

Game Playing

A (pure) strategy:

- a complete set of advance instructions that specifies a definite choice for every conceivable situation in which the player may be required to act.

In a two-player game, a strategy allows the player to have a response to every move of the opponent.

Game-playing programs implement a strategy as a software mechanism that supplies the right move on request.

a complete set of advance instructions that specifies a definite choice for every conceivable situation in which the player may be required to act.

Game-playing programs implement a strategy as a software mechanism that supplies the right move on request.

Two-Person Perfect Information Deterministic Game

- Two players take turns making moves
- Call one Min and the other Max
- Deterministic moves: Board state fully known,
- One player wins by defeating the other (or else there is a tie)
- Want a strategy to win, assuming the other person plays rationally

- ## Two-Person Perfect Information Deterministic Game
- Two players take turns making moves
 - Call one Min and the other Max
 - Deterministic moves: Board state fully known,
 - One player wins by defeating the other (or else there is a tie)
 - Want a strategy to win, assuming the other person plays rationally

A logic-based approach to games

Find a winning strategy by *proving* that the game can be won -- use backward chaining.

A very simple game: **nim**.

- initially, there is one stack of chips;
- a move: select a stack and divide it in two unequal non-empty stacks;
- a player who cannot move loses the game.

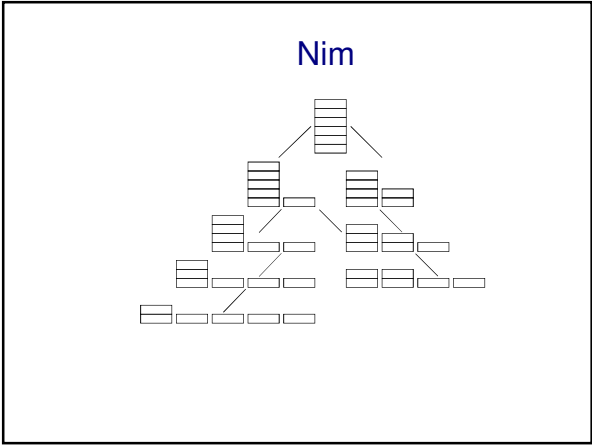
(The player who moves first can win.)

A very simple game: **nim**.

- (The player who moves first can win.)

