy (and hence look at previous moves in the game).

AI *vs* Game Theory Approaches

AI:

From the perspective of AI we tend to look at game playing as an elaboration the problem of searching a plan that achieves a goal.

Game strategies are contingent plans aimed a achieving a goal (winning) within the context of a rective and opposing environment.

This is sometimes called *combinatorial game theory*.

(Simultaneous) Game Theory:

Game theory typically reduces games to a situation where players simultaneously choose actions from a set of choices; and each player gains some reward or pays some penalty depending on the combination of actions that were chosen by them and by the other players.

Game Strategy (in AI Approach)

Informally, a game *strategy* is simply a way of playing a game.

Mathematically, a game strategy can be modelled by a function which determines the next move of a player for any state of the game that might occur when it is that player's move.

(A strategy is associated with only one player of the game.
It does not determine the moves of other players.)

For computerised game play, we would typically define a strategy by means of some kind of rule set and/or algorithm.
(Though if there are a sufficiently small number of move states, it could just be a lookup table.)

Good Strategies

A strategy for a game does not necessarily have to be a good way of playing it.

Whether a strategy is good (i.e. likely to lead to a win) may depend on what strategy is being used by the opponent(s).

Some strategies may be good against certain opposing strategies but bad against other opposing strategies.

Winning Strategy

A *winning strategy* for a game is one that will always win the game whatever moves the opponent(s) play.

For some games, there is a winning strategy that works right from the beginning.

There could be a winning strategy for the first player or for the second player.
(There cannot be a winning strategy for both players. Why?)

Rather than having a winning strategy from the beginning a player may find a winning strategy from a game state that occurs part way through the game.
From then on, by following this strategy, victory is guaranteed.
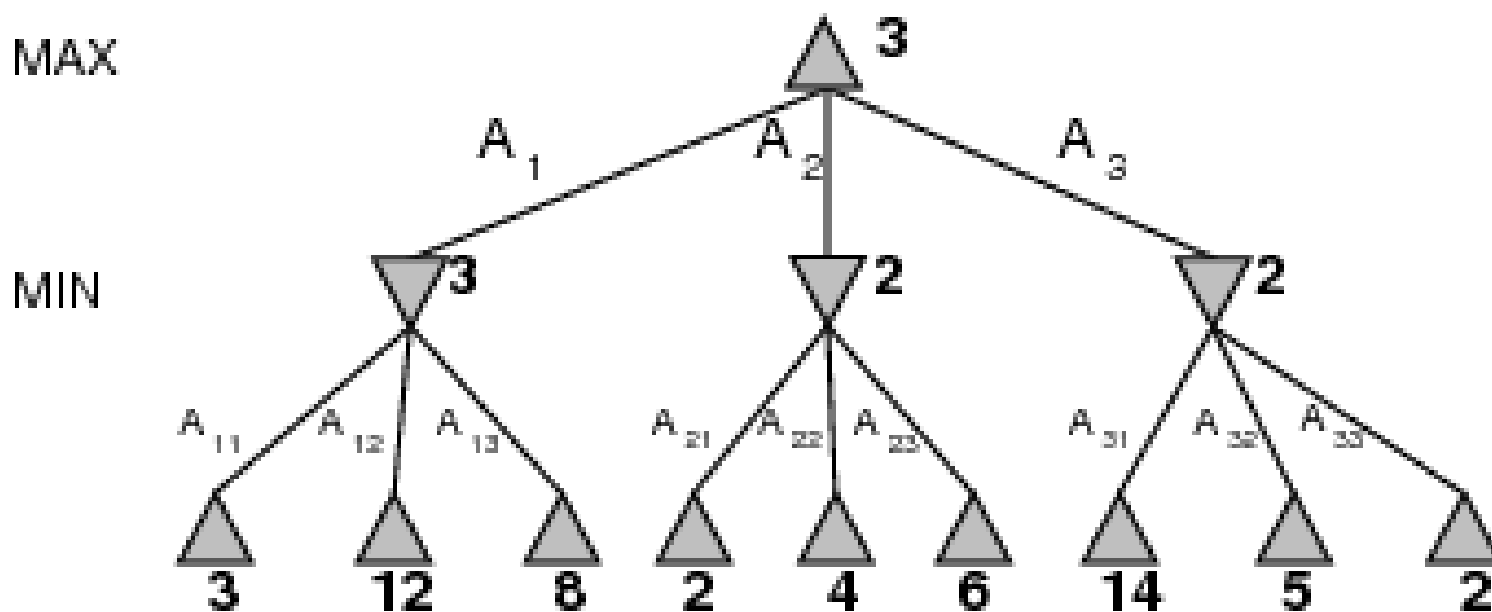
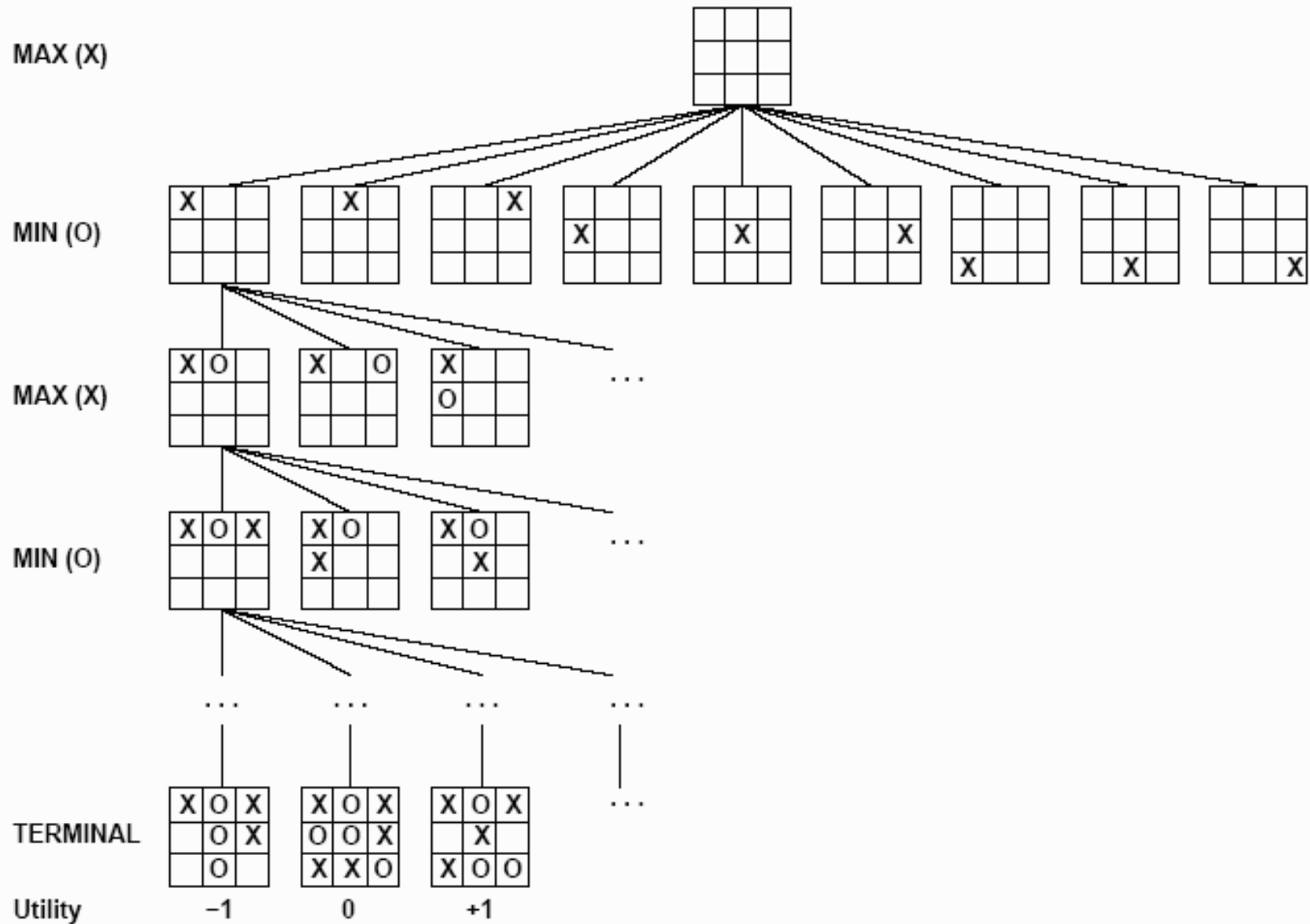# Tic Tac Toe Winning Strategies
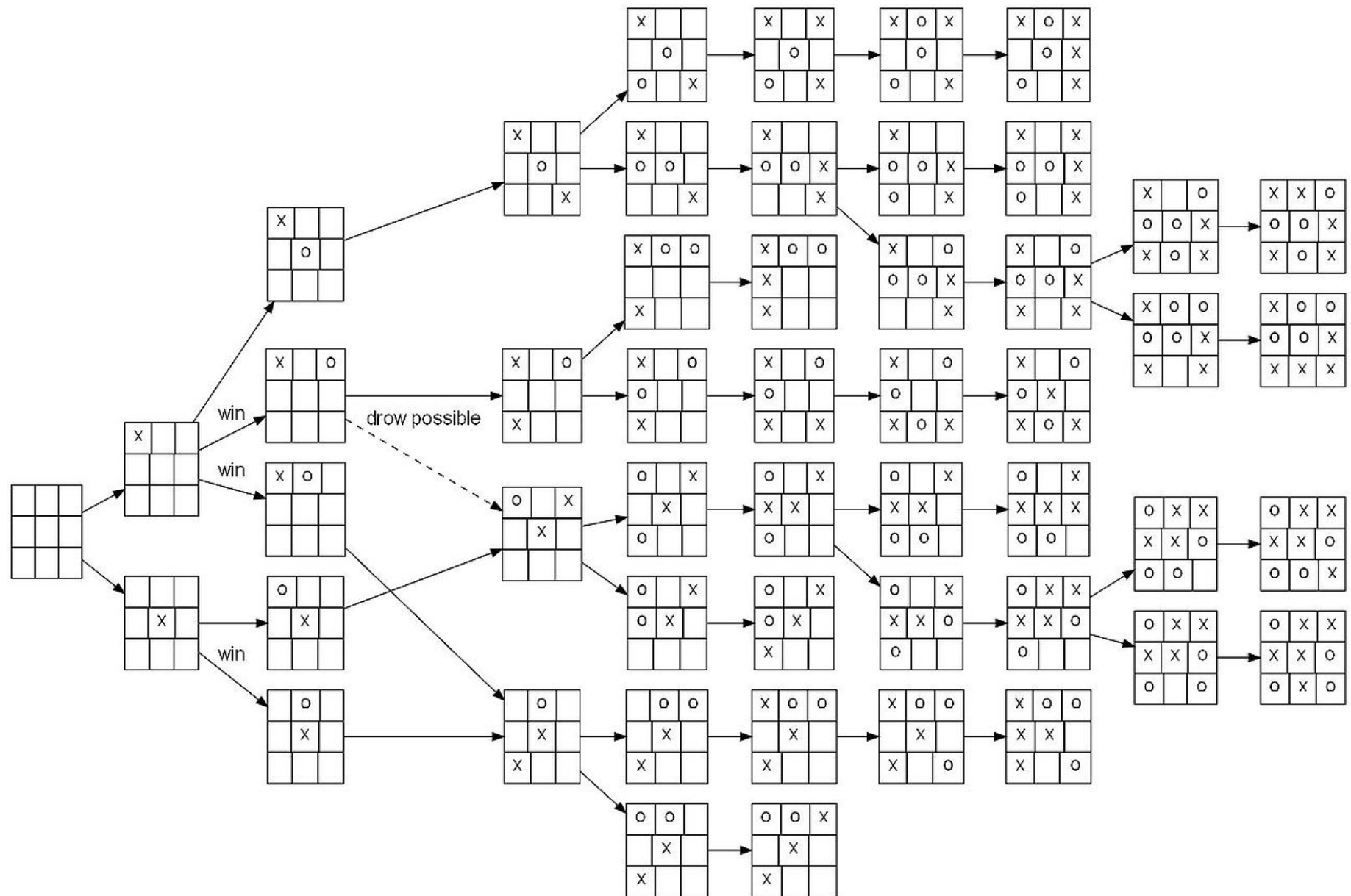
Minimax

