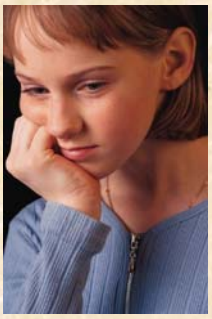
知能情報処理　探索(4)

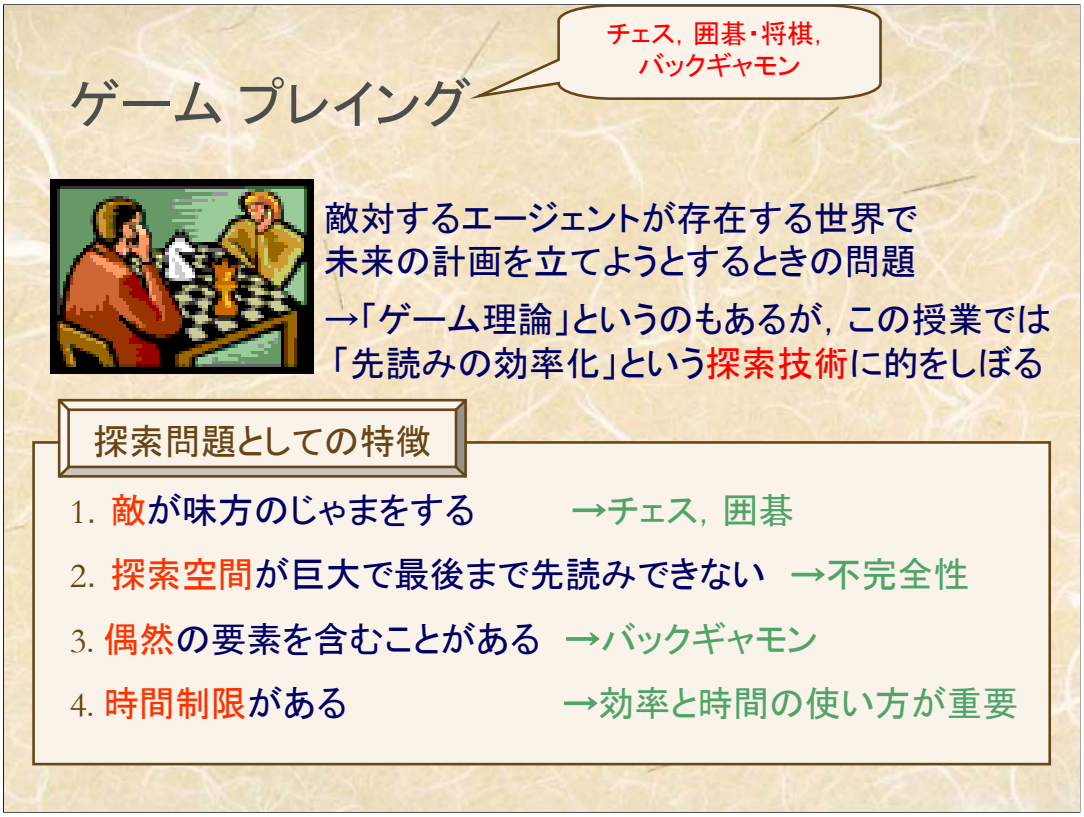
先を読んで知的な行動を選択するエージェント

ゲームプレイング

(Game Playing)

- ゲーム木と評価関数
- ミニマックス法
- アルファベータ法
- ゲームプログラムの現在





今回の授業では，コンピュータにゲームをプレイさせる技術を学ぶ．ただし，ここでとりあげるゲームは，RPGやシューティングのような現代的なコンピュータゲームではなく，チェスや囲碁・将棋あるいはバックギャモン（西洋すごろく）のように，２人のプレイヤーが交互に指し手を進めて勝負を競うものである．このようなゲームをうまく指すことは，伝統的に「知能的」な行動と考えられており，人工知能技術の初期のころから精力的に研究されてきたのである．

これは「遊び」の研究とも言えるが，少しまじめな人向きに，一般化して述べると，「**敵対するエージェント**が存在する世界で，未来の計画を立てようとするときの問題」といえる．実際，経済学や社会学では，企業や人間の行動を行列ゲームと呼ばれる数学的に単純化されたゲームとしてモデル化した「**ゲーム理論**」に興味が持たれており，優れた研究者はノーベル経済学賞を受けている．人工知能の分野でも，敵対あるいは協力しあう複数のエージェントを扱うための基礎理論として，ゲーム理論を参考にすることが多い．

