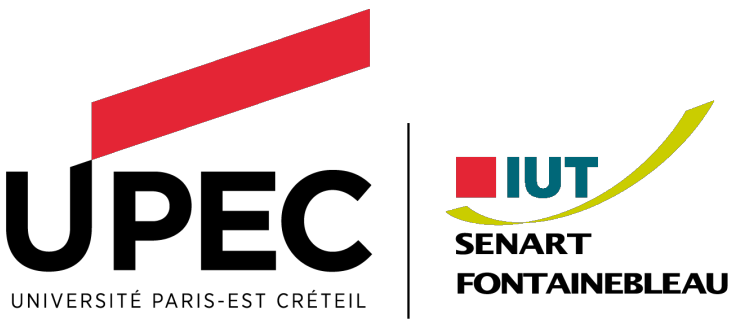
How to design AI for games

An Introduction

Florent Madelaine

IUT de Sénart Fontainebleau
Département Informatique

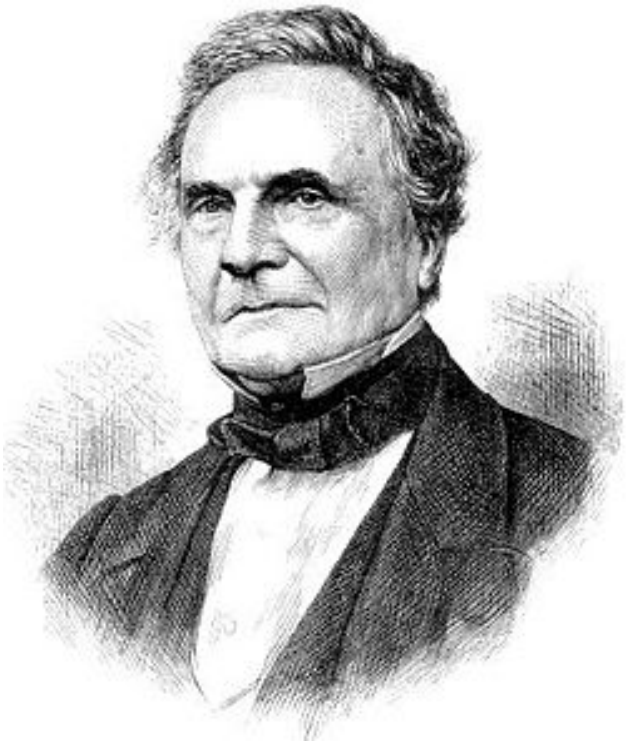


Outline

1. History
2. What is a game ?
3. Strategy
4. Minimax
5. Combinatorial Explosion
6. Using human knowledge

Early 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)



Early History

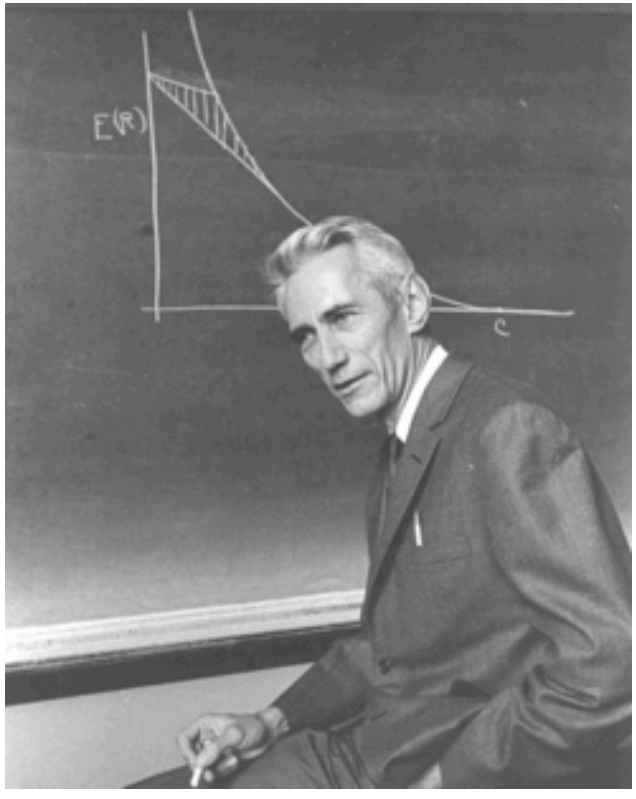
- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912 ; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945 ; Wiener, 1948 ; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Early 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)