Moving to position with highest *minimax value* gives best achievable payoff assuming that the opponent always makes their best play.

E.g., 2-ply game:

Gives perfect play for deterministic, perfect-information games.

Tic Tac Toe Minimax

Checkers

The early work on computer game playing by Arthur L. Samuel introduced and developed several techniques that were key to progress in this area and have had major influences on the field of AI in general.

Techniques used in Samuel's Checkers Program

The Samuel Checkers-playing Program appears to be the world's first self-learning program, and as such a very early demonstration of this fundamental concept of AI.

Board state evaluation used a heuristic based on a weighted sum of numerical feature scores.

Best weightings learned by playing many different versions against each other.

Move preferences calculated from the heuristics by means of n-ply look-ahead using minimax with α-β pruning.

Book Learning (storing calculated values of board states) used to improve efficiency.

Chess

Chess Position Evaluation

As with most complex games, it is not possible for a computer to consider all possible move sequences right up to the end of the game. Thus it needs to evaluate board states by some heuristic.

Values of Pieces		Position of pieces (white Knight)							
		-50	-40	-30	-30	-30	-30	-40	-50
Pawn	100	-40	-20	0	0	0	0	-20	-40
Knight	320	-30	0	10	15	15	10	0	-30
Bishop	330	-30	5	15	20	20	15	5	-30
Rook	500	-30	0	15	20	20	15	0	-30
Queen	900	-30	5	10	15	15	10	5	-30
King	20000	-40	-20	0	5	5	0	-20	-40
		-50	-40	-30	-30	-30	-30	-40	-50

The white Knight position table encodes the heuristic that Knights are usually strongest near the centre of the board.

Endgame Problems

Look ahead approaches (such as MiniMax) tend to do badly at the *endgame* play of chess.
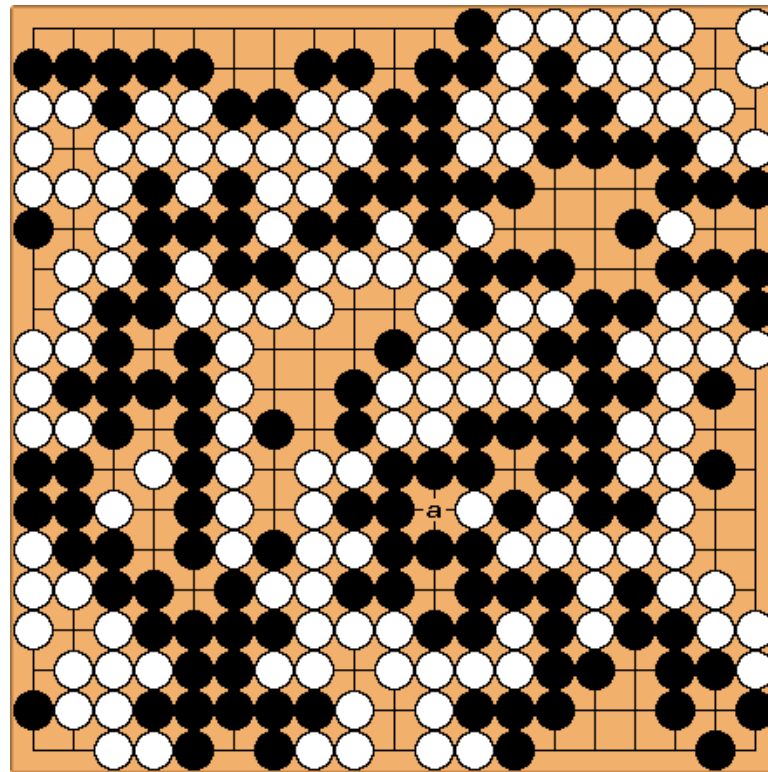
Endgame strategies can involve quite long sequences of moves where a player gradually forces their opponent into a losing position.

Go

The game of *Go*, which originated in China more than 2,500 years ago, has proved extremely challenging problem for computational game playing.

Go End Position

The winner is the player who surrounds the most unoccupied vertices at the end of the game.

Many Moves and Increasing Complexity

The large board size (1919, 361) allows many different moves and prevents deep lookahead.

For the first move in chess, the player has twenty choices. Go players begin with a choice of 55 distinct legal moves, accounting for symmetry. This number rises quickly as symmetry is broken and soon almost all of the 361 points of the board must be evaluated. Some are much more popular than others, some are almost never played, but all are possible.

Also, pieces do not disappear, so the game state gets more and more complicated.

Why Might Humans be Better

Once placed, go pieces are not moved.

It has been suggested that humans find it easier to think about development in time that is 'additive'.

This means that the situation develops by adding more structure, but the original structure is still present.

This kind of change may be easier for humans to think about.

Why?

