How to design AI for games

To go further

Florent Madelaine

IUT de Sénart Fontainebleau
Département Informatique

Part I

Recap from last lecture on games

Minimax Games

The games we will focus on will have the following properties.

- *Zero-sum* (The players are adversaries: there is no point in collaborating).
- *Perfect information* (no secret)
- A player has always only *finitely many possible moves*.
- The game always ends after a *finite sequence of moves*.
- *Deterministic* (no chance).
- *Two* players *alternate* play

We will call such games *minimax games* for short.

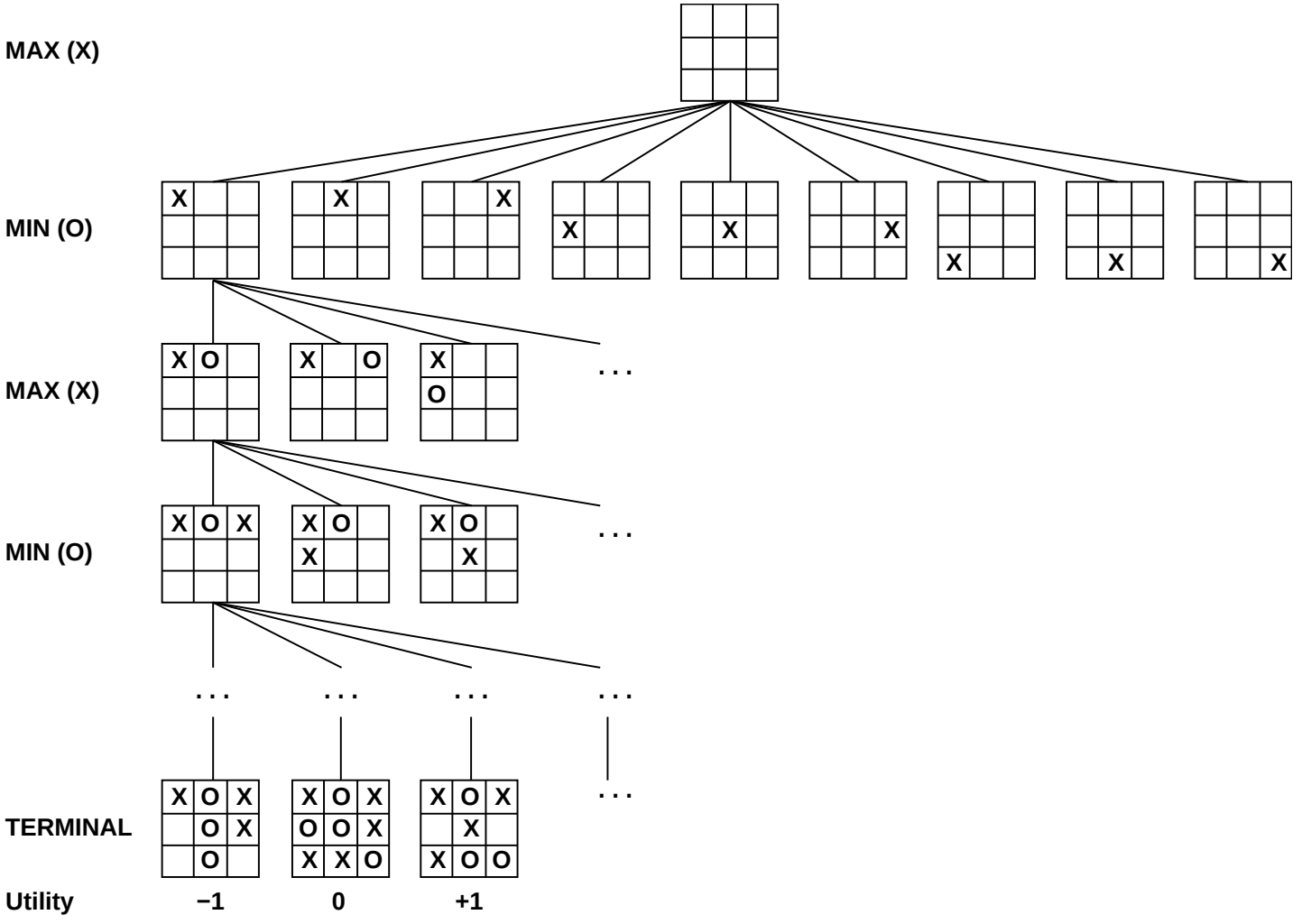
Game tree

- Nodes: game configuration
- Successors: configuration reachable in one *ply*
- Leafs: are endgame position labelled by a *payoff function*, *e.g.* Loss$=-1$, Draw$=0$ and Win$=+1$.

Remark

The game tree of a minimax game is finite.

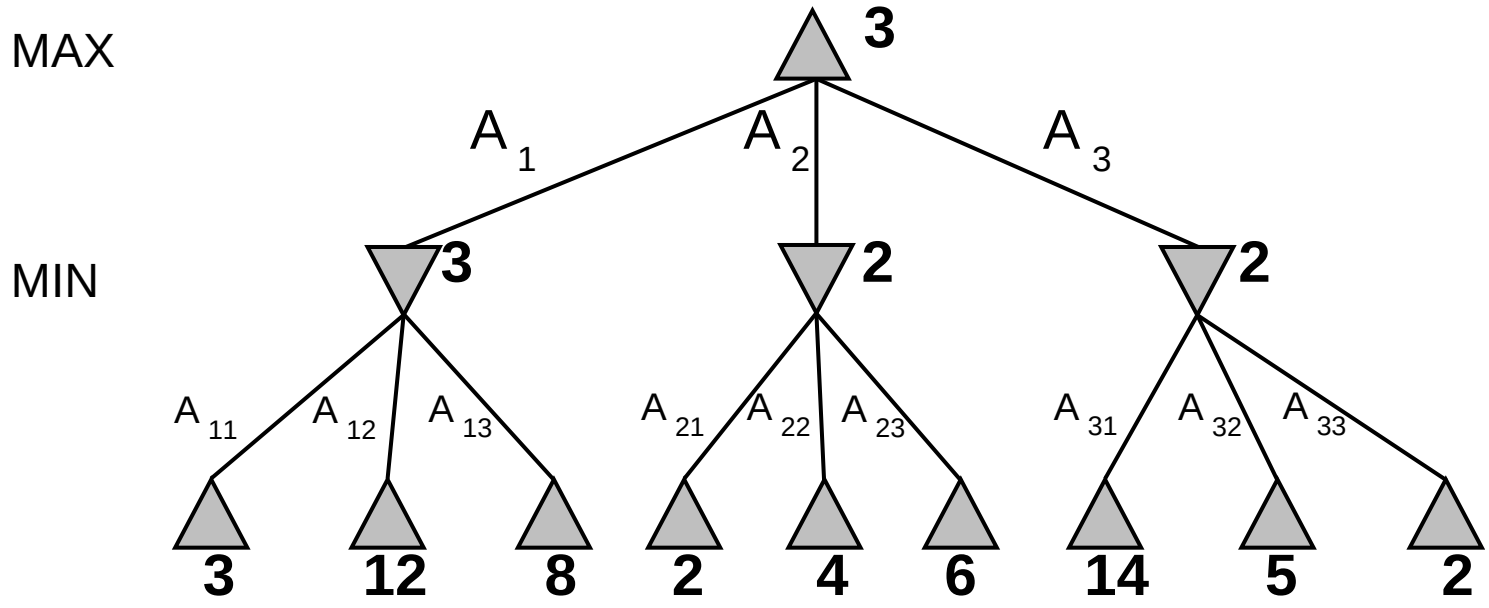
Example: Tic-Tac-Toe



Strategy

- A *strategy* is a method that allows a player to choose a move from any legal game position.
- Using our game tree model, it is simply a function from the set of nodes of the game tree which *selects a successor of a node*.
- A *winning strategy* for a player is a strategy which allows him to win **all the time no matter what the other player does**.
- A *non losing strategy* for a player is a strategy which allows him to never lose (win or draw) all the time no matter what the other player does.