Early 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)



Early History

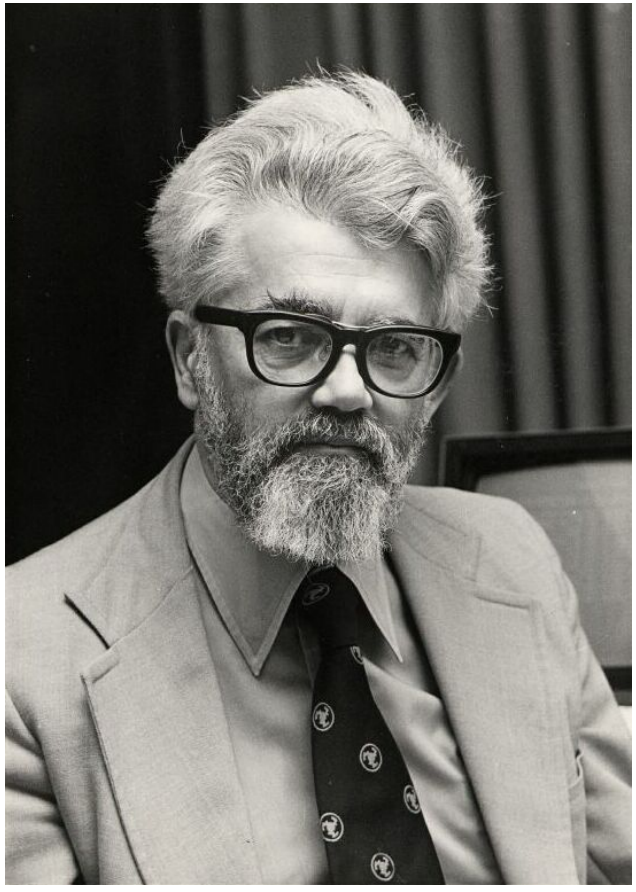
- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Early 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)

Early 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)



Recent History

- The best Othello computer programs have easily defeated the best humans since 1980, when the program *the Moor* beat the reigning world champion.
- The game Connect-4 was solved independently in 1988 by James D. Allen and Victor Allis. With perfect play, the first player can force a win by starting in the middle column. By starting in the two adjacent columns, the first player allows the second player to reach a draw ; by starting with the four outer columns, the first player allows the second player to force a win.
- In 1997, *Logistello* defeated the human champion Takeshi Murakami with a score of 6 :0.

Recent History

- In 1994, *Chinook* ended the 40-years-reign of human world champion Marion Tinsley at checkers. It 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.
- In 1997, *Deep Blue* defeated human world champion Gary Kasparov at chess in a six-game match. 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.
- Until 2000, human champions refused to compete against computers at Go, who were very very bad.

The recent case of Go in details

- A new idea for the AI design has been very successful from 2000 onward : Monte Carlo Tree Search.
- In 2008, the contest MoGo (an AI) versus Motoki Noguchi (the best human Go Player in France at the time) ended up in a draw (on a smaller board 9×9).
- ...
- In 2016, Lee Sedol the Human Go Champion is defeated 4-1 in a match against AlphaGo, an AI that built on ideas present in MoGo together with techniques from deep learning.

Different types of games

Perfect Information

Deterministic

checkers, othello, chess, go, *diplomacy*

With Chance

backgammon, monopoly, *risk*

Imperfect Information

Deterministic

battleships, stratego, *diplomacy*

With Chance

bridge, poker, scrabble, *risk*

special cases : risk and diplomacy

Risk : different aims

- conquer the world (perfect information)
- secret aim, *e.g. conquer continents x and y* (imperfect information)

Diplomacy and Risk : alliances or not

- In both games, when played on a board, human players will make temporary alliances (these are both multiplayer games in general).
- Depending of the implementation of the game, we have
  - no alliance authorised and no talking between players (perfect information)
  - alliances authorised (imperfect information)

Different types of games

Perfect Information

Deterministic

checkers, othello, chess, go, *diplomacy*

With Chance

backgammon, monopoly, *risk*

Imperfect Information

Deterministic

battleships, stratego, *diplomacy*

With Chance

bridge, poker, scrabble, *risk*

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.