Game Theoretic Approach: eg 1

Rock-paper-scissors

Column player aka.
player 2
(simultaneously)
chooses a column

Row player
aka. player 1
chooses a row

0, 0	-1, 1	1, -1
1, -1	0, 0	-1, 1
-1, 1	1, -1	0, 0

A row or column is called an **action** or (pure) strategy

Row player's utility is always listed first, column player's second

Zero-sum game: the utilities in each entry sum to 0 (or a constant)
Three-player game would be a 3D table with 3 utilities per entry, etc.

Game Theoretic Approach: eg 2

"Chicken"

- Two players drive cars towards each other
- If one player goes straight, that player wins
- If both go straight, they both die
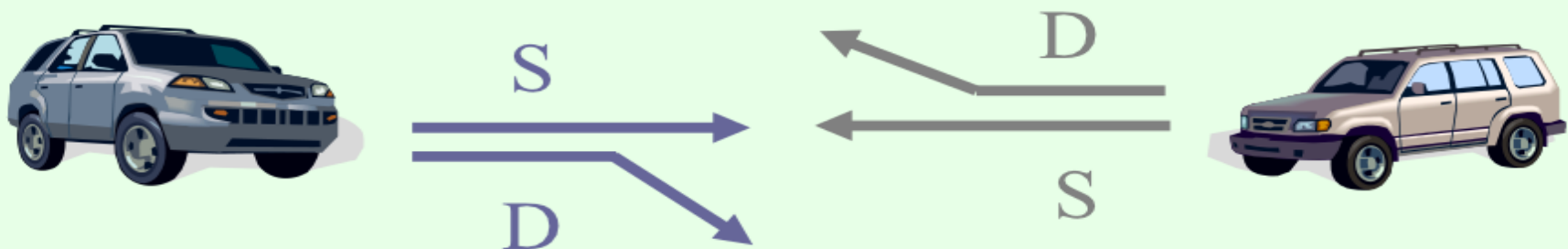
	D	S
D	0, 0	-1, 1
S	1, -1	-5, -5

not zero-sum

COMP2240

Artificial

Intelligence

Lecture AG-2

Move Evaluation using MiniMax

Basic Idea of MiniMax

We want to find the best move from a given game position, on the assumption that the opponent will also play their best move.

Need to look at subsequent moves.
(*Ideally* we would look ahead right to the end of the game, but this may not be possible).
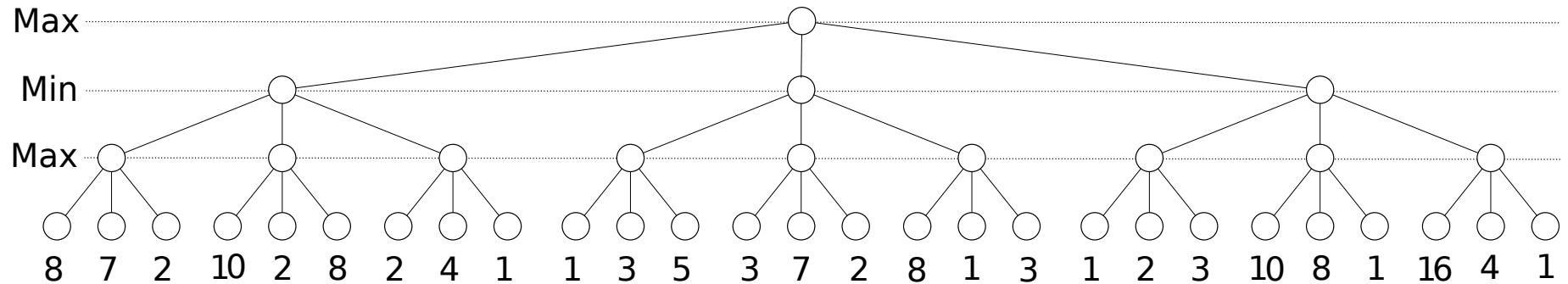
We can define a recursive procedure for calculating the value of a move based on the evaluation of subsequent moves.

Since moves alternate between our player and the opponent, the calculation alternates between taking the maximum and the minimum of the values calculated for the following state.
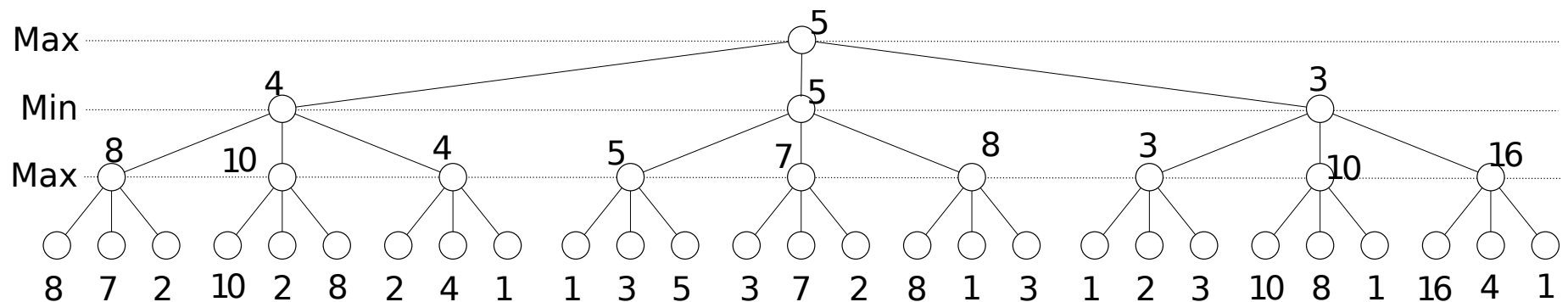
MiniMax Example

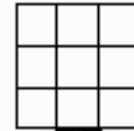## Game state evaluation scores for 2-ply look-ahead:

Max ·
Min ·
Max ·

8 7 2 10 2 8 2 4 1 1 3 5 3 7 2 8 1 3 1 2 3 10 8 1 16 4 1

Calculated minimax evaluation:

Max · · · · · · · · · · · · 5 ·
Min · · · · 4 · · · · · · · · · · · 5 · · · · · · · · · · · · · · · · · 3 · · · · · · · · · · · · · · ·
Max · · 8 · · 10 · · 4 · · · 5 · · 7 · · 8 · · 3 · · 10 · · 16 · · · · ·

8 7 2 10 2 8 2 4 1 1 3 5 3 7 2 8 1 3 1 2 3 10 8 1 16 4 1

Tic Tac Toe Minimax
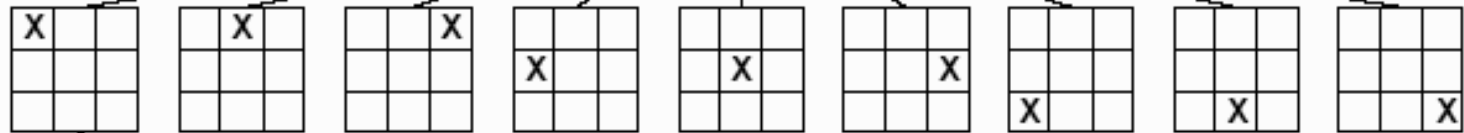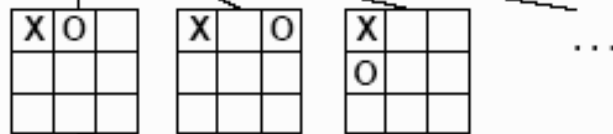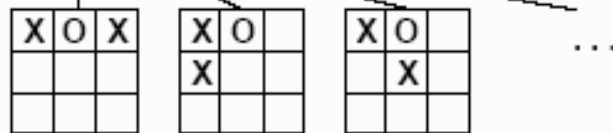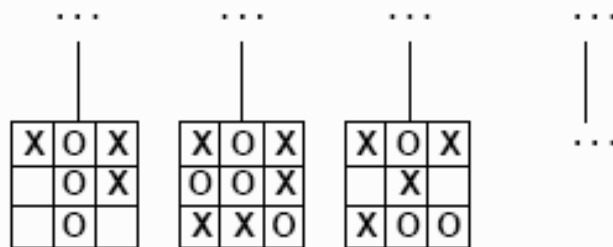
Minimax Algorithm

function Minimax-Decision(*game*) **returns** *an operator*

    for each *op* **in** Operators[*game*] **do**
        Value[*op*] ← Minimax-Value(Apply(*op, game*), *game*)
    end
    return the *op* with the highest Value[*op*]

---

function Minimax-Value(*state, game*) **returns** *a utility value*

    if Terminal-Test[*game*](*state*) **then**
        return Utility[*game*](*state*)
    else if MAX is to move in *state* **then**
     return the highest Minimax-Value of Successors(*state*)
    else
      return the lowest Minimax-Value of Successors(*state*)

Properties of minimax

Complete ?

Properties of minimax

Complete ?

Yes, if tree is finite (chess has specific rules to ensure this)

Optimal ?