What is the best strategy?



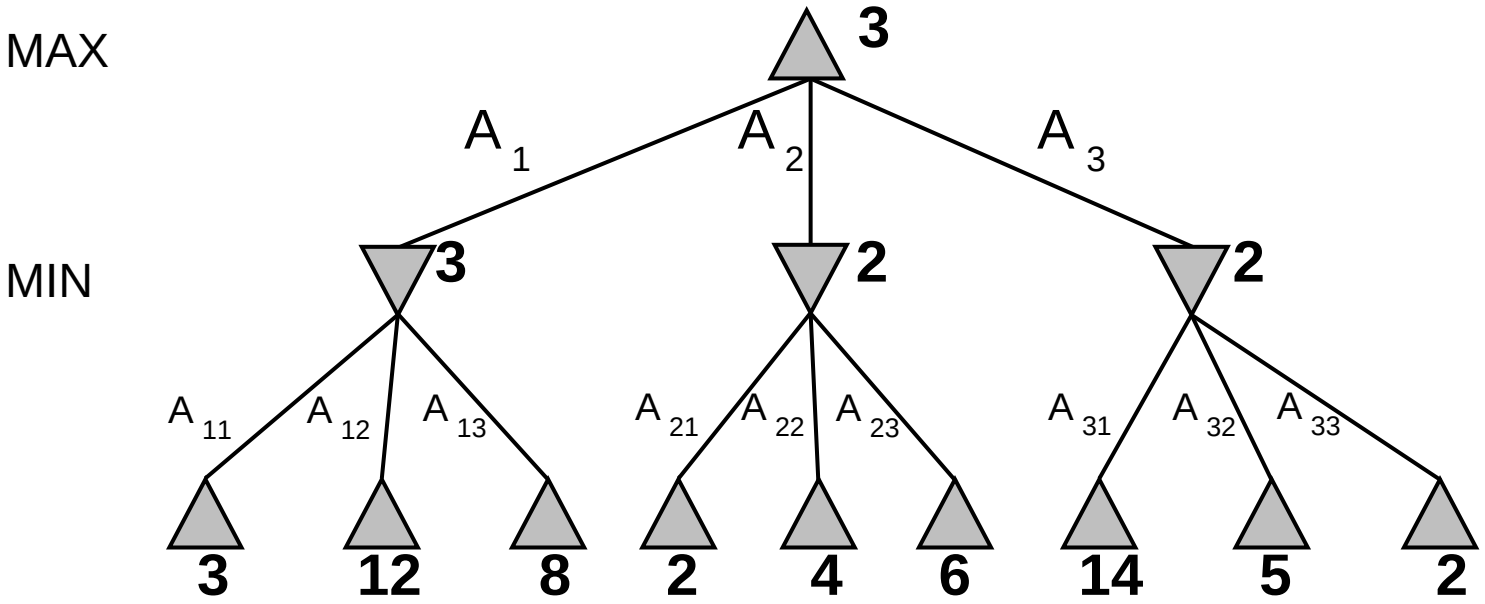
- The first player tries to maximise his payoff.
- The second player tries to minimise the first player's payoff.

Remark

The method works also when the pay off is not just win, draw or lose.

Minimax

- *Perfect play* for deterministic, perfect-information games.
- *strategy*: choose move to position with highest *minimax value*
- It is the *best achievable payoff against best play*.



How do we compute the best strategy?

- Drawing the game tree for a machine would mean to write in memory the whole tree.
- The game tree is usually very big.
- Instead we will only remember some of the tree and work out the best branch *recursively* in a *depth-first-search* fashion.

Properties

- In theory, the previous algorithm decides in finite time if the starting player has a winning strategy, provided that *the tree is finite*
- complexity (d = number of possible moves, h = height of the tree)
  - time: $\mathcal{O}(d^h)$
  - space: $\mathcal{O}(h)$

Part II

Today : advanced techniques to deal with huge game trees

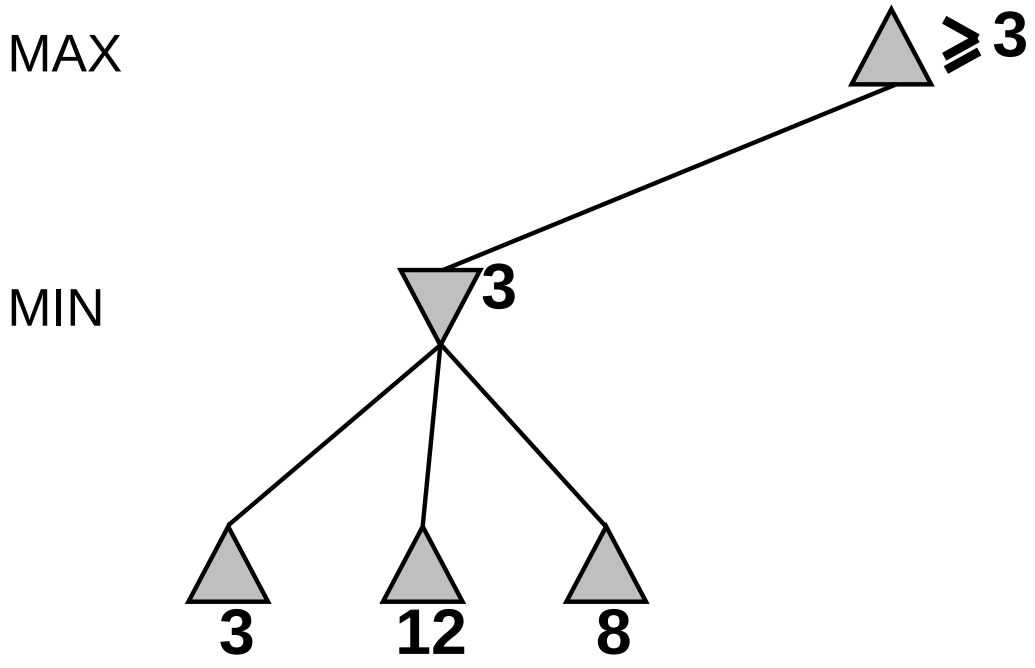
Outline

7. Exploring less for the same result
8. $\alpha - \beta$ pruning
9. Pushing the limit
10. Final products
11. Game With Chance
12. Conclusion