しかし，今回の授業では，そのような数学的ゲームの理論解析ではなく，実際のゲームにおいて，いかに効率よく何手も先を読むかというような探索技術に的をしぼる．この探索技術は，これまで学んできた一般的な探索問題に比べて，つぎの4つの点で特徴的である．

1. 敵が味方のじゃまをする．
2. 探索空間が巨大で最後まで先読みできない．
3. 偶然の要素を含むことがある．
4. 時間制限がある．

この授業では，上記の1と2について考慮したアルゴリズムについて学ぶ．また，3については，そのアルゴリズムを自然に拡張することによって対応できることを理解する．4はこの授業の範囲外である．

二人ゲームの形式化

- 初期状態(initial state)
  - 盤面の状態，どちらの手番か
- オペレータ(operator)の集合
  - プレイヤが指すことのできる合法手
  - その手を指したら，盤面の状態と手番はどうなるか
- 終端テスト(terminal test)
  - ゲームの終了の決定
- 効用関数(utility function)
  - ゲームの結果を数値として与える．
  - 勝ち(+1)，負け(-1)，引分け(0)

この授業で考察の対象とするのは，オセロやチェスなどを包含するように抽象化された二人ゲームと呼ばれるクラス(種類)である．このゲームにおける指し手の決定は，**初期状態**，**オペレータ**，**終端テスト**，**効用関数**の4種類の情報から規定される探索問題として扱うことができる．それぞれの意味はこのスライドに書いてあるとおりである．

特徴的なのは，ふつうの探索問題の経路コストのかわりに，**終端状態**でのゲームの勝ち負けを評価する**効用関数**を導入することである．ここでは，勝ちを＋1，負けを－1などとしたが，同じ勝ちでも，勝ち方の程度によって，この値を変えても良い．たとえば，大勝ちなら＋2，僅少差の勝ちなら＋1などである．さしづめ，コンピュータゲームなら，ゲットできるHPとかコインの数といったところである．

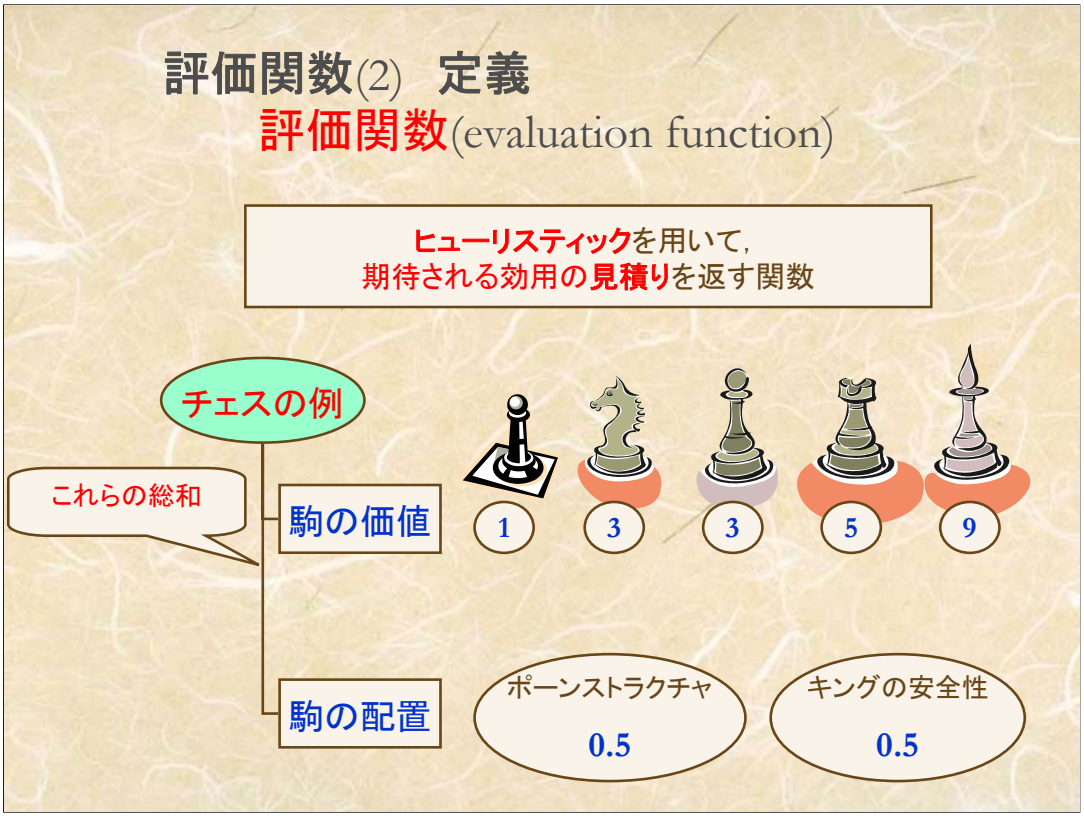
探索アルゴリズムの中心にある考え方が探索木の生成であったのと同様，ゲームプレイングでは**ゲーム木**と呼ばれる一種の探索木を生成する．