Properties of minimax

Complete ?

Yes, if tree is finite (chess has specific rules to ensure this)

Optimal ?

Yes, against an optimal opponent. Otherwise??

Time complexity ?

Properties of minimax

Complete ?

Yes, if tree is finite (chess has specific rules to ensure this)

Optimal ?

Yes, against an optimal opponent. Otherwise??

Time complexity ? $O(b^m)$

Space complexity ?

Properties of minimax

Complete ?

Yes, if tree is finite (chess has specific rules to ensure this)

Optimal ?

Yes, against an optimal opponent. Otherwise??

Time complexity ? $O(b^m)$

Space complexity ? $O(m)$ (using depth-first exploration)

Properties of minimax

Complete ?

Yes, if tree is finite (chess has specific rules to ensure this)

Optimal ?

Yes, against an optimal opponent. Otherwise??

Time complexity ? $O(b^m)$

Space complexity ? $O(m)$ (using depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
$$\Rightarrow \text{exact solution completely infeasible.}$$
(b = branching factor, m = depth.)

Limited Search

Suppose we have 100 seconds and explore 10^4 nodes/second
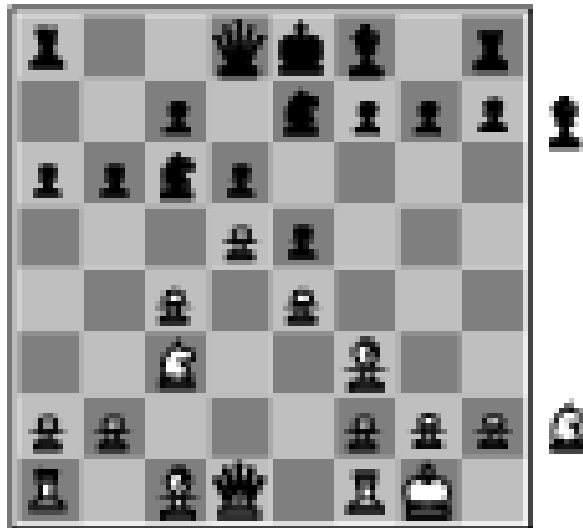$$\implies 10^6 \text{ nodes per move.}$$

But $35^4 = 1500625 \;\; (> 10^6)$.
So can only do around 4 ply look-ahead, which is quite naive play.
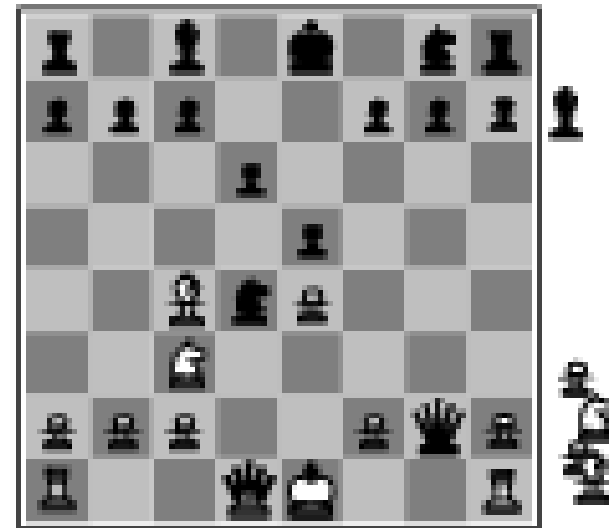
Standard approach:

- *cutoff test*

  e.g., depth limit (perhaps add *quiescence* test)

- *evaluation function*

  Gives estimated desirability of position. (Heuristic)

Evaluation functions

Black to move

White slightly better

White to move

Black winning

For chess, typically *linear* weighted sum of features:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) − (number of black queens)

etc.

Problem of Cutting Off Search

There is an inherent danger in stopping lookahead at some limited depth. Something bad may happen shortly after that depth.

This is more likely if the game is in a phase where things are changing fast (e.g. a multi-piece tradeoff in chess).

Quiescence

When a game is in a phase of play where the available reasonable moves only make small differences to the strength of either players position, the game is said to be *quiescent*.

The performance of lookahead minimax may be significantly improved by using some measure of quiescence to vary the depth of lookahead accordingly.

Quiescence

When a game is in a phase of play where the available reasonable moves only make small differences to the strength of either players position, the game is said to be *quiescent*.

The performance of lookahead minimax may be significantly improved by using some measure of quiescence to vary the depth of lookahead accordingly.

However, the *Horizon Problem* may cause serious problems for some kinds of game, and can make quiescence misleading.
The problem arises when there is some bad consequence that will happen after a long sequence of uneventful moves.

Humans may be better at spotting bad things on the horizon than are brute force search techniques.

Some Other Limitations of MiniMax

Does not take account of the fact that the opponent may not play as expected (may use different evaluations or make mistakes).

With depth limited MiniMax, it is only as good as the game state evaluation heuristic. This may be a crude measure for the more subtle games.

Will not perform well when there are many choices that lead to only slightly different states, that cannot easily be differentiated by heuristics. (e.g. Slow developing games such as Go.)

Conclusion

- Minimax is a powerful algorithm that is relatively simple to implement.

- It achieves *perfect play* for games that are simple enough for the algorithm to search right to its possible end states.

- However, for most games of reasonable complexity, resource limits mean that the depth of search has to be limited.

- Heuristic game state evaluations are used instead of end states.

COMP2240

Artificial

Intelligence

Lecture AG-3

α-β Pruning

The General Idea

The power of Minimax is limited by the huge size of game tree that arises for deep lookahead.

Much of this tree is actually redundant because we may know that a state will never be reached because of choices that could be made earlier in the game.
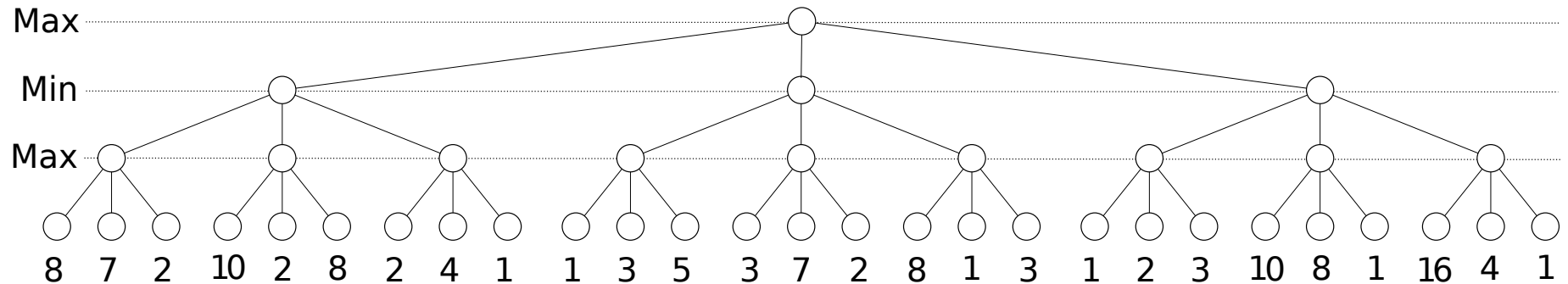
α-β pruning systematically eliminates a certain type of redundancy.
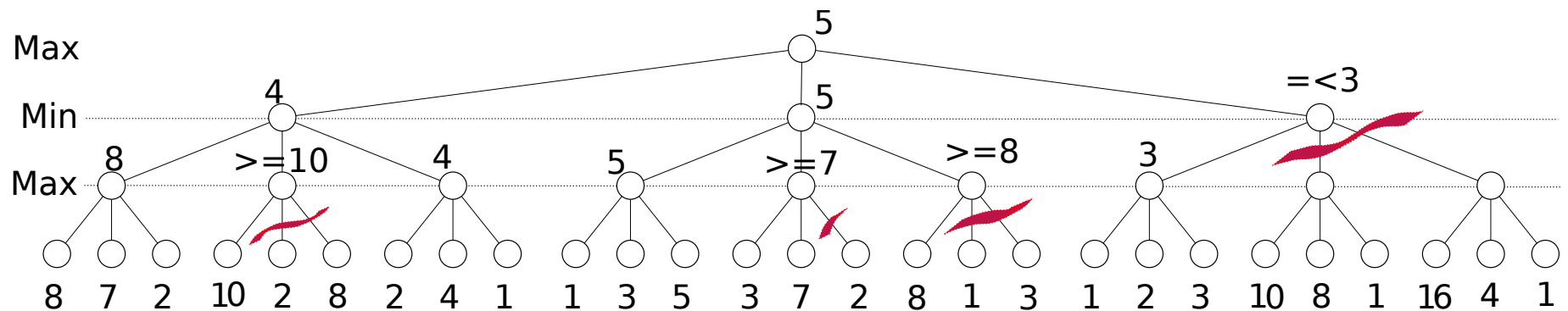
It does not affect the result of the Minimax calculation.