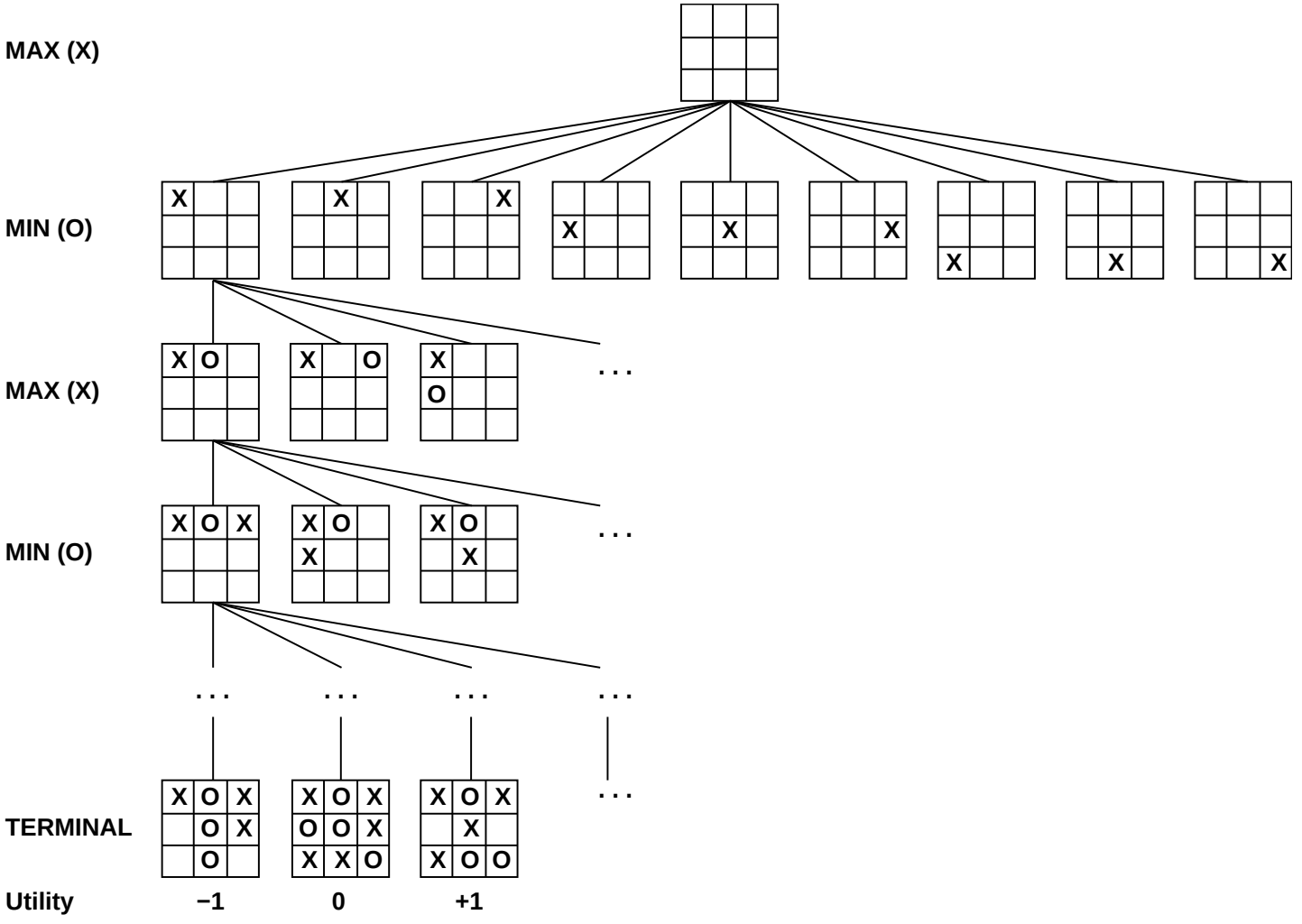
Game tree

- 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.