このスライドは，三目並べ（自分のコマを縦・横・斜めのいずれか一列に先に並べたら勝ち）を2人のプレイヤーMAX（×）とMIN（○）が対戦するときのゲーム木である．先手であるMAXはゲーム木の終端状態の効用関数の値を（その名とおり）**最大化**（＝1）しようとプレイし，一方，後手のMINはそれを**最小化**（＝－1）しようとする．

したがって，ここでいう効用とはMAXにとってのものである．MINの効用は，MAXの効用の符号（正負）を反転させたものと考え，あえて表記をしていない．

三目並べでは，ゲーム木を完全に作り出すことは簡単である．しかし，現実のゲームでは，それは困難である．ふつうは，終端状態までのすべての道筋を探索する時間はない．なぜなら，ふつう，そのような道筋の数は，探索木の深さに関して指数関数的に急激に増加するからである．

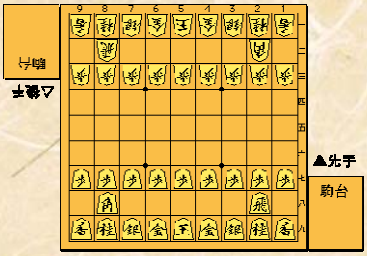
そこで，終端状態で効用関数を適用して勝敗を判断することはあきらめ，終端でなくても先読みを打ち切って，勝勢か劣勢かを判定することにする．そのため，終端テストのかわりに，**打ち切りテスト**を導入して打ち切り状態かどうかを判定する．打ち切り状態であるならば，**評価関数**を導入して，打ち切り状態での勝勢の度合いを数量的に表現する．



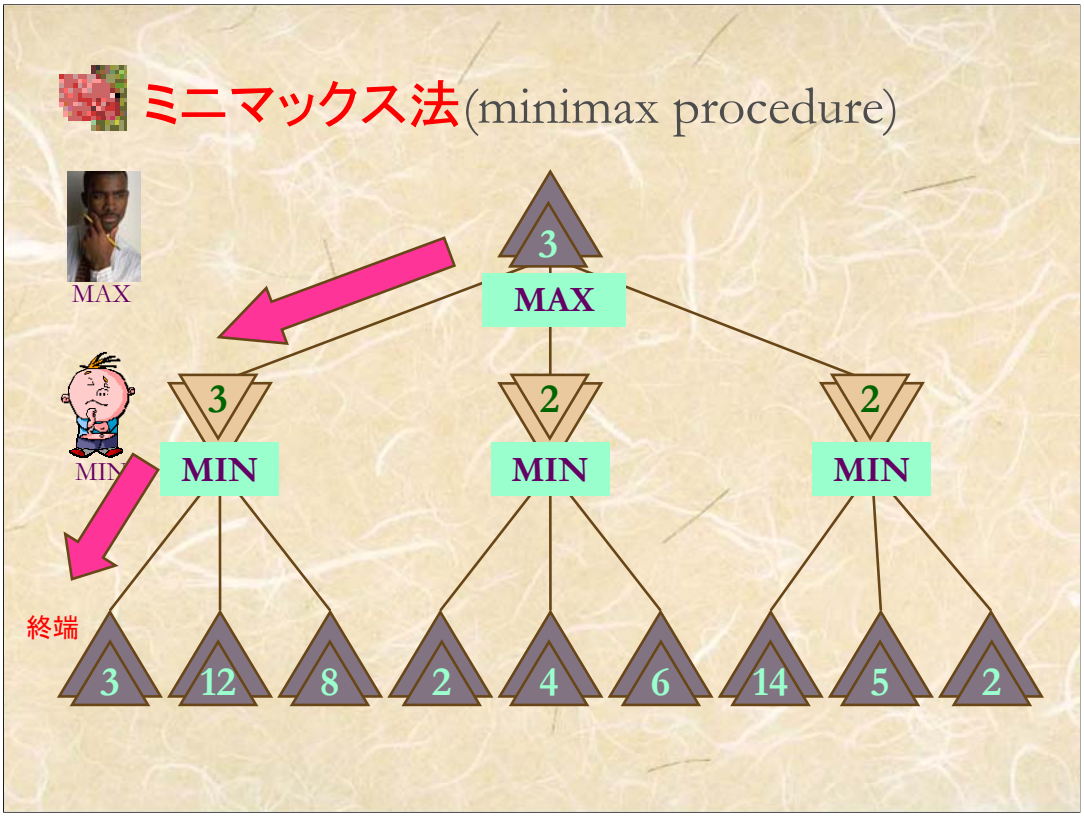
前のスライドで述べた評価関数は，ヒューリスティックを用いて，期待される効用の見積もりを返す関数といえる．たとえば，チェスの場合は，駒の価値や配置を経験に基づいて数値化し，打ち切り状態におけるそれらの数値の総和として評価関数を定義する．