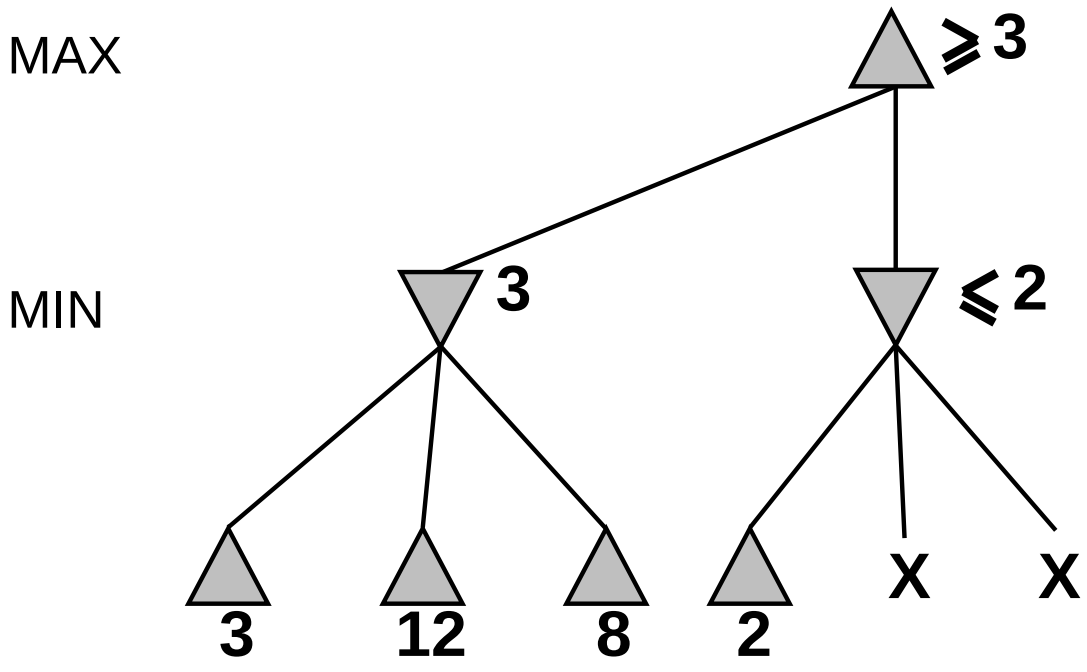
How to reduce search?

- We can use information about the game like symmetries of the board.
- There are a number of techniques which *do not depend of the game*. For example,
  - at a max node, we can abort search as soon as we know that one of the siblings gives us a win; and,
  - at a min node, we can abort search as soon as we know that one of the siblings gives us a loss.
- We can extend this idea to use more fully the information from the part of the tree we have explored: this is called *$\alpha - \beta$-pruning*.