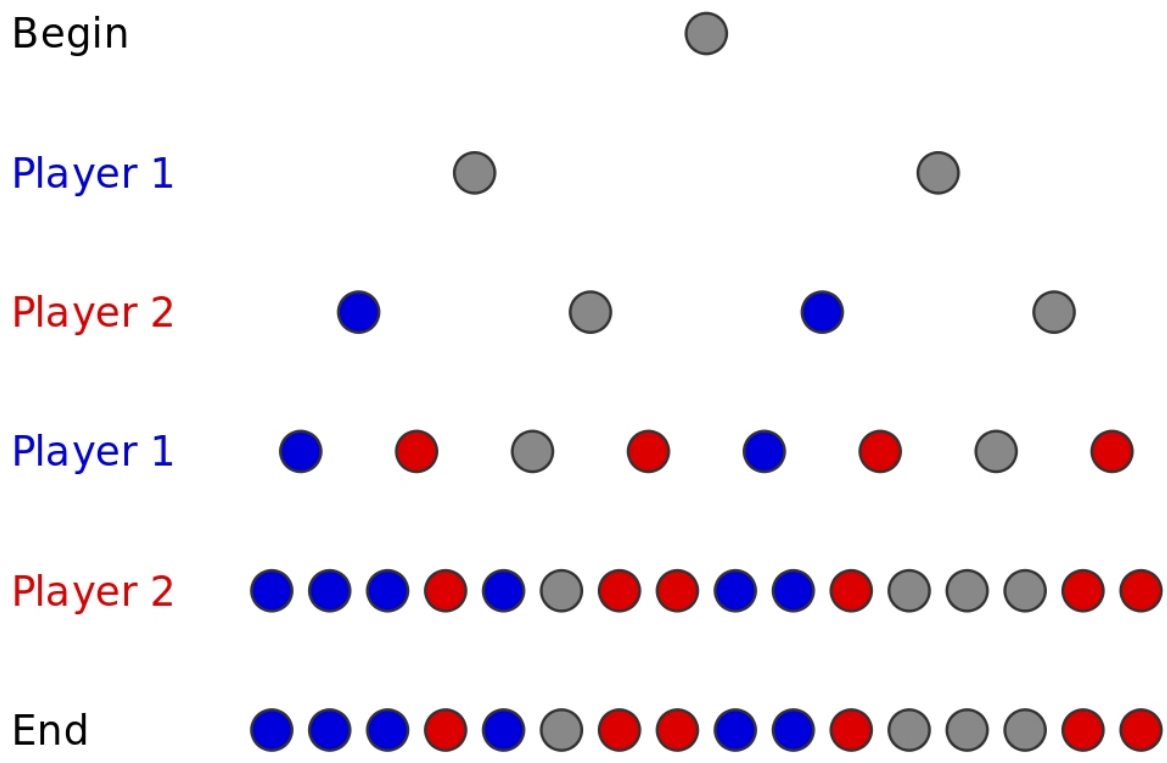
Example : Tic-Tac-Toe



Strategy

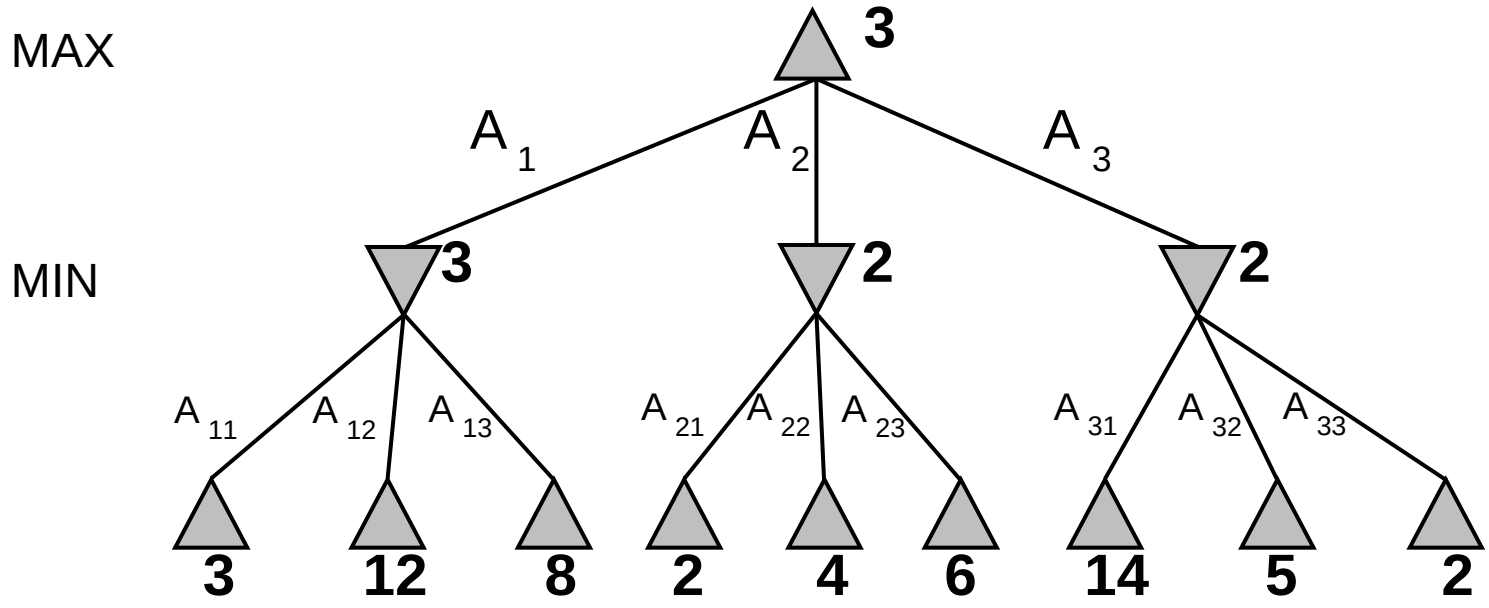
- A *strategy* is a method that allows a player to choose a move from any legal game position.
- Using our game tree model, it is simply a function from the set of nodes of the game tree which *selects a successor of a node*.
- A *winning strategy* for a player is a strategy which allows him to win **all 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.