評価関数(3)　参考：将棋の駒の価値（谷川浩司）

飛車	15	竜王	17
角行	13	龍馬	15
金将	9		
銀将	8	成銀	9
桂馬	6	成桂	10
香車	5	成香	10
歩兵	1	と金	12



単なる参考だが，将棋の場合の駒の価値は，このスライドのような感じである．



では，具体的に探索アルゴリズムの概要を見ていこう．

最も基本となるのは，**ミニマックス法**と呼ばれるものである．

この方法は，ゲーム木を終端まで生成し，終端状態の効用を求める．（実際には，ゲーム木を打ち切り状態まで生成し，評価関数で評価値を求めるのでもよいが，簡単のため，終端まで生成するとして説明する．）

ミニマックス法は，それらの効用の最大値または最小値を親ノードに付し，それを上へ上へと伝播させながら，最終的に，初期状態ノードに値を付すまで伝播させる．各ノードに付された値は，その状態以後，両プレイヤーが最適な指し手を選んだときに，両者に保証される効用を表している．

具体的には，MAXのノードには，その子ノードたちの効用の最大値を付す．その後，実際にゲームをするときに MAXがその効用を得るには，現在のノードにおいて，最大値をもつような子ノードの指し手を選べばよい．一方，MINのノードには，その子ノードの効用の最小値を付す．MINがその効用を得るには，最小値をもつ子ノードの指し手を選べばよい．

ゲームプレイングのアルゴリズムを設計するにあたり，前提として，ゲームの盤面の情報が1つの適切なオブジェクトとして表現されているとする．その情報にはその盤面で次はどちらのプレイヤの手番なのかが含まれている．また，指し手(オペレータ)の各々も適切なデータで表現されており，その数は有限個で，そのすべてが全オペレータという変数に保持されているとする．さらに，盤面にオペレータを適用することによって次の盤面を生成する手続きがすでに用意されているとしよう．

そういう前提でミニマックス法のアルゴリズムを記述したものがスライドの内容になっている．このアルゴリズムは，MAXの立場から書かれていて，先手のMAXが，

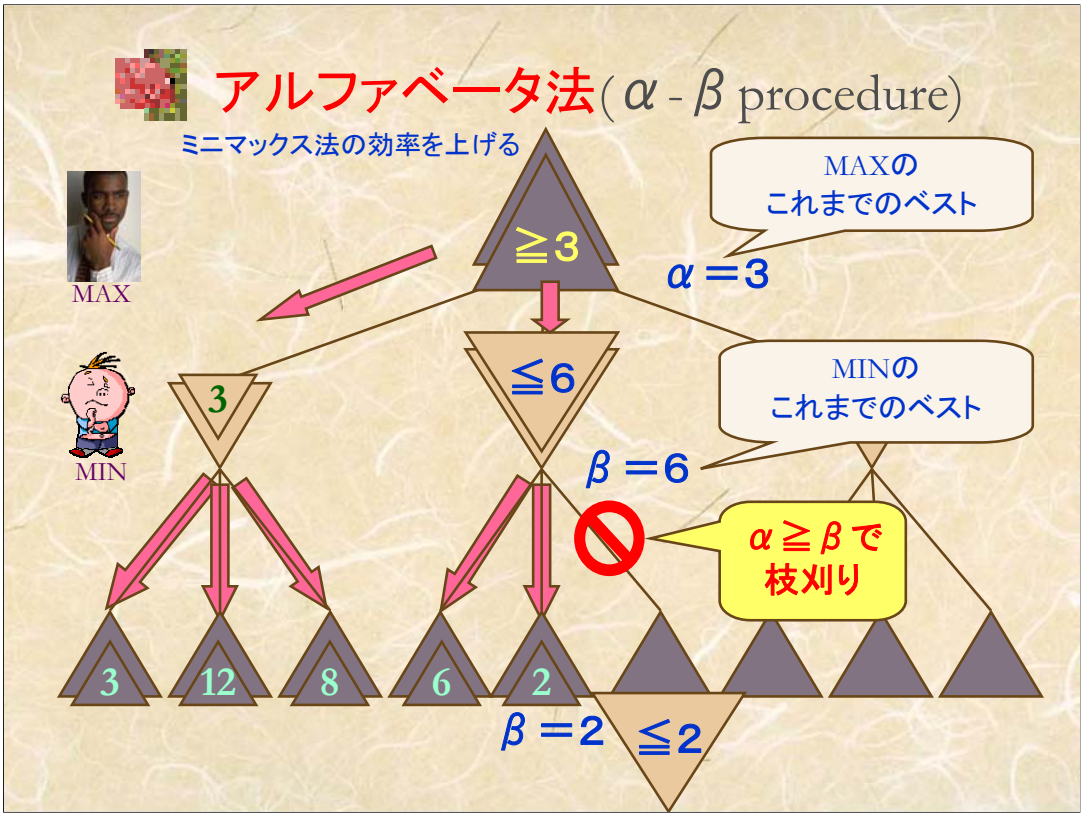
ミニマックス法(初期盤面);

という関数手続きの呼び出しをすることから起動される．その戻り値は，MAXが指すべき指し手(オペレータ)である．この手続きは，初期盤面を受け取ると，適用可能なすべてのオペレータをそれに適用して次の盤面を仮に作成し，ミニマックス値(次の盤面)の計算によってその盤面の評価値を求める．そして，それらのうち最大の評価値をもつオペレータを選んで，MAXが指すべきものとして返す．

このアルゴリズムで最も重要な役割を果たすのが，ミニマックス値(盤面)という関数手続きである．その役割は，与えられた盤面の効用を求めることである．盤面を受け取ると，この関数は，まずそれが終端状態かどうか判定する．もし終端状態ならば，効用関数を適用した結果を返して終了する．

終端状態でないときには，ミニマックス法(盤面)のアルゴリズムとまったく同様の考え方で，この盤面に各オペレータを適用したときの次の盤面の評価値を，ミニマックス値(盤面)の再帰的手続きを利用して求める．その後，この盤面での手番がMAXかMINかに応じて，それぞれ，評価値の最大値または最小値を返す．

ミニマックス値(盤面)は，自分自身を再帰的に呼び出すことによって，より深くまで先読みをしたときの盤面の効用を計算していることを確認してほしい．



アルファベータ法は，ゲームプレイングのアルゴリズムとして最も有名なものである．これはミニマックス法を改良して，ゲーム木のノードのうち，それより深く先読みする必要のない部分をうまく検出し，そこから先の枝をカット(**枝刈り**)することによって，探索木を小さくし，探索効率を上げるものである．

このスライドはアニメーションによって，その様子を説明している．

まず，ミニマックス法と同じように最も左の部分木を探索することによって，MINの左端のノードの評価値が min(3,12,8)=3となる．

この時点で，MAXの初期状態ノードの評価値は，確実に3以上になることがわかる．なぜなら，最悪の場合でも，MAXが初手としてこの最左の手を選べば，MINが最善手を選んで評価値3のノードに至るからである．MINが間違って最善手以外の指し手を選べば，評価値はそれ以上(12か8)となる．

つぎに，初期状態から真下(真ん中)の部分木を探索し，その先端で評価値6のノードを見つけたとする．このとき，その親ノードであるMINの手番を表すノードの評価値は確実に6以下となる．なぜなら，MINは今後6より小さい子ノードを見つけたらそれを選び，最悪の場合でもいま見つけた評価値6のノードを選べばよいからである．

アルファベータ法は，このような情報を**α**と**β**という変数で管理する．αはMAXのこれまでのベストの評価値で，MAXはこれをさらに大きくしようとする．βはMINのこれまでのベストの評価値で，MINはそれをさらに小さくしようとする．いま見ている例では，現在，$\alpha = 3$，$\beta = 6$となっている．，この例のように，ふつうは$\alpha < \beta$となっていて，αとβの間に両プレイヤにとっての最適解が存在する．

さて，例題に戻り，つぎに，この部分木でMINが評価値2の子ノードを見つけたとしよう．これでβが改善され，6から2に減少する．その結果，ここで$\alpha \geqq \beta$となったことが非常に重要な意味をもってくる．すでに述べたように，ふつうは$\alpha < \beta$なのだが，$3 = \alpha \geqq \beta = 2$はその反対である．

この時点で，MINは最悪でも評価値2の子ノードに進む手を指せばよいことがわかる．これは，$\alpha = 3$以上の子ノードを探しているMAXにとっては，おもしろくない．MAXが初手に真ん中の手を指せば，MAXの得る評価値は高々2にすぎない．それくらいなら，すでに見てきたように，左端の手を選んで評価値3を確保した方がよい．つまり，MAXにとって，いま見ていたMINのノードはこれ以上考察する必要のない価値のないものとなるのである．

アルファベータ法は，ここで**枝刈り**(pruning)と称して，それより先の探索枝をすべてカットし，MAXの初期状態からの第3の指し手の考察に進む．この枝刈りが大きな効果を発揮し，ふつう，探索効率は飛躍的に向上する．

アルファベータ法のアルゴリズム(1)

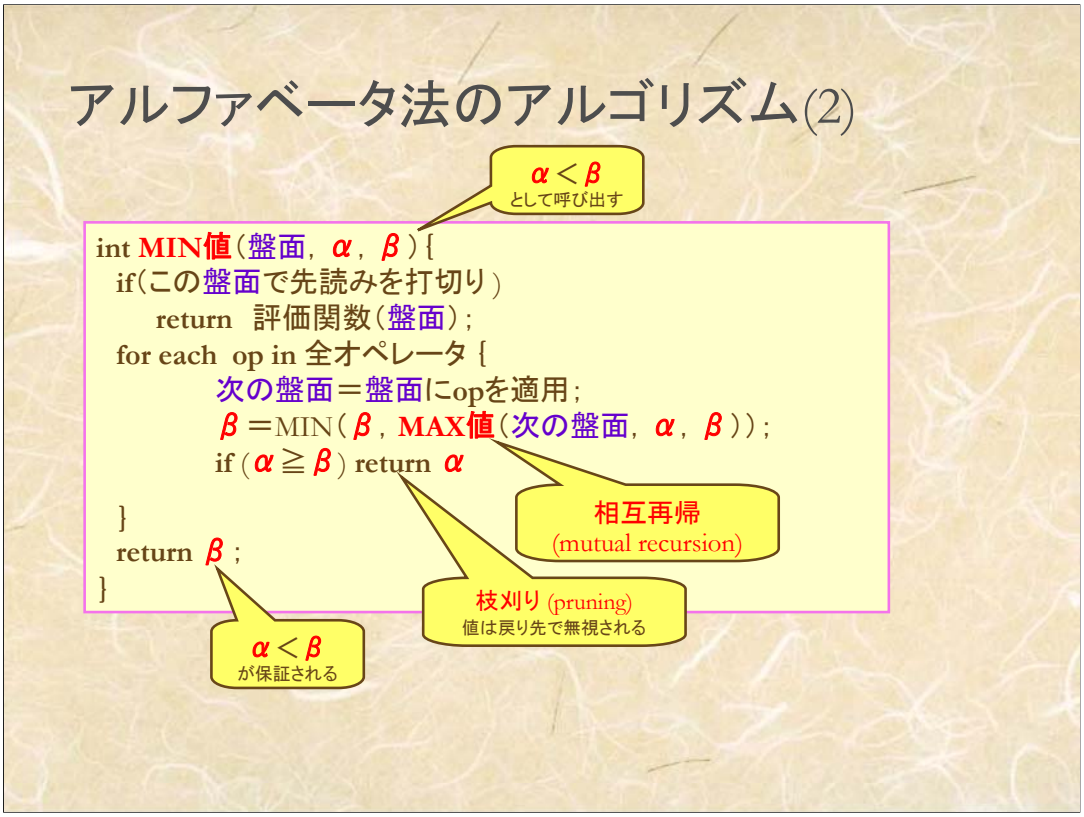
```
Operator アルファベータ法(盤面){
  α＝－∞;  β＝＋∞;
  for each op in 全オペレータ {
        次の盤面＝盤面にopを適用;
        α＝MAX(α, MIN値(次の盤面, α, β));
  }
  return αを最大にしたオペレータ op;
}
```

これがMAXが呼び出すアルファベータ法のメイン関数である.

最初，α を十分に小さい数，β を十分に大きい数とし，すべてのオペレータ対して次の盤面を生成し，

MIN値(次の盤面，α，β)

という関数によって，次の盤面でMINがベストの手を指したときの評価値を求める．それが現在の α の値より大きいときには，そのオペレータがこれまで最善だったオペレータよりも良いということを意味するので，α の値をその値に更新することによって， α が常に最大値であることを維持する.



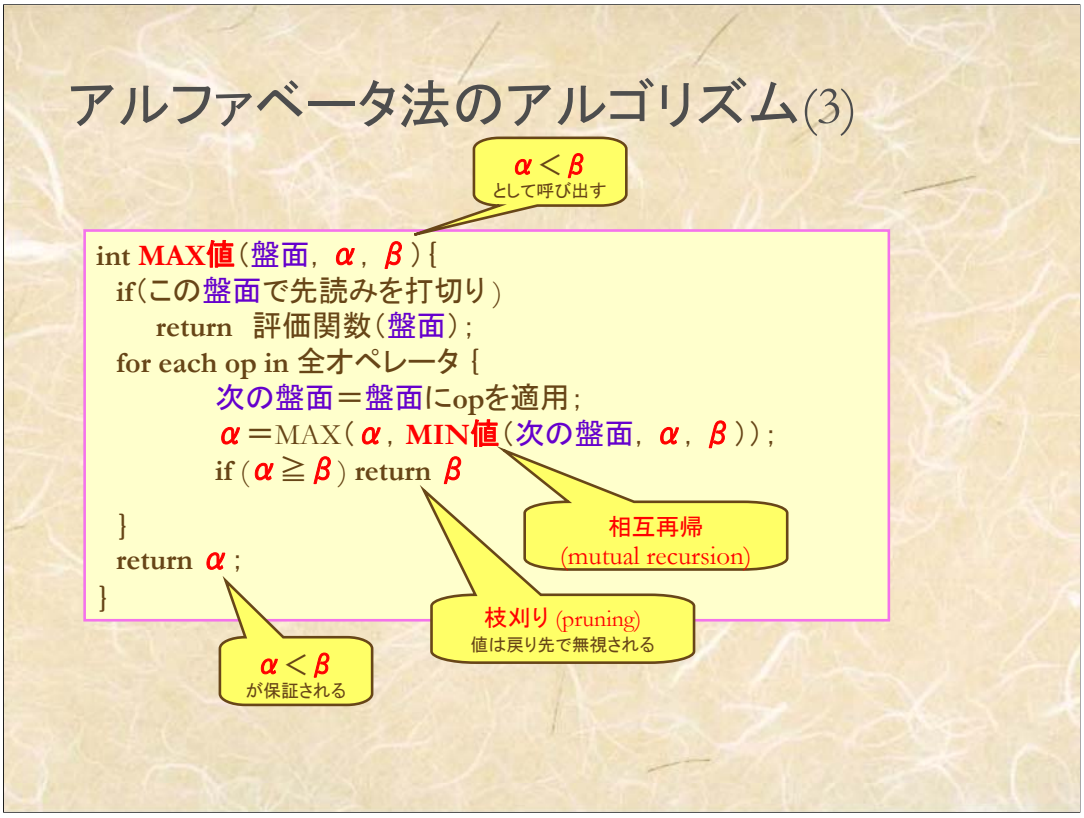
関数 MIN値(盤面，α，β)は，プレイヤーMINにとってのベストな手を探索する手続きである．この関数は，まず，この盤面で先読みを打ち切るかどうかを判断し，打ち切るときには，この盤面を評価関数で評価した値を返す．

先読みを打ち切らないときは，さらに先読みをするために，すべてのオペレータに対して次の盤面を生成し，

MAX値(次の盤面，α，β)

という関数によって，次の盤面でMAXがベストの手を指したときの評価値を求める．MINはそれらすべての評価値のうち，最小のものを返す．それが現在の β の値より小さいときには，そのオペレータがMINにとってこれまで最善だったオペレータよりも良いということを意味するので，β の値をその値に更新することによって， β が常に最小値であることを維持する．

その直後に枝刈りの判定が挿入される．もし， α ≧ β ならば，それ以上の探索をやめ，ただちに return する．このとき，戻り値は，戻り先のMAXの指し手の探索にとって意味のない十分小さな値なら何でもよい．MAXは現在の α より大きい値を探しているので，それを越えない値なら何でもよいのだが，このアルゴリズムでは α を戻すことにしている．



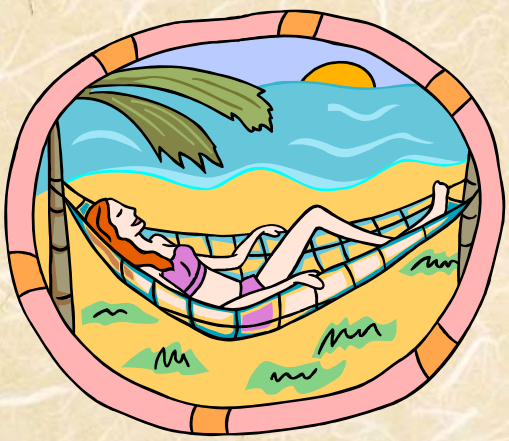
関数 MAX値(盤面，α，β)は，プレイヤーMAXにとってのベストな手を探索する手続きで，その内容は MIN値とほとんど同じである．ここからさらに先読みをする場合は，すべての指し手に対して次の盤面を生成し，

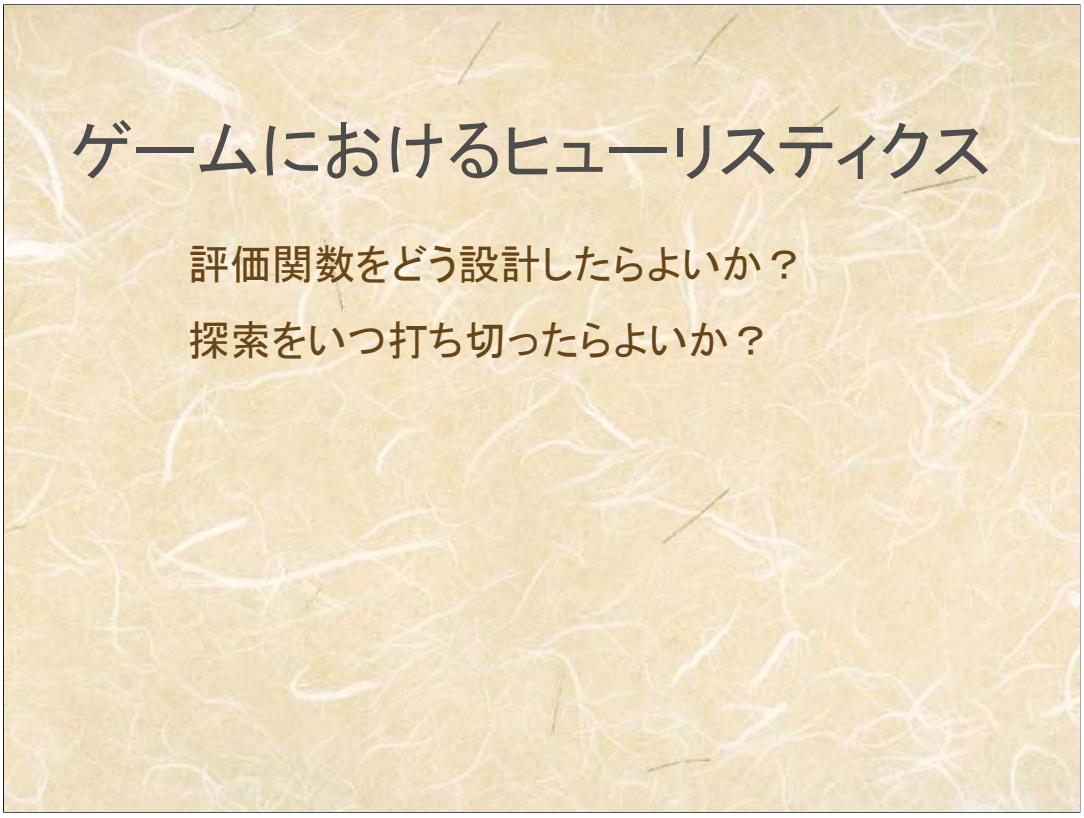
MIN値(次の盤面，α，β)

の計算によって，次の盤面でMINがベストの手を指したときの評価値を求める．

MAXはそれらすべての評価値のうちの最大値を常にαに維持するとともに，α≧βのときには枝刈りをする体制をとっている．

休憩





ここまでで基本的なアルゴリズムの概略がわかったが，実際に強いコンピュータプレイヤーを作るには，評価関数をどう作るか，また，探索をいつ打ち切ったらよいかという問題をクリアしなければならない．これは個別のゲーム毎にうまいヒューリスティクスを何とか見つけなければならないということになるのだが，おおまかな指針は幾つかあるので，それを見ておこう．