α-β Pruning
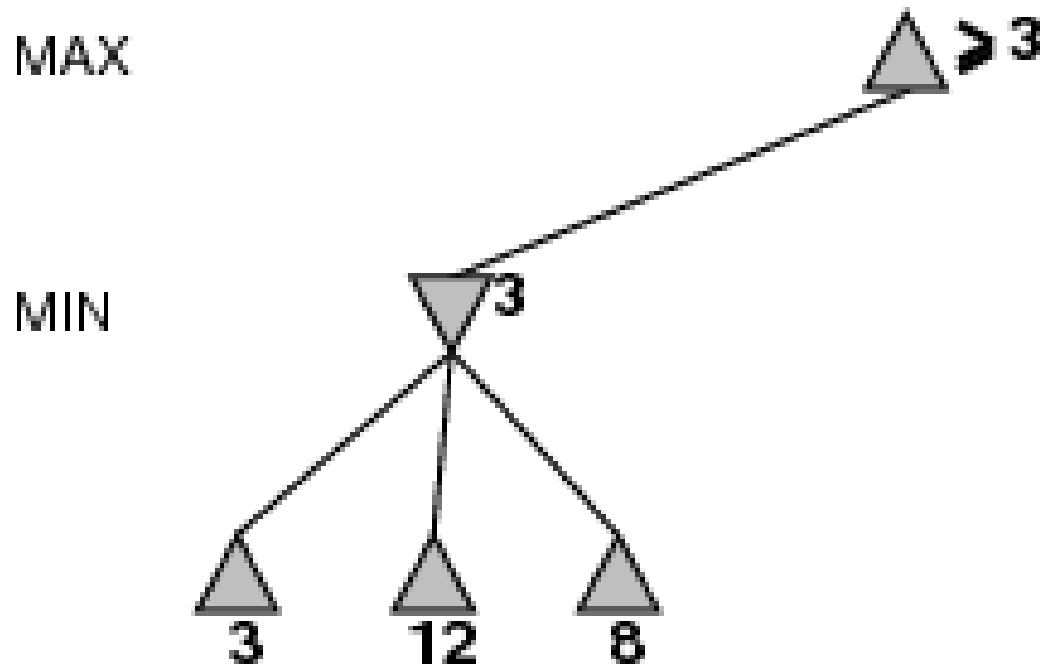
Calculated minimax evaluation:

Max

Min

Max

8 7 2 10 2 8 2 4 1 1 3 5 3 7 2 8 1 3 1 2 3 10 8 1 16 4 1

Reduction of branches to check using α-β pruning:

Max

Min

Max

8 7 2 10 2 8 2 4 1 1 3 5 3 7 2 8 1 3 1 2 3 10 8 1 16 4 1
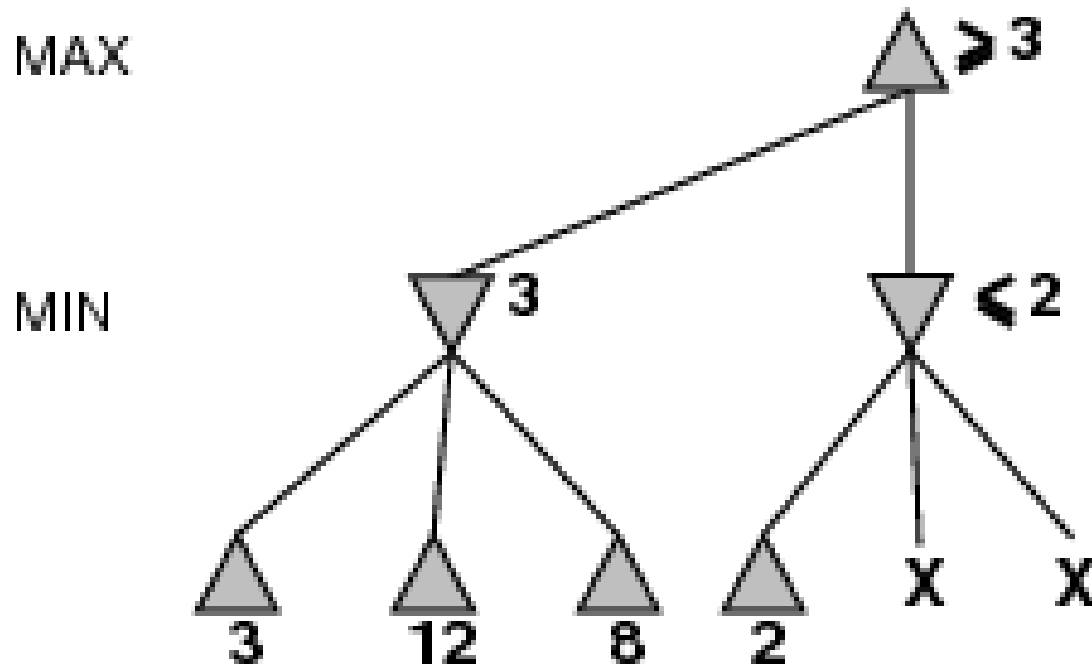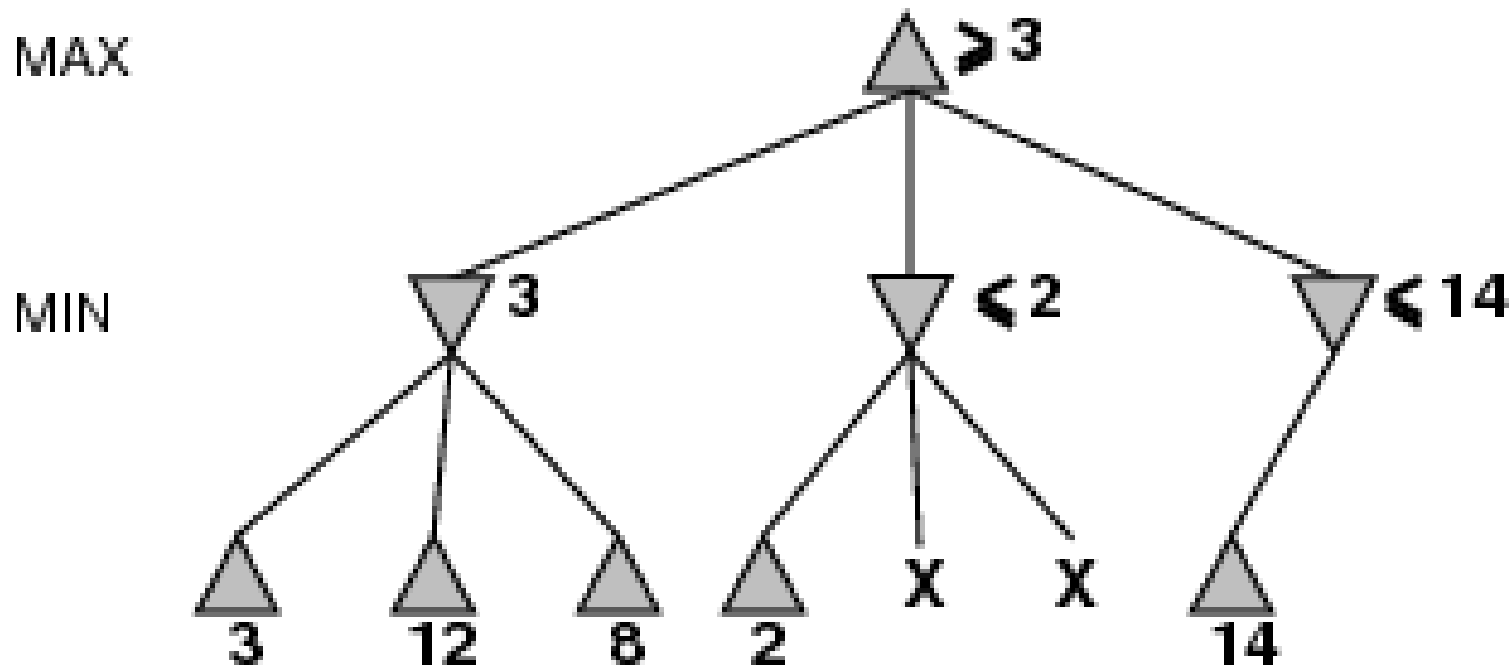
α-β **Pruning, Step by Step — 1**