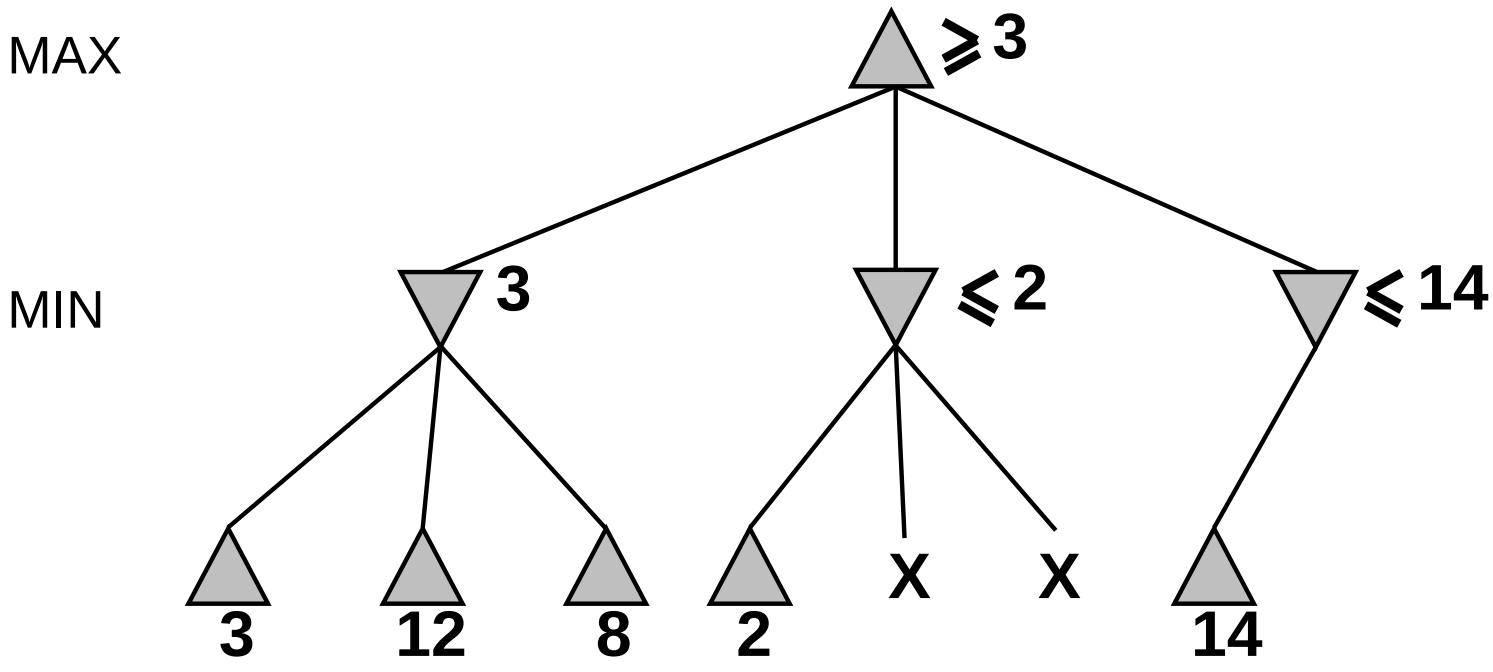
$\alpha - \beta$ pruning



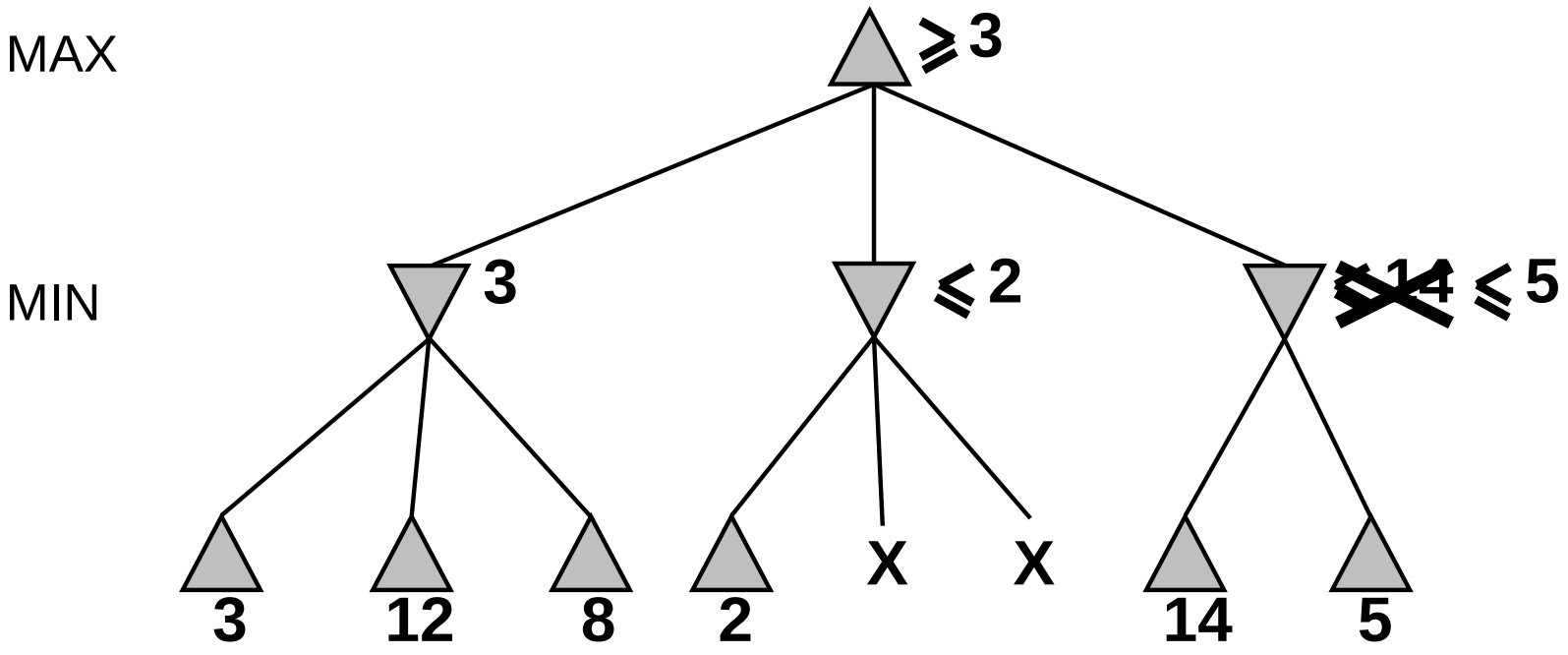
$\alpha - \beta$ pruning



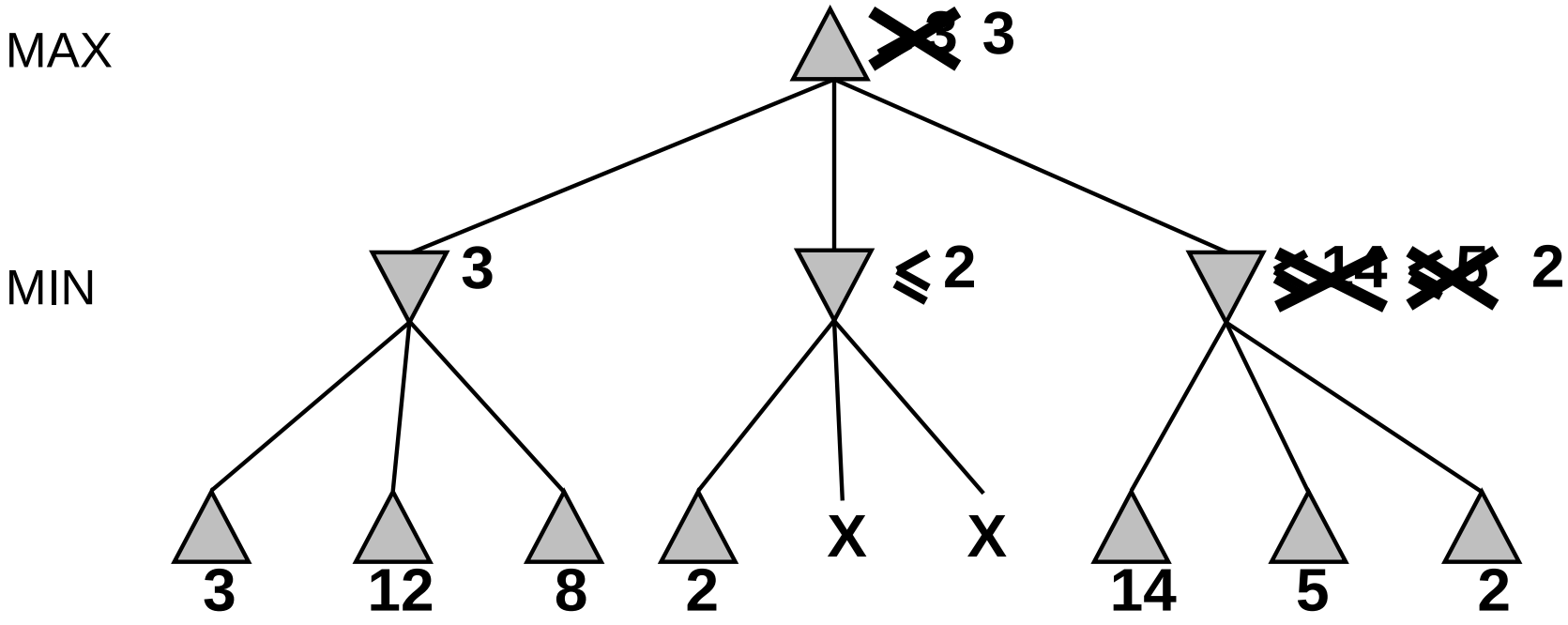
$\alpha - \beta$ pruning



$\alpha - \beta$ pruning



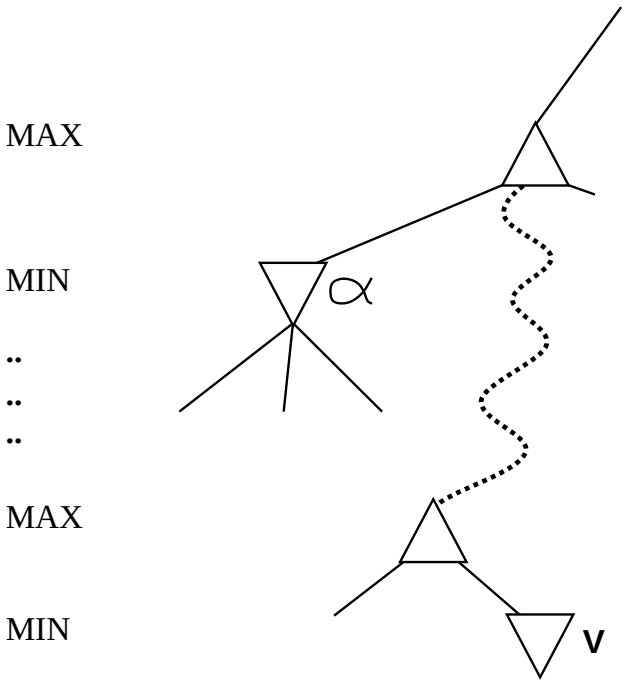
$\alpha - \beta$ pruning



Properties of $\alpha - \beta$

- Pruning *does not* affect final result.
- Good move ordering improves effectiveness of pruning.
- With "perfect ordering," time complexity = $O(d^{h/2})$.
- This effectively *doubles* the depth of search.
- We can easily reach depth 8 and start playing good chess.

Why is it called $\alpha-\beta$?



- α is the best value (to **max**) found so far off the current path
- If V is worse than α, **max** will avoid it and prune that branch.
- Define β similarly for **min**.

Recall Minimax with cutoff

ExploreMax(currentState,remainingDepth)

```
 1: if currentState is a terminal state then
 2:    return payoff(currentState)
 3: else if remainingDepth = 0 then
 4:    return eval(currentState)
 5: else
 6:    bestOutcome := −∞
 7:    for each successor nextState do
 8:       Outcome := ExploreMin(nextState, remainingDepth − 1)
 9:       if Outcome > bestOutcome then
10:          bestOutcome := Outcome
11:       end if
12:    end for
13:    return bestOutcome
14: end if
```

From Max nodes: increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local α := α// generalises bestOutcome.
7:    for each successor nextState do
8:       Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
         1, Local α, β)
9:       if Outcome > local α then
10:         local α := Outcome
11:         if Local α ≥ β then
12:            //further up in the exploration, my opponent(min) can play
               β
               // which is at least as bad for me.
13:         end if
14:      end if
15:   end for
16:   return Local α
17: end if
```

From Max nodes: increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local α := α// generalises bestOutcome.
7:    for each successor nextState do
8:       Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
         1, Local α, β)
9:       if Outcome > local α then
10:         local α := Outcome
11:         if Local α ≥ β then
12:            //pruning step
13:         end if
14:      end if
15:   end for
16:   return Local α
17: end if
```

From Max nodes: increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local α := α// generalises bestOutcome.
7:    for each successor nextState do
8:       Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
         1, Local α, β)
9:       if Outcome > local α then
10:         local α := Outcome
11:         if Local α ≥ β then
12:            //pruning step
13:            return ≥ β
14:         end if
15:      end if
16:   end for
17:   return Local α
18: end if
```

From Max nodes: increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local α := α// generalises bestOutcome.
7:    for each successor nextState do
8:       Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
         1, Local α, β)
9:       if Outcome > local α then
10:         local α := Outcome
11:         if Local α ≥ β then
12:            //pruning step
13:            return Local α
14:         end if
15:      end if
16:   end for
17:   return Local α
18: end if
```

From Min nodes: decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
 1: if currentState is a terminal state then
 2:    return payoff(currentState)
 3: else if remainingDepth = 0 then
 4:    return eval(currentState)
 5: else
 6:    Localβ := β// generalises worstOutcome
 7:    for each successor nextState do
 8:       Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
          1, α, Localβ)
 9:       if Outcome < localβ then
10:          localβ := Outcome
11:          if localβ ≤ α then
12:             //further up in the exploration, I (max) can play α
                // which is at least as good for me.
13:          end if
14:       end if
15:    end for
16:    return Localβ
17: end if
```

From Min nodes: decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local β := β// generalises worstOutcome
7:    for each successor nextState do
8:       Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
         1, α, Local β)
9:       if Outcome < local β then
10:         local β := Outcome
11:         if local β ≤ α then
12:            //pruning step
13:         end if
14:      end if
15:   end for
16:   return Local β
17: end if
```

From Min nodes: decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local-β := β// generalises worstOutcome
7:    for each successor nextState do
8:       Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
         1, α, Local-β)
9:       if Outcome < local-β then
10:         local-β := Outcome
11:         if local-β ≤ α then
12:            //pruning step
13:            return ≤ α
14:         end if
15:      end if
16:   end for
17:   return Local-β
18: end if
```

From Min nodes: decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return eval(currentState)
5: else
6:    Local-β := β// generalises worstOutcome
7:    for each successor nextState do
8:       Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
         1, α, Local-β)
9:       if Outcome < local-β then
10:         local-β := Outcome
11:         if local-β ≤ α then
12:            //pruning step
13:            return Local-β
14:         end if
15:      end if
16:   end for
17:   return Local-β
18: end if
```

What if your AI still plays badly?

- $\alpha - \beta$ allows to search deeper since we are not wasting time doing the same things.
- However, in many games, it is usual that we still play badly because we miss the information that lies after the cut off and because our evaluation heuristic can be quite bad sometimes.
- This is called *the horizon effect*.

Fighting the horizon effect

- Use every technology available.
- Use common knowledge of the problem.
- Improve the cutoff to something less naive than fixed depth.

Use every technology available

- *Dedicated hardware*: allows to search further (brute force).
- *Transposition table*: use hash table to store every position we compute, together with its evaluation (dynamic programming).
- *Endgame database*: precompute as many endgame as possible and store them.
- *Machine Learning*: play many games against yourself to learn from your mistake.

Use Common Knowledge of the Problem

- *Optimal Opening Table*: store all the usual opening together with the agreed reply and their variants, *e.g.* , *ouverture à la sicilienne*.
- *Evaluation function*: get this *je ne sais quoi* out of grand masters and put it into your evaluation function.
- *Learn from your opponent*: store all the games your opponent has ever played in an official competition and precompute games to find in advance way of beating him.

Quiescent Search

- A position is *quiescent* when it is unlikely to exhibit wild swings in the near future.
- It is dangerous when the search is cut off on a non quiescent positions.
- More sophisticated cutoff tests are needed: add a *quiescence search*.
- *e.g.* for chess, if the evaluation function involves counting material, then the *quiescence search* involves considering only capture moves

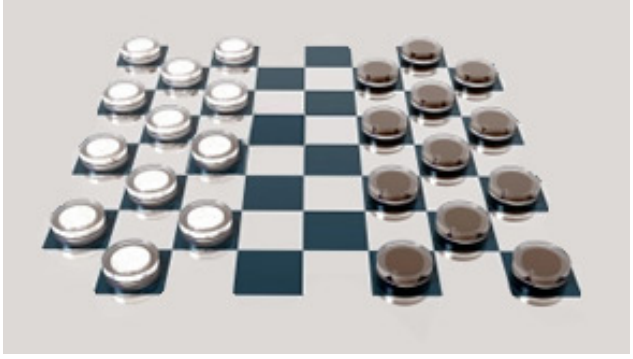
Other Search Improvements

- *Singular extensions*: expand nodes which are clearly better.
- *Forward pruning*: use heuristic to remove moves (dangerous).
- *Futility pruning*: helps decide in advance which move will cause a beta cutoff in the successor nodes.

Checkers

Boring now: perfect play leads to a draw.

- Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994.
- Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions (1995).
- A lot of work later: Perfect play by both sides leads to a draw (2007).



Chess

Better or same level as grand Masters.

- Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997.
- Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to *40* ply.

- Major advance are also due to the improvement of the software: in 2002, the eight game match between Vladimir Kramnik and a mere PC ended in a draw.

Othello

Humans too bad and Computers too good.

- The search space is relatively small: $5 \leq b \leq 15$ and $m = 64$.
- There was little expertise to develop the evaluation function.
- But now, the human champions refuse to compete against computers, who are too good.

- Recent advances: very advanced evaluation functions were designed using a vast number of automatically generated patterns and Neural Networks techniques to learn optimal weights for the evaluation functions (Buro 1997).

Othello

Humans too bad and Computers too good.

- The search space is relatively small: $5 \leq b \leq 15$ and $m = 64$.
- There was little expertise to develop the evaluation function.
- But now, the human champions refuse to compete against computers, who are too good.

- Recent advances: very advanced evaluation functions were designed using a vast number of automatically generated patterns and Neural Networks techniques to learn optimal weights for the evaluation functions (Buro 1997).

Werbung

Stefan Zweig “Schachnovelle”

Und da ich nichts anderes hatte als dies unsinnige Spiel gegen mich selbst, fuhr meine Wut, meine Rachelust fanatisch in dieses Spiel hinein. Etwas in mir wollte Recht behalten, und ich hatte doch nur dieses andere Ich in mir, das ich bekämpfen konnte; so steigerte ich mich während des Spiels in eine fast manische Erregung.

A digression: genetic methods to learn to play better

The idea: survival of the fittest

- Start with a population: an individual is represented by a chromosome.
- Look at the fitness of each individual
- Select some of the individuals (according to their fitness)
- Alter the population (crossover & mutations).
- Start over.
  - Population = heuristics for the game (based on a single heuristic)
  - chromosome = values of the weights
  - Fitness of individual $i = \frac{\#\text{gameswonby}i}{\sum_{j\in\text{Population}} \#\text{gameswonby}j}$

Go

Computers were very poor last millenium

- One of the oldest known strategy game, from China, now mostly played in Japan.
- The search space is huge: the board is very large $19^2 = 361$ and initially $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

- The human champions refused to compete against computers until 2000, who were really *too bad*.

Go

Computers were very poor last millenium

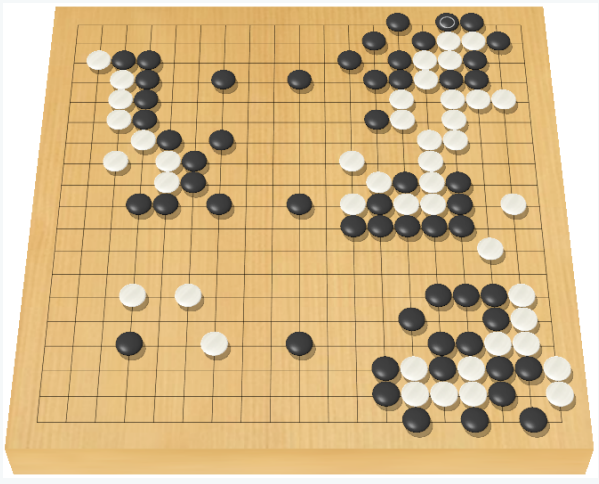
- One of the oldest known strategy game, from China, now mostly played in Japan.
- The search space is huge: the board is very large $19^2 = 361$ and initially $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

- The human champions refused to compete against computers until 2000, who were really *too bad*.

Go

Computers were very poor last millenium

- One of the oldest known strategy game, from China, now mostly played in Japan.
- The search space is huge: the board is very large $19^2 = 361$ and initially $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

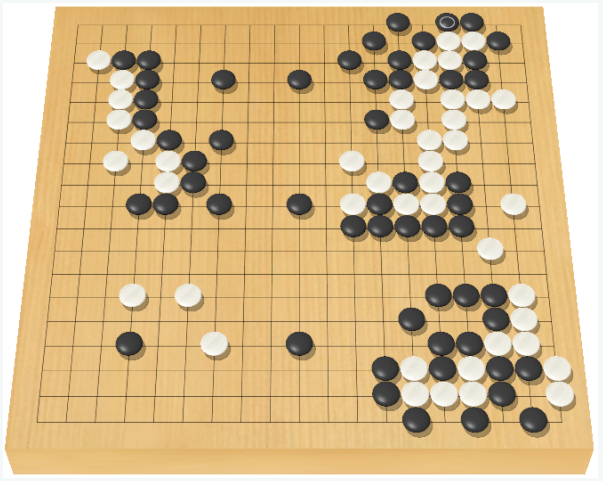


- The human champions refused to compete against computers until 2000, who were really *too bad*.

Go

Computers were very poor last millenium

- One of the oldest known strategy game, from China, now mostly played in Japan.
- The search space is huge: the board is very large $19^2 = 361$ and initially $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.

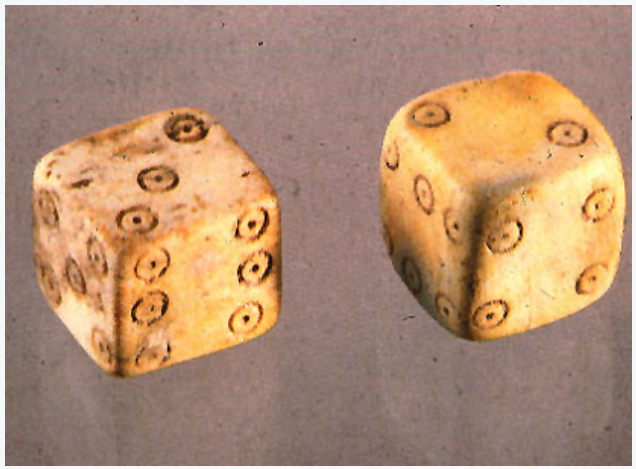


- The human champions refused to compete against computers until 2000, who were really *too bad*.

Interesting new method different from $\alpha-\beta$

Monte Carlo Tree Search

- Recent new method in planning (*multiarmed bandit problem*, Auer *et. al* 2002).
- Can be applied to games (Kocsis *et. al* 2006).
- It is much more efficient for Go than $\alpha-\beta$ and has lead to dramatic improvements (Teytaud *et. al*).
- It can be stopped at any time contrarily to $\alpha-\beta$ (breadth-first search rather than depth-first search)



Go

Humans and Computers as good as one another (on a 9x9)

- Currently, MoGo the world champion of programs has not only managed to win games against strong amateurs but has managed a draw (2 games each) versus Motoki Noguchi a professional, the current French number 1 (*December 2008, IUT Clermont-Ferrand*).

- However, it is only on a 9x9 board and using enormous ressources (25% of the national dutch supercomputer).

Go

Humans and Computers as good as one another (on a 9x9)

- Currently, MoGo the world champion of programs has not only managed to win games against strong amateurs but has managed a draw (2 games each) versus Motoki Noguchi a professional, the current French number 1 (*December 2008, IUT Clermont-Ferrand*).

- However, it is only on a 9x9 board and using enormous ressources (25% of the national dutch supercomputer).

Go

Humans and Computers as good as one another (on a 9x9)

- Currently, MoGo the world champion of programs has not only managed to win games against strong amateurs but has managed a draw (2 games each) versus Motoki Noguchi a professional, the current French number 1 (*December 2008, IUT Clermont-Ferrand*).

- However, it is only on a 9x9 board and using enormous ressources (25% of the national dutch supercomputer).

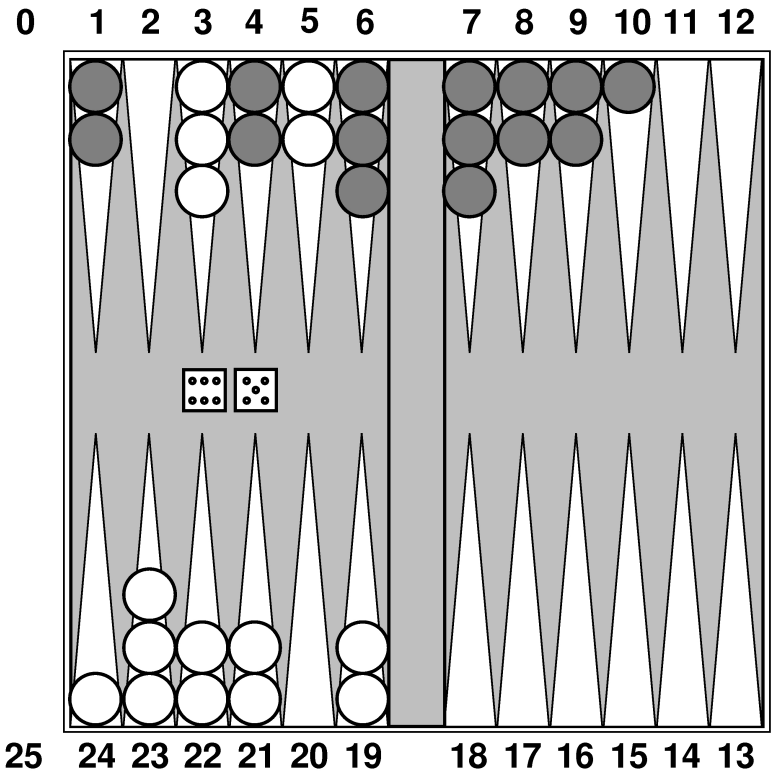
Go

Humans and Computers as good as one another (on a 9x9)

- Currently, MoGo the world champion of programs has not only managed to win games against strong amateurs but has managed a draw (2 games each) versus Motoki Noguchi a professional, the current French number 1 (*December 2008, IUT Clermont-Ferrand*).

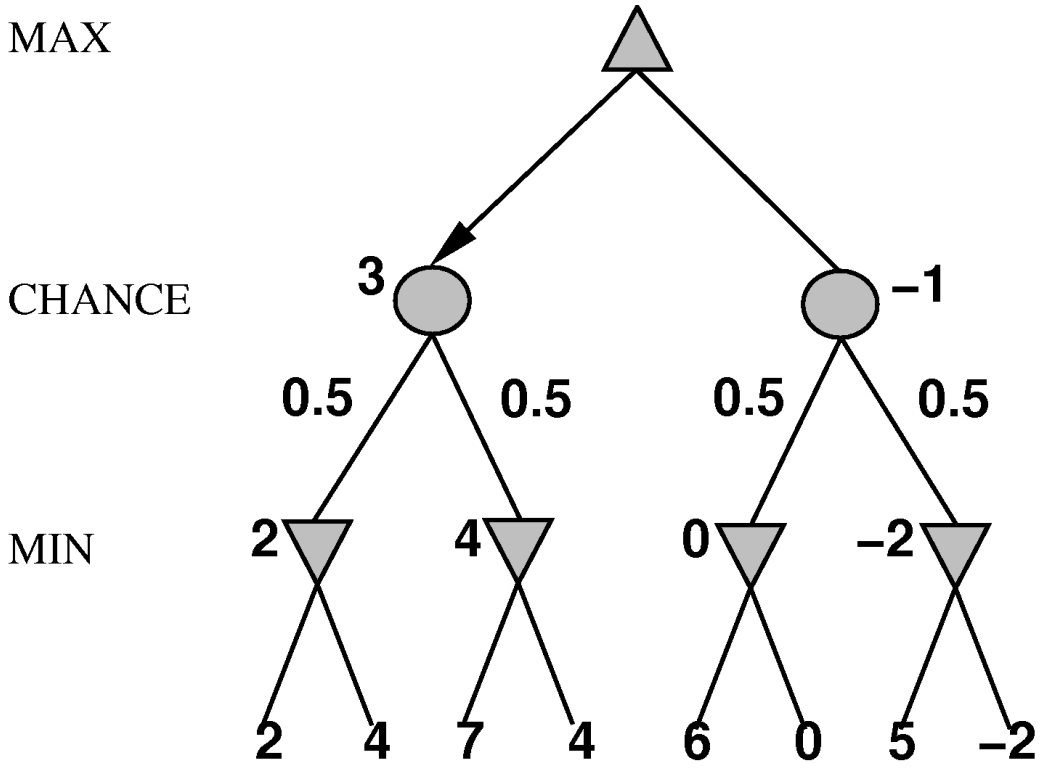
- However, it is only on a 9x9 board and using enormous ressources (25% of the national dutch supercomputer).

Nondeterministic games: backgammon



Nondeterministic games in general

- In nondeterministic games, chance is introduced by dice, card-shuffling ...
- Simplified example with coin-flipping.



Algorithm for nondeterministic games

- In the context of games with chance, perfect play is the strategy with the best *expected utility*.
- We can not do much against an opponent who is too lucky but on average over many games we will win some money if our strategy gives us a positive expected utility.
- Expectiminimax gives perfect play.
- It works just like Minimax, except we must also handle chance nodes.

Expectiminimax

ExpectMinimax(currentState)

if currentState is a terminal state **then**
  return *payoff*(currentState)
else if Chance Node **then**
  return $\text{average}_{\text{nextState}}$ {**ExpectMinimax**(nextState)}
else if I am to move **then**
  return $\max_{\text{nextState}}$ {**ExpectMinimax**(nextState)}
else
  return $\min_{\text{nextState}}$ {**ExpectMinimax**(nextState)}
end if

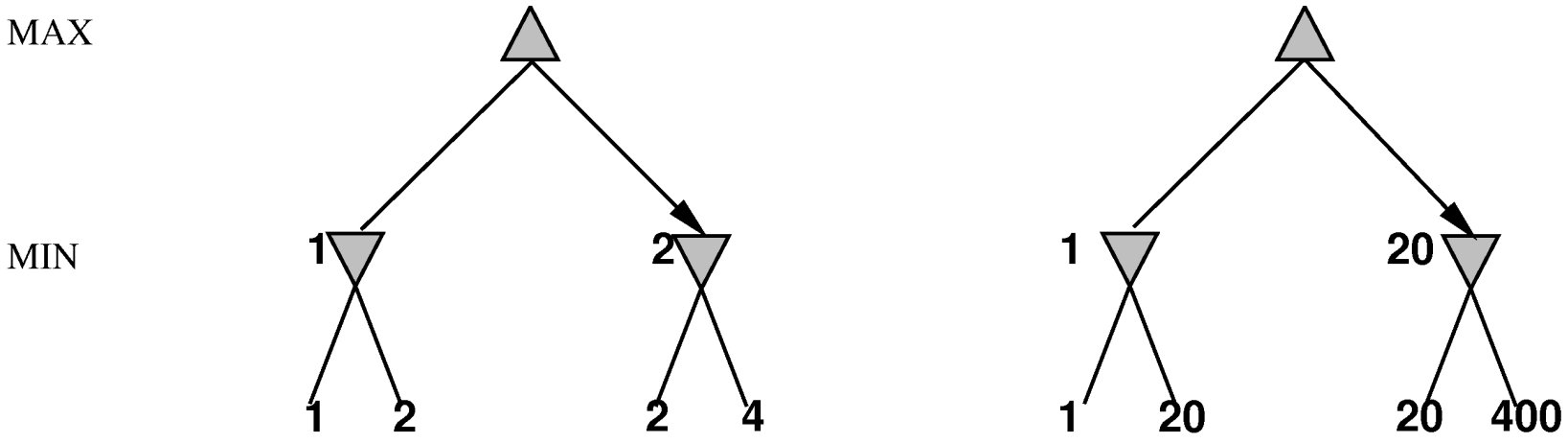
Nondeterministic games in practice

- Dice rolls increase the degree d: 21 possible rolls with 2 dice
- Backgammon $\approx$ 20 legal moves (can be 6,000 with 1-1 roll)

$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

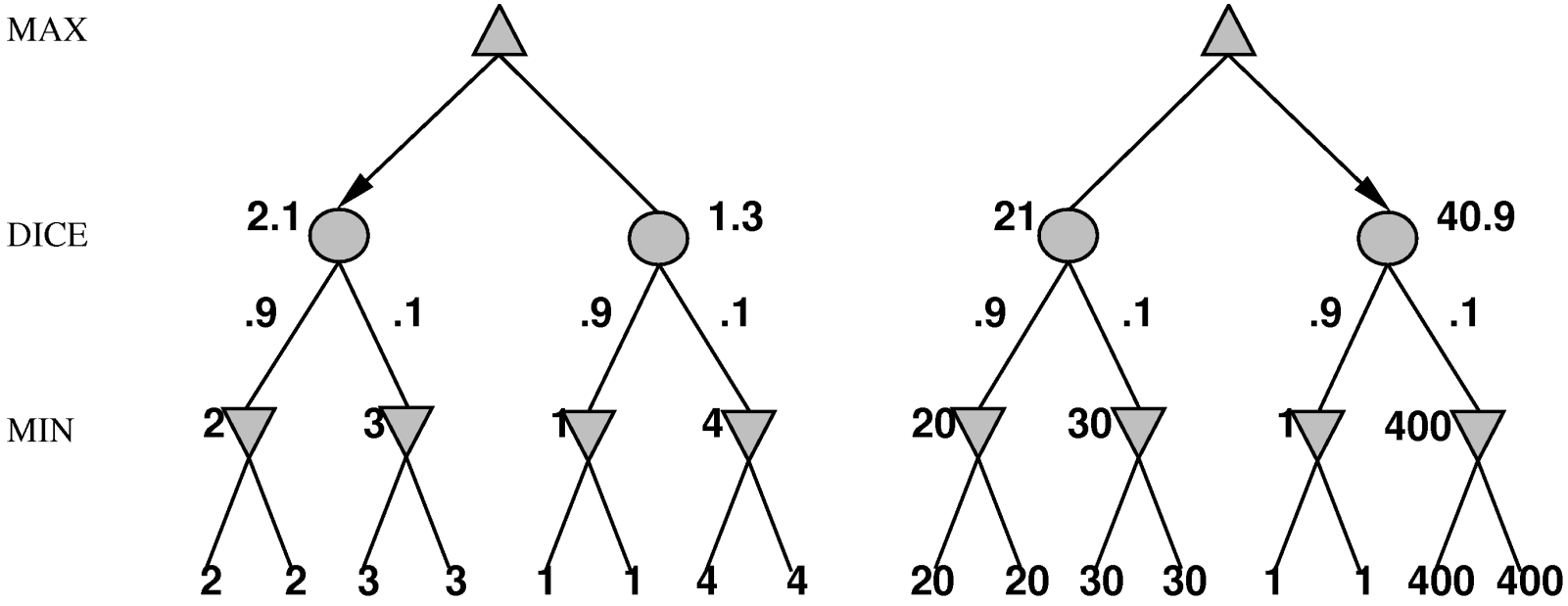
- As depth increases, the probability of reaching a given node shrinks, so the value of lookahead is diminished
- This means that α–β pruning is much less effective
- TDGammon uses depth-2 search + a very good Eval and is roughly at world-champion level.

Exact values don't matter for deterministic games



- Behaviour is preserved under any *monotonic* transformation of Eval
- Only the order matters: the payoff in deterministic games acts as an *ordinal utility* function.

Exact values DO matter for Games with Chance



- Behaviour is preserved only by a *positive linear* transformation of Eval
- Hence Eval should be proportional to the expected payoff.

Conclusion

- Perfect play is not practically possible for a number of interesting games.
- But, an opportunistic mix of good algorithmic methods and good. implementation of Human know-How leads to very good play.
- Games are unarguably a major success of **AI**.

La Fin

Sacha Guitry “Mémoires d’un tricheur”

Ce que les gens qui ne jouent pas ne savent pas, ce qu’ils ignorent, ce sont les bienfaits du jeu. Ses inconvénients, je les connais comme eux. Certes, c’est un danger, mais qu’est-ce qui n’est pas un danger dans la vie! Or, il ne faut pas contester l’influence excellente que le jeu peut avoir sur le moral. L’homme qui vient de gagner mille francs, ce n’est pas un billet de mille francs qu’il a gagné - c’est la possibilité d’en gagner cent fois plus. Il n’a pas gagné mille francs - il a gagné! Quand il perd mille francs, il n’a perdu que mille francs. Quand il les gagne, il a gagné les premiers mille francs d’une fortune incalculable. Tous les espoirs lui sont permis - et voyez cette confiance en lui qu’il a, c’est magnifique! En amour, en affaires, pendant vingt-quatre heures, il va tout oser - et ce début d’une fortune, dû au hasard uniquement, peut le mener à la fortune véritable.