Nim

```
graph TD; A[5] --> B[4]; A --> C[3]; B --> D[3]; B --> E[2]; C --> F[2]; C --> G[1]; D --> H[2]; D --> I[1]; E --> J[1]; E --> K[1]; F --> L[1]; F --> M[1]; G --> N[0]; H --> O[1]; H --> P[1]; I --> Q[0]; J --> R[0]; J --> S[0]; K --> T[0]; K --> U[0]; L --> V[0]; L --> W[0]; M --> X[0]; M --> Y[0]; N --> Z[0]; O --> AA[0]; O --> AB[0]; P --> AC[0]; Q --> AD[0]; R --> AE[0]; S --> AF[0]; T --> AG[0]; U --> AH[0]; V --> AI[0]; W --> AJ[0]; X --> AK[0]; Y --> AL[0]; Z --> AM[0]; AA --> AN[0]; AB --> AO[0]; AC --> AP[0]; AD --> AQ[0]; AE --> AR[0]; AF --> AS[0]; AG --> AT[0]; AH --> AU[0]; AI --> AV[0]; AJ --> AW[0]; AK --> AX[0]; AL --> AY[0]; AM --> AZ[0]; AN --> BA[0]; AO --> BB[0]; AP --> BC[0]; AQ --> BD[0]; AR --> BE[0]; AS --> BF[0]; AT --> BG[0]; AU --> BH[0]; AV --> BI[0]; AW --> BJ[0]; AX --> BK[0]; AY --> BL[0]; AZ --> BM[0]; BA --> BN[0]; BB --> BO[0]; BC --> BP[0]; BD --> BQ[0]; BE --> BR[0]; BF --> BS[0]; BG --> BT[0]; BH --> BU[0]; BI --> BV[0]; BJ --> BW[0]; BK --> BX[0]; BL --> BY[0]; BM --> BZ[0]; BN --> CA[0]; BO --> CB[0]; BP --> CC[0]; BQ --> CD[0]; BR --> CE[0]; BS --> CF[0]; BT --> CG[0]; BU --> CH[0]; BV --> CI[0]; BW --> CJ[0]; BX --> CK[0]; BY --> CL[0]; BZ --> CM[0]; CA --> CN[0]; CB --> CO[0]; CC --> CP[0]; CD --> CQ[0]; CE --> CR[0]; CF --> CS[0]; CG --> CT[0]; CH --> CU[0]; CI --> CV[0]; CJ --> CW[0]; CK --> CX[0]; CL --> CY[0]; CM --> CZ[0]; CN --> DA[0]; CO --> DB[0]; CP --> DC[0]; CQ --> DD[0]; CR --> DE[0]; CS --> DF[0]; CT --> DG[0]; CU --> DH[0]; CV --> DI[0]; CW --> DJ[0]; CX --> DK[0]; CY --> DL[0]; CZ --> DM[0]; DA --> DN[0]; DB --> DO[0]; DC --> DP[0]; DD --> DQ[0]; DE --> DR[0]; DF --> DS[0]; DG --> DT[0]; DH --> DU[0]; DI --> DV[0]; DJ --> DW[0]; DK --> DX[0]; DL --> DY[0]; DM --> DZ[0]; DN --> EA[0]; DO --> EB[0]; DP --> EC[0]; DQ --> ED[0]; DR --> EE[0]; DS --> EF[0]; DT --> EG[0]; DU --> EH[0]; DV --> EI[0]; DW --> EJ[0]; DX --> EK[0]; DY --> EL[0]; DZ --> EM[0]; EA --> EN[0]; EB --> EO[0]; EC --> EP[0]; ED --> EQ[0]; EE --> ER[0]; EF --> ES[0]; EG --> ET[0]; EH --> EU[0]; EI --> EV[0]; EJ --> EW[0]; EK --> EX[0]; EL --> EY[0]; EM --> EZ[0]; EN --> FA[0]; EO --> FB[0]; EP --> FC[0]; EQ --> FD[0]; ER --> FE[0]; ES --> FF[0]; ET --> FG[0]; EU --> FH[0]; EV --> FI[0]; EW --> FJ[0]; EX --> FK[0]; EY --> FL[0]; EZ --> FM[0]; FA --> FN[0]; FB --> FO[0]; FC --> FP[0]; FD --> FQ[0]; FE --> FR[0]; FF --> FS[0]; FG --> FT[0]; FH --> FU[0]; FI --> FV[0]; FJ --> FW[0]; FK --> FX[0]; FL --> FY[0]; FM --> FZ[0]; FN --> GA[0]; FO --> GB[0]; FP --> GC[0]; FQ --> GD[0]; FR --> GE[0]; FS --> GF[0]; FT --> GG[0]; FU --> GH[0]; FV --> GI[0]; FW --> GJ[0]; FX --> GK[0]; FY --> GL[0]; FZ --> GM[0]; GA --> GN[0]; GB --> GO[0]; GC --> GP[0]; GD --> GQ[0]; GE --> GR[0]; GF --> GS[0]; GG --> GT[0]; GH --> GU[0]; GI --> GV[0]; GJ --> GW[0]; GK --> GX[0]; GL --> GY[0]; GM --> GZ[0]; GN --> HA[0]; GO --> HB[0]; GP --> HC[0]; GQ --> HD[0]; GR --> HE[0]; GS --> HF[0]; GT --> HG[0]; GU --> HH[0]; GV --> HI[0]; GW --> HJ[0]; GX --> HK[0]; GY --> HL[0]; GZ --> HM[0]; HA --> HN[0]; HB --> HO[0]; HC --> HP[0]; HD --> HQ[0]; HE --> HR[0]; HF --> HS[0]; HG --> HT[0]; HH --> HU[0]; HI --> HV[0]; HJ --> HW[0]; HK --> HX[0]; HL --> HY[0]; HM --> HZ[0]; HN --> IA[0]; HO --> IB[0]; HP --> IC[0]; HQ --> ID[0]; HR --> IE[0]; HS --> IF[0]; HT --> IG[0]; HU --> IH[0]; HV --> II[0]; HW --> IJ[0]; HX --> IK[0]; HY --> IL[0]; HZ --> IM[0]; IA --> IN[0]; IB --> IO[0]; IC --> IP[0]; ID --> IQ[0]; IE --> IR[0]; IF --> IS[0]; IG --> IT[0]; IH --> IU[0]; II --> IV[0]; IJ --> IW[0]; IK --> IX[0]; IL --> IY[0]; IM --> IZ[0]; IN --> JA[0]; IO --> JB[0]; IP --> JC[0]; IQ --> JD[0]; IR --> JE[0]; IS --> JF[0]; IT --> JG[0]; IU --> JH[0]; IV --> JI[0]; IW --> JJ[0]; IX --> JK[0]; IY --> JL[0]; IZ --> JM[0]; JA --> JN[0]; JB --> JO[0]; JC --> JP[0]; JD --> JQ[0]; JE --> JR[0]; JF --> JS[0]; JG --> JT[0]; JH --> JU[0]; JI --> JV[0]; JJ --> JW[0]; JK --> JX[0]; JL --> JY[0]; JM --> JZ[0]; JN --> KA[0]; JO --> KB[0]; JP --> KC[0]; JQ --> KD[0]; JR --> KE[0]; JS --> KF[0]; JT --> KG[0]; JU --> KH[0]; JV --> KI[0]; JW --> KJ[0]; JX --> KK[0]; JY --> KL[0]; JZ --> KM[0]; KA --> KN[0]; KB --> KO[0]; KC --> KP[0]; KD --> KQ[0]; KE --> KR[0]; KF --> KS[0]; KG --> KT[0]; KH --> KU[0]; KI --> KV[0]; KJ --> KW[0]; KK --> KX[0]; KL --> KY[0]; KM --> KZ[0]; KN --> LA[0]; KO --> LB[0]; KP --> LC[0]; KQ --> LD[0]; KR --> LE[0]; KS --> LF[0]; KT --> LG[0]; KU --> LH[0]; KV --> LI[0]; KW --> LJ[0]; KX --> LK[0]; KY --> LL[0]; KZ --> LM[0]; LA --> LN[0]; LB --> LO[0]; LC --> LP[0]; LD --> LQ[0]; LE --> LR[0]; LF --> LS[0]; LG --> LT[0]; LH --> LU[0]; LI --> LV[0]; LJ --> LW[0]; LK --> LX[0]; LL --> LY[0]; LM --> LZ[0]; LN --> MA[0]; LO --> MB[0]; LP --> MC[0]; LQ --> MD[0]; LR --> ME[0]; LS --> MF[0]; LT --> MG[0]; LU --> MH[0]; LV --> MI[0]; LW --> MJ[0]; LX --> MK[0]; LY --> ML[0]; LZ --> MN[0]; MA --> MO[0]; MB --> MP[0]; MC --> MQ[0]; MD --> MR[0]; ME --> MS[0]; MF --> MT[0]; MH --> MU[0]; MI --> MV[0]; MJ --> MW[0]; MK --> MX[0]; ML --> MY[0]; MN --> MZ[0]; MO --> NA[0]; MP --> NB[0]; MQ --> NC[0]; MR --> ND[0]; MS --> NE[0]; MT --> NF[0]; MU --> NG[0]; MV --> NH[0]; MW --> NI[0]; MX --> NJ[0]; MY --> NK[0]; MZ --> NM[0]; NA --> NO[0]; NB --> NP[0]; NC --> NQ[0]; ND --> NR[0]; NE --> NS[0]; NF --> NT[0]; NH --> NU[0]; NI --> NV[0]; NJ --> NW[0]; NK --> NX[0]; NM --> NY[0]; NO --> NZ[0]; NP --> OA[0]; NQ --> OB[0]; NR --> OC[0]; NS --> OD[0]; NT --> OE[0]; NU --> OF[0]; NV --> OG[0]; NW --> OH[0]; NX --> OI[0]; NY --> OJ[0]; NZ --> OM[0]; OA --> ON[0]; OB --> OP[0]; OC --> OQ[0]; OD --> OR[0]; OE --> OS[0]; OF --> OT[0]; OH --> OU[0]; OI --> OV[0]; OJ --> OW[0]; OM --> OX[0]; ON --> OY[0]; OP --> OZ[0]; OQ --> PA[0]; OR --> PB[0]; OS --> PC[0]; OT --> PD[0]; OU --> PE[0]; OV --> PF[0]; OW --> PG[0]; OX --> PH[0]; OY --> PI[0]; OZ --> PM[0]; PA --> PN[0]; PB --> PO[0]; PC --> PP[0]; PD --> PQ[0]; PE --> PR[0]; PF --> PS[0]; PG --> PT[0]; PH --> PU[0]; PI --> PV[0]; PM --> PW[0];
```