(Note that in illustrating α-β pruning, we always assume left to right search of the tree.)
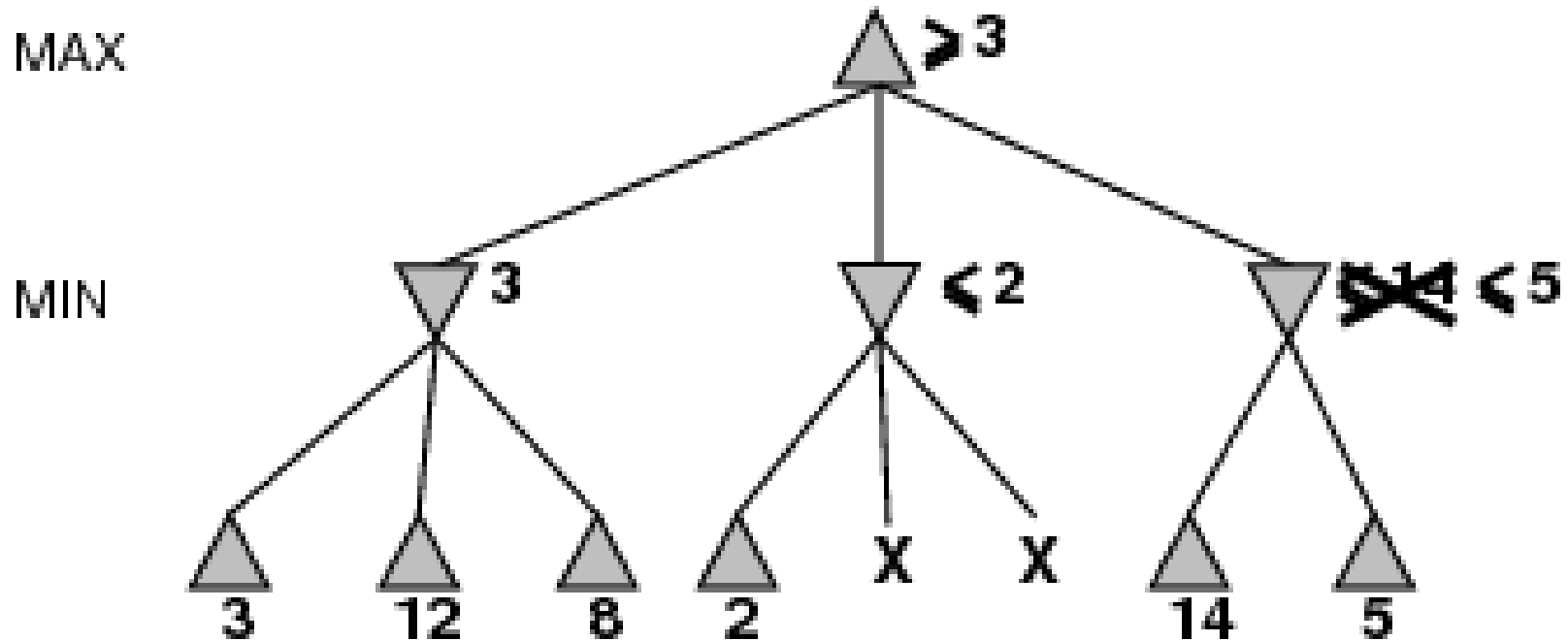
α-β Pruning, Step by Step — 2

(The effect of α-β pruning will vary depending on the order in which the choices are searched.)

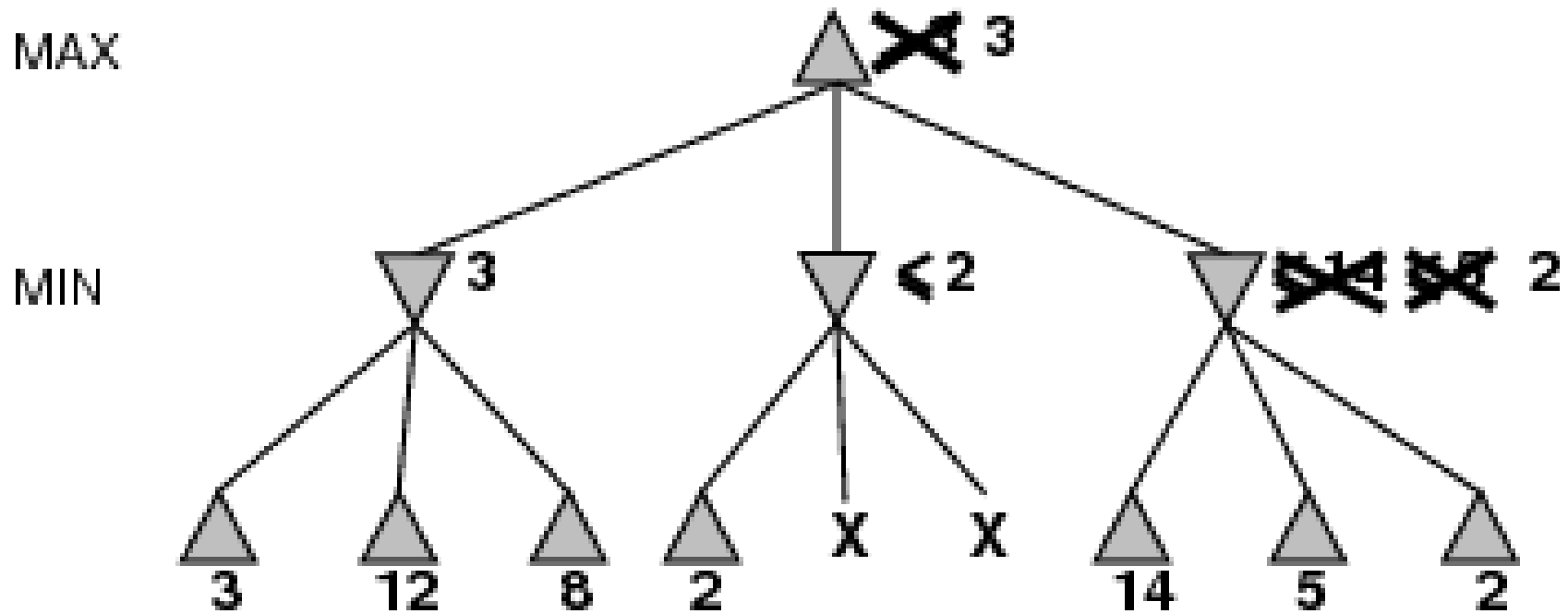# α-β **Pruning, Step by Step — 3**

(It is difficult to tell in advance which ordering of searching move choices will give the best pruning.)