Finding strategies on the game tree



- red = Player 1 has a winning strategy from here.
- blue = Player 2 has a winning strategy from here.
- gray = Both players have a non losing strategy.

What is the best strategy ?



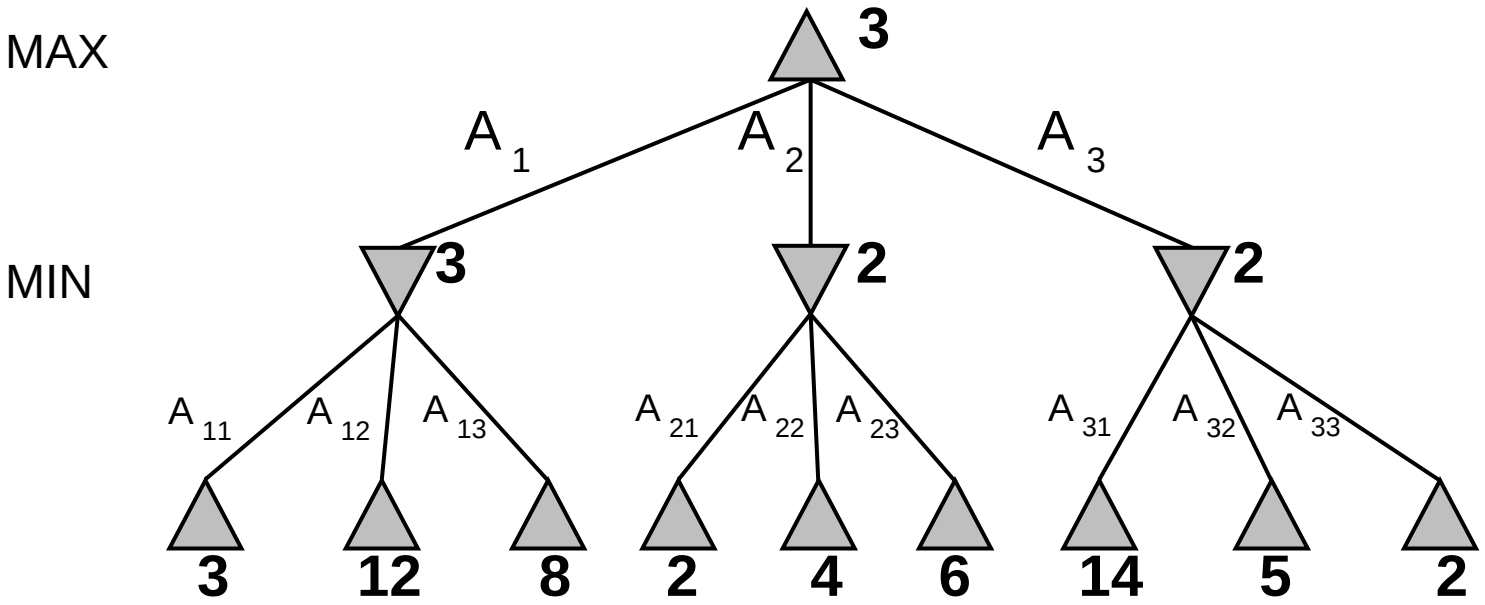
- The first player tries to maximise his payoff.
- The second player tries to minimise the first player's payoff.