Static evaluation

A **static evaluation function** returns the value of a move without trying to play (which would mean simulating the rest of the game but not playing it).

“Usually” a static evaluation function returns positive values for positions advantageous to **Player 1**, negative values for positions advantageous to **Player 2**.

If **Player 1** is rational, he will choose the maximal value of a leaf.

Then, **Player 2** will choose the minimal value.

“Usually” a static evaluation function returns positive values for positions advantageous to **Player 1**, negative values for positions advantageous to **Player 2**.

If **Player 1** is rational, he will choose the maximal value of a leaf.
Then, **Player 2** will choose the minimal value.

Static evaluation

If we can have (guess or calculate) the value of an internal node **N**, we can treat it as if it were a leaf. This is the basis of the **minimax** procedure.

No tree would be necessary if we could evaluate the initial position **statically**. Normally we need a tree, and we need to look-ahead into it. Further positions can be evaluated more precisely, because there is more information, and a more focussed search.

Static evaluation

If we can have (guess or calculate) the value of an internal node **N**, we can treat it as if it were a leaf. This is the basis of the **minimax** procedure.

No tree would be necessary if we could evaluate the initial position **statically**. Normally we need a tree, and we need to look-ahead into it. Further positions can be evaluated more precisely, because there is more information, and a more focussed search.

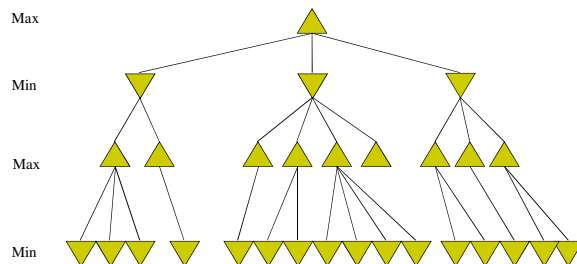
Minimax Tree

- Create a **utility function**
 - Evaluation of board/game state to determine how strong the position of each player.
 - Player 1 wants to **maximize** the utility function
 - Player 2 wants to **minimize** the utility function
- Minimax tree
 - Generate a new level for each move
 - Levels alternate between “max” (player 1 moves) and “min” (player 2 moves)

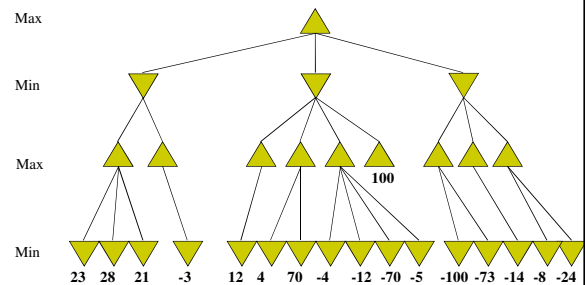
Minimax Tree Evaluation

- Assign utility values to leaves
 - If leaf is a “final” state, assign the maximum or minimum possible utility value (depending on who would win)
 - If leaf is not a “final” state, must use some other heuristic, specific to the game, to evaluate how good/bad the state is at that point

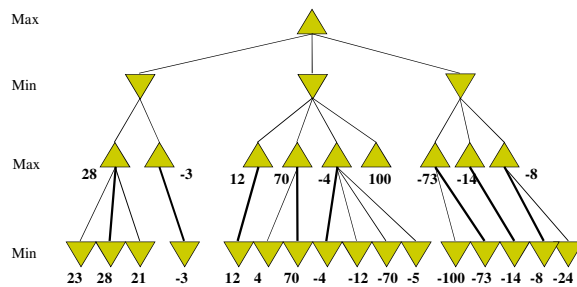
Minimax tree



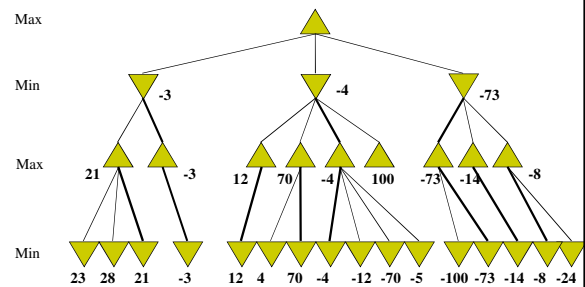
Minimax tree



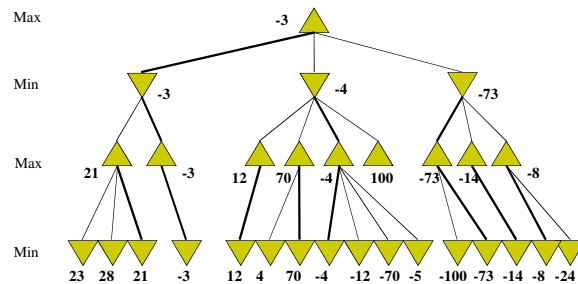
Minimax tree



Minimax tree



Minimax tree



Tic-Tac-Toe

Let player **A** be **x** and let $\text{open}(x)$, $\text{open}(o)$ mean the number of lines open to **x** and **o**. There are 8 lines. An evaluation function for position **P**:

$f(P) = -\infty$ if **o** wins

$f(P) = +\infty$ if **x** wins, otherwise

$f(P) = \text{open}(x) - \text{open}(o)$

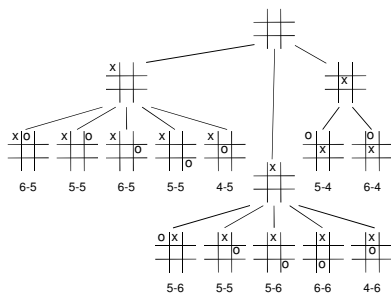
Example:

$\text{open}(x) - \text{open}(o) = 4 - 6$

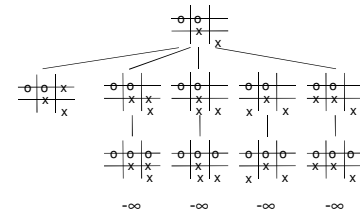
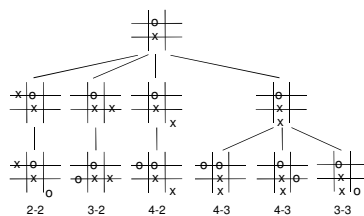
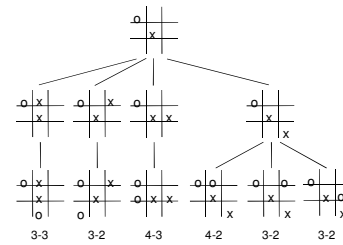


Assumptions:

only one of symmetrical positions is generated;



Player **B** chooses the minimal backed-up value among level 1 nodes.
Player **A** chooses the maximal value, and makes the move.
Player **B**, as a rational agent, selects the optimal response.



Building complete piles is usually not necessary. If we evaluate a position when it is generated, we may save a lot.

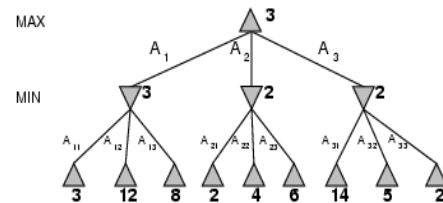
Assume that we are at a minimizing level. If the evaluation function returns $-\infty$, we do not need to consider other positions:
 $-\infty$ will be the minimum.

The same applies to $+\infty$ at a maximizing level.

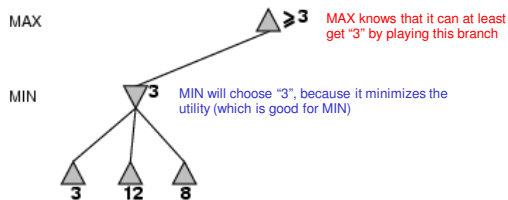
Pruning the Minimax Tree

- Minimax works best for large trees, but it can be useful even in mini-games such as tic-tac-toe.
- Since we have limited time available, we want to avoid unnecessary computation in the minimax tree.
- **Pruning**: ways of determining that certain branches will not be useful. Then cut off these branches

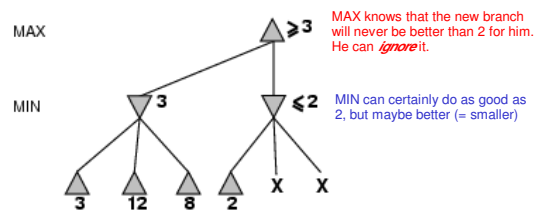
pruning



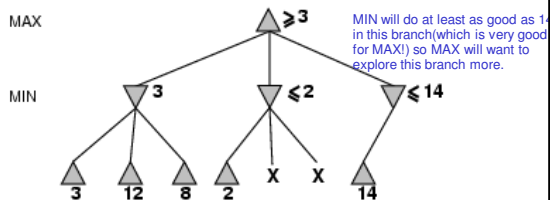
α pruning



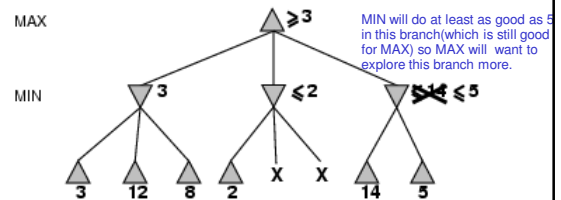
α pruning



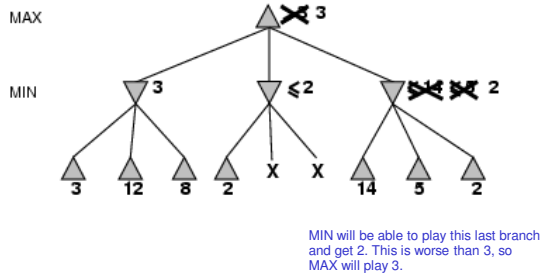
α pruning



α pruning



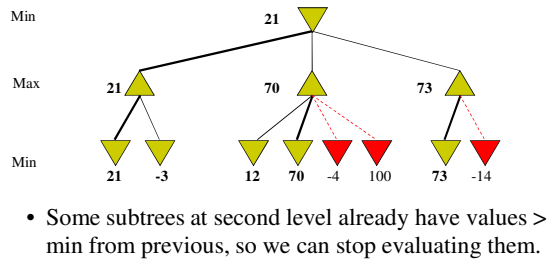
α pruning



β pruning

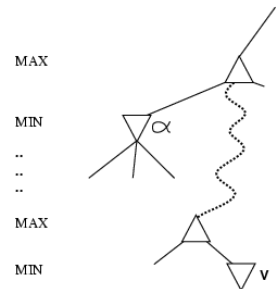
- Similar idea to α pruning, but the other way around
- If the current minimum is less than the successor's max value, don't look down that max tree any more

β pruning example



Why is it called α - β ?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If v is worse than α , *max* will avoid v
- prune that branch
- Define β similarly for *min*



α - β Pruning properties

- Pruning by these cuts does not affect final result
 - May allow you to go much deeper in tree
- Properties:
 - Evaluating "best" branch first yields better likelihood of pruning later branches
 - Perfect ordering reduces time to $O(b^{m/2})$

Properties of minimax

- **Complete?** Yes (if tree is finite)
- **Optimal?** Yes (against an rational opponent)
- **Time complexity?** $O(b^m)$
- **Space complexity?** $O(bm)$ (depth-first exploration)
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
 - exact solution completely infeasible