MAX

MIN

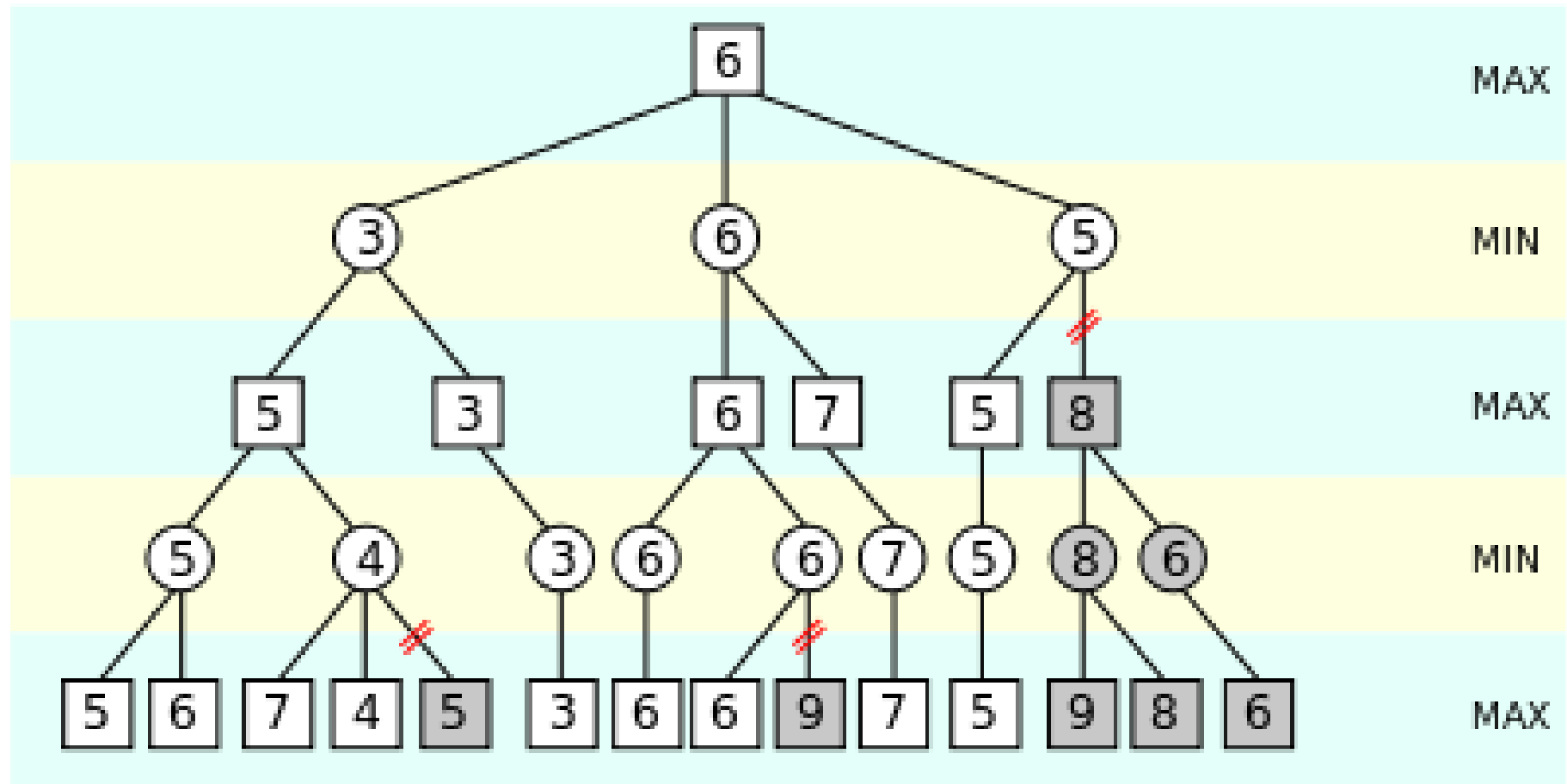# A Deeper Example
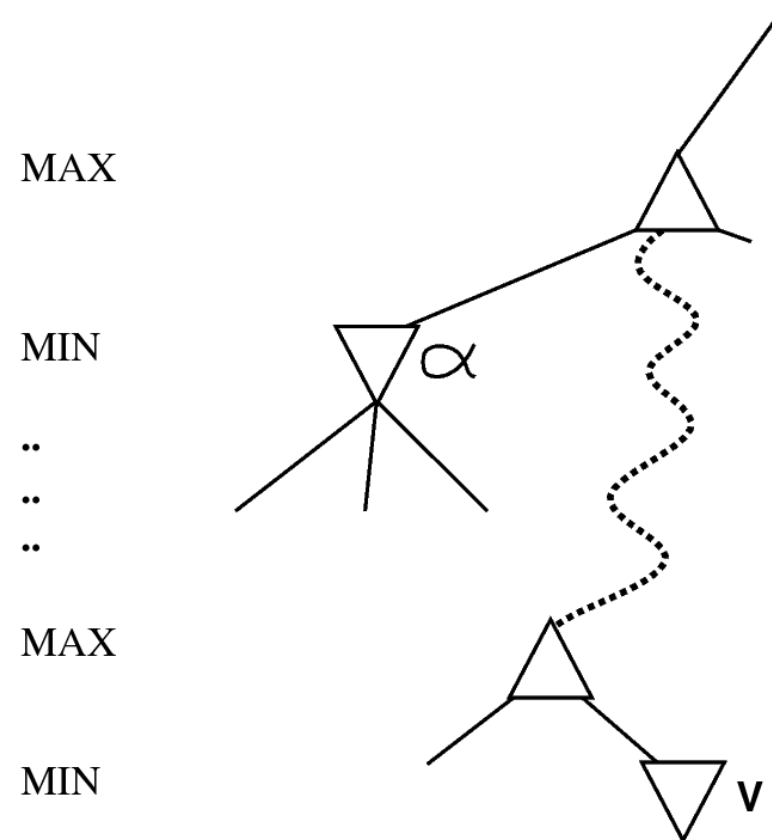
α-β **Pruning — the general case**

If α is better than v, then, when playing according to MiniMax, state v will never be reached. So there is no point considering any further continuations from v.

α-β **Pruning Algorithm**

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**

    $\alpha \leftarrow$ MAX(α, MIN-VALUE(*s*, *game*, α, β))

    if $\alpha \geq \beta$ **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**
>> $\beta \leftarrow$ MIN(β, MAX-VALUE(*s*, *game*, α, β))
>>   **if** $\beta \leq \alpha$ **then return** α
>   **end**
>   **return** β

COMP2240

Artificial Intelligence

Lecture AG-4

Randomness and Uncertainty
(and Pac-Man)

Randomness *vs* Uncertainty

Although closely related *randomness* and *uncertainty* are not the same thing.

The term 'randomness' is usually applied to situations where the range and frequency of outcomes is known. For instance, when rolling a standard die, each of the numbers 1–6 occurs with 1/6 probability

Taking into account the randomness of a dice roll is in most cases easier to model than the uncertainty of not knowing what your opponent will do in a given situation.

Uncertainty and Lack of Knowledge

Uncertainty also results from *lack of knowledge*.

A phenomenon may follow some pattern due to some underlying constraints or rules, or because of the strategy of an opponent.

If we do not understand why the pattern arises, it will be unpredictable (it will seem random).

Often uncertainty will involve both uncertainty due to randomness and uncertainty due to lack of knowledge.

Unknown Unknowns

... there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know. And if one looks throughout the history of our country and other free countries, it is the latter category that tend to be the difficult ones.

(Donald Rumsfeld — US Secretary of Defence, 2001–6).

In the setting of a game, we do not usually have unknown unknowns. We normally have a set of rules that defines all possible moves and board states. Even though some facts may be unknown (e.g. an opponent's hand of cards), we know what is possible and can reason about this.

This is one reason why AI approaches to reasoning about games may be less successful at reasoning about the actual world.
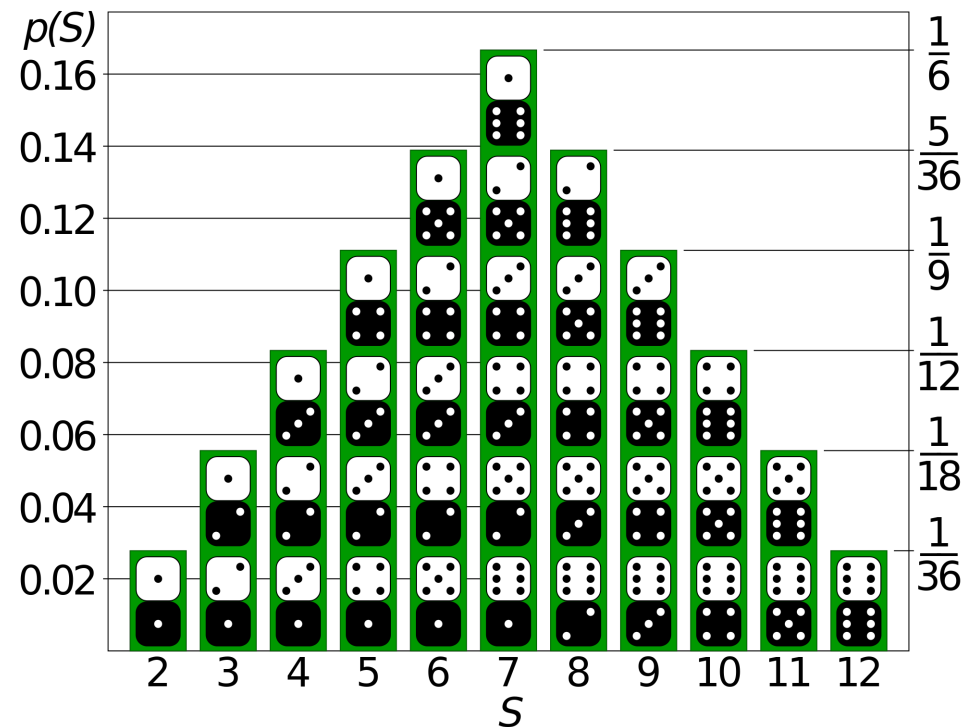
Dice Roll Probabilities

Many games involve rolling dice to determine some outcome or possibility.

Skillful play often requires understanding the relative liklihood of different outcomes and combining this with their value.