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.

The algorithm

Minimax(currentState)

if currentState is a terminal state **then**
  return *payoff*(currentState)
else if I am to move **then**
  return $\max_{\text{nextState}}\{\textbf{Minimax}(\text{nextState})\}$
else
  return $\min_{\text{nextState}}\{\textbf{Minimax}(\text{nextState})\}$
end if

With one function for each player

ExploreMax(currentState)

```
if currentState is a terminal state then
  return payoff(currentState)
else
  return max_{nextState} {ExploreMin(nextState)}
end if
```

With one function for each player

ExploreMin(currentState)

if currentState is a terminal state **then**
  return *payoff*(currentState)
else
  return $\min_{\text{nextState}}$ {**ExploreMax**(nextState)}
end if

With one function for each player

First call

ExploreMax(`currentState`)

ExploreMax

ExploreMin

Remark

- the two functions are calling each other recursively until a leaf of the game tree is reached
- the values are propagated upward

Example

- Try some applets available on the web.
- For example with `minimax` on a complete tree with nodes of degree 3 and height 2 (beware the variables are called differently in the applet and in the rest of this lecture : their b denotes de degree and their d the depth).
- You can also enter evaluation for the nodes, try for example

$$1, 0, 1, 1, -1, 0, 0, -1, 0.$$

- Note that, there is no pruning mechanism implemented (e.g. when reaching Win).
- If you use the algorithm `Minimax alpha beta`, some pruning occurs.

With one function for each player

First call

ExploreMax(`currentState`)

ExploreMax

ExploreMin

Remark

- the two functions are calling each other recursively until a leaf of the game tree is reached
- the values are propagated upward

Further remarks

Warning

- the tree is never fully stored on the computer
- some parts of it are stored in RAM
- Essentially by the call stack of the two functions.

Implementation issues

- the max and min in these functions are implemented by loops of recursive calls
- to avoid pointless computing, if we find a maximal value in a max loop (or a minimal value in a min loop) we can stop the loop early.
- in an *object oriented framework*, the passing of state as arguments will probably not be needed

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 : $\mathcal{O}(h)$

Is chess easy ?

- There is an official limit at chess on h (due to special rules such as the *threefold repetition* or *the fifty-move rule*).
- At every stage, there is a finite number of possible moves ;so d is also finite.
- Consequently, the game tree of chess is finite ;
- we can explore the tree and decide whether white has a winning strategy.
- *Is this lecture over ?*

0.

Resource limit

- At chess, for reasonable games, we have $h = 80$ and $d = 30$.
- So the tree has around 10^{118} nodes.
- Note that the *number of atoms of the earth* is estimated at 10^{51} and the *number of atoms of the universe* is estimated at 10^{80}.
- Another indication of the difficulty of games : numerous related problems are Pspace-complete or even *ExpTime*-complete and are very unlikely or can not have polynomial time algorithms (= efficient algorithms).

Parameters of some well known games

Game	board size	States	Tree Nodes	Average length
Tic-Tac-Toe	9	10^3	10^5	9
Connect Four	42	10^{13}	10^{21}	36
Checkers	32	10^{20}	10^{31}	70
Chess	64	10^{47}	10^{123}	80
Go	361	10^{171}	10^{360}	150

Notes

- Taken from wikipedia : `http://en.wikipedia.org/wiki/Game_complexity`
- The values are mostly approximations so they may differ a bit from other sources but the order of magnitude is always the same.