The value of position can often be expressed as a weighted sum of possible subsequent outcomes.
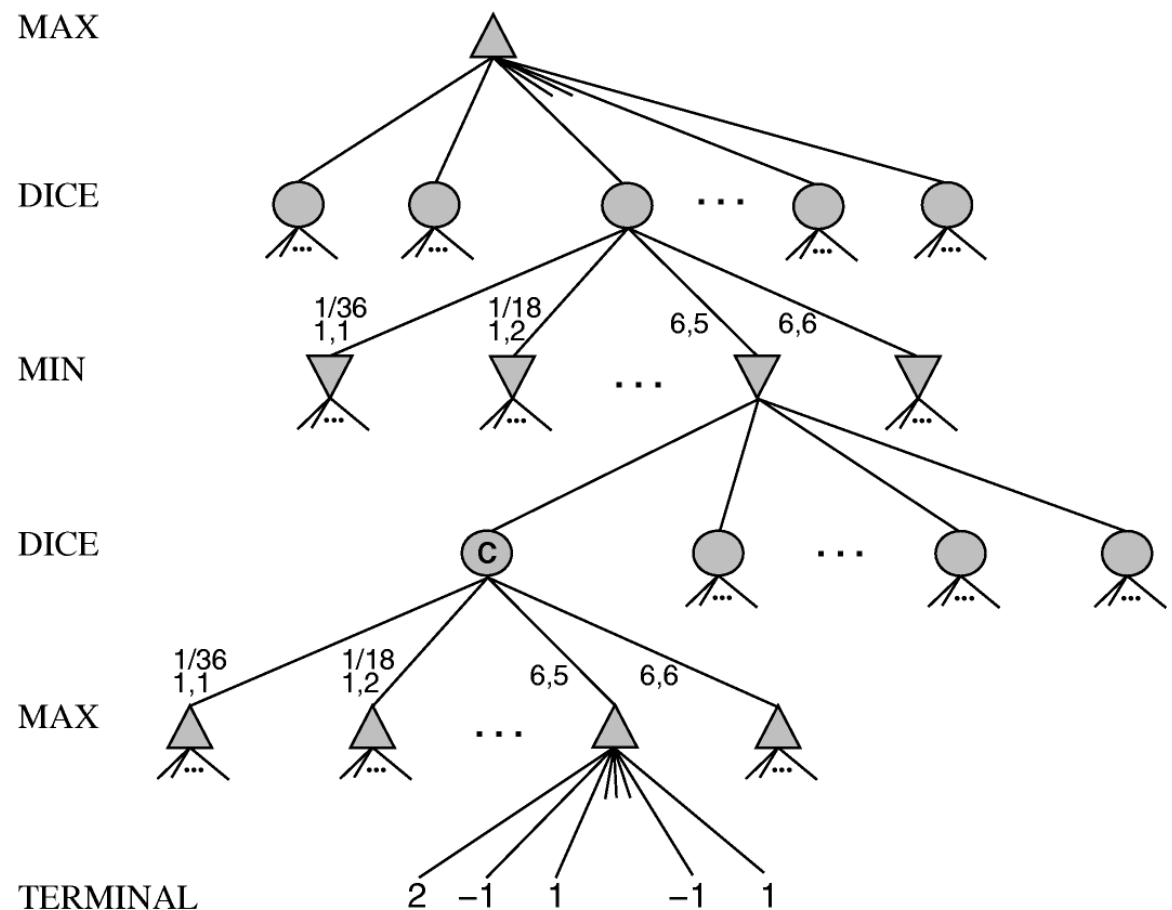

Computers can often outperform humans when it comes to making accurate calculations involving probabilities.

Minimax with Randomness

Chance events, such as dice rolls, can be represented within a game tree.

The minimax algorithm can be extended to incorporate probability-weighted sums of the values of subtrees below each possible outcome.

Rock, Paper, Scissors

Although rock-paper-scissors (RPS) may seem like a trivial game, it actually involves the hard computational problem of *temporal pattern recognition*.

This problem is fundamental to the fields of machine learning, artificial intelligence, and data compression. In fact, it might even be essential to understanding how human intelligence works.

Temporal patterns can be modelled by the use of *Markov Models*, which are widely used in AI and can be directly applied to games such as rock-paper-scissors.

One simple type of Markov Model is the *n-gram* model, which can easily be applied to rock-paper-scissors.

N-Grams for Predicting Sequences

An n-gram is a sequence of n items from some sequence.

The items could be numbers, letters, words or some kind of action (such as throws in a game of rock-paper-scissors).

An n-gram model is a frequency distribution over a (usually large) set of example sequences. For every combination $\langle x_1, \ldots, x_n \rangle$ of items, the frequency that this occurs in the example set is recorded.

This information can be used to determine the probability with which a particular item x_n will follow a preceding sequence of $n-1$ items:

$$P(x_n | x_1, \ldots, x_{n-1})$$

Pac-Man AI

Pac-Man is a good example of the use of very simple but effective AI in a computer game.

Movement

Each ghost is heading for a specific location (on a tiled grid).

Each of the four ghosts has a different way of picking its target tile:

- Red Blinky (pursuer) — Pac-Man's current tile.
- Pinky (speedy/ambusher) — four tiles in front of Pac Man.
- Inky (bashful/whimsical) — the tile equidistant and opposite to the position of Blinky, relative to the point two tiles in front of Pac-Man.
- Orange Clyde (pokey/ feigning ignorance) — If > 8 tiles from Pac-Man, target Pac-Man's current tile, otherwise head to bottom left corner (retreat).

Ghost Navigation

At each intersection, it chooses the way to go using the heuristic of *smallest Euclidean distance to goal* (calculated from the centre of each possible exit tile).
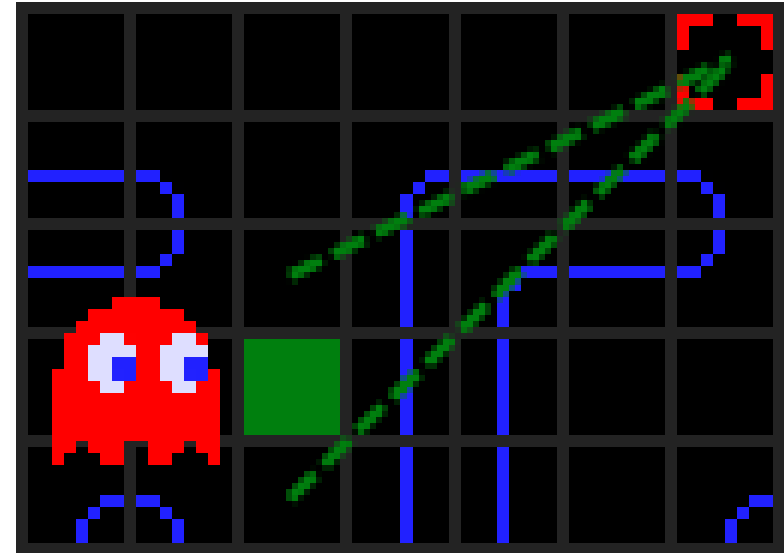
Why Pac-Man Starves

Sometimes looking ahead can result in dallying behaviour, where a beneficial action is never taken because it would always be possible to take it later with the same net benefit.
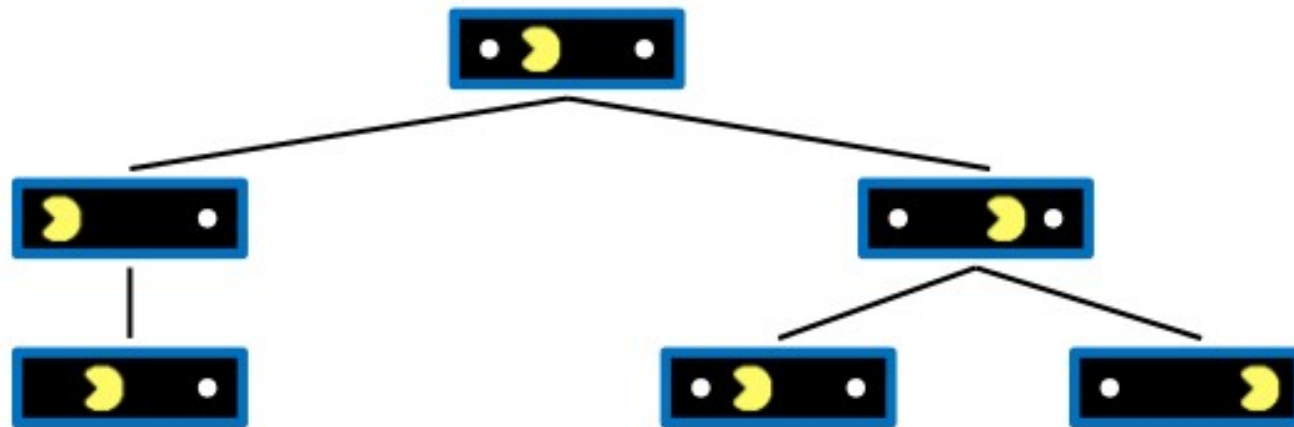
Pac-Man may choose not to move left and eat the energy blip, since it could always move right and eat the other energy blip on the following turn. Also moving right leaves more options open.

But reasoning like that will lead to starvation.