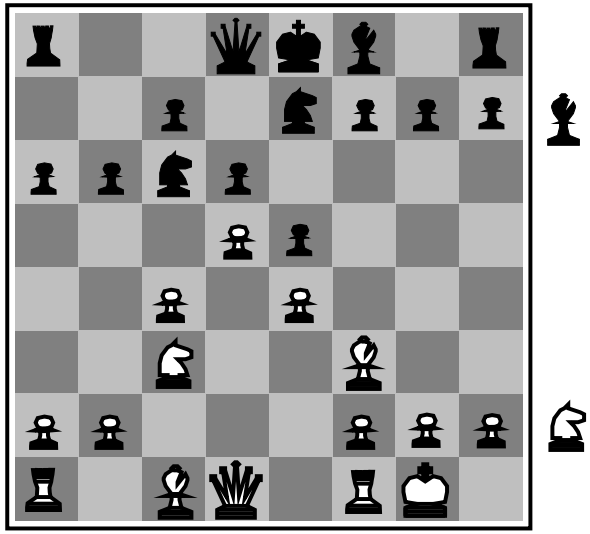
What does it mean?

- We can only use *brute force computing* with our minimax algorithm for small game trees, for example in the case of Tic-Tac-Toe.
- In some cases (Connect Four, Checkers), mixing minimax with various clever tricks, it is possible to search enough of the game tree to decide what happens (player one has a winning strategy for connect four and both have a non-losing strategy for Checkers).
- For most games, like Chess or Go, we just do not known who has a winning strategy but *we can use minimax together with some human knowledge about the game to design a program that can play the game like a human would* (at grand Master level or better for chess and at weak amateur level but in progress for Go).

In practice

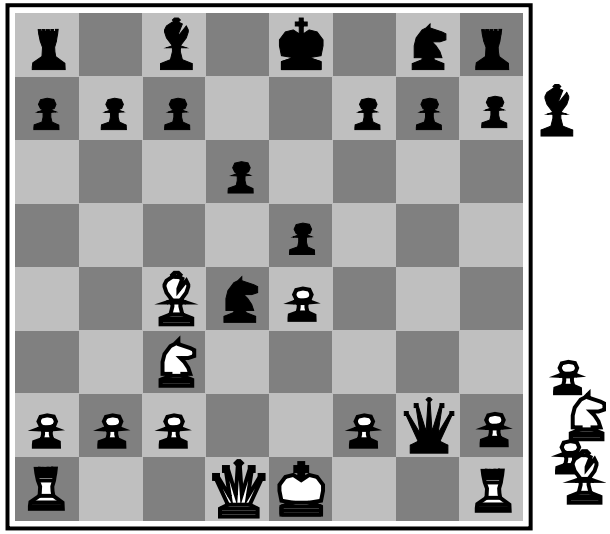
- We have about 100 seconds to play.
- Suppose that we can explore up to 10^4 nodes per second.
- We can check 10^6 nodes before answering.
- Standard approach :
  - put a *cutoff test e.g. depth limit d* (a bit naive) ; and,
  - use an *evaluation function f* to give an *artificial* payoff to positions that are not genuine endgame :*e.g.* Loss$= -\infty$, Draw$= 0$, Win$= +\infty$, other positions given a *heuristic value that reflects human knowledge about the game*.

Evaluation functions



Black to move

White slightly better



White to move

Black winning

- For chess, typically *linear* weighted sum of *features*,
  $\mathbf{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- e.g., $w_1 = 9$ and, $f_1(s) = \#\text{white horses} - \#\text{black horses}$.

Cutting off search

Minimax(currentState,remainingDepth)

```
1: if currentState is a terminal state then
2:    return payoff(currentState)
3: else if remainingDepth = 0 then
4:    return f(currentState)
5: else if I am to move then
6:    return max_nextState{Minimax(nextState, remainingDepth-
      1)}
7: else
8:    return min_nextState{Minimax(nextState, remainingDepth-
      1)}
9: end if
```

Cutting off search

Implementation Issues

- With two functions **ExploreMax** and **ExploreMin**, note that both will need a cut-off test
- If a cut-off is added, we also need to implement an evaluation function f.
- In a two player game, it is recommended to keep this function f symmetric.
- In this case, the simplest evaluation function is $+\infty$ for a win, $-\infty$ for a loss and 0 otherwise (essentially every game that has not been simulated until the end is treated as a draw by the AI in its analysis).

Advertising

Vladimir Nabokov “The Defence”.

He glanced at the chessboard and his brain wilted from hitherto unprecedented weariness. But the chessmen were pitiless, they held and absorbed him. There was horror in this, but in this also was the sole harmony, for what else exists in the world besides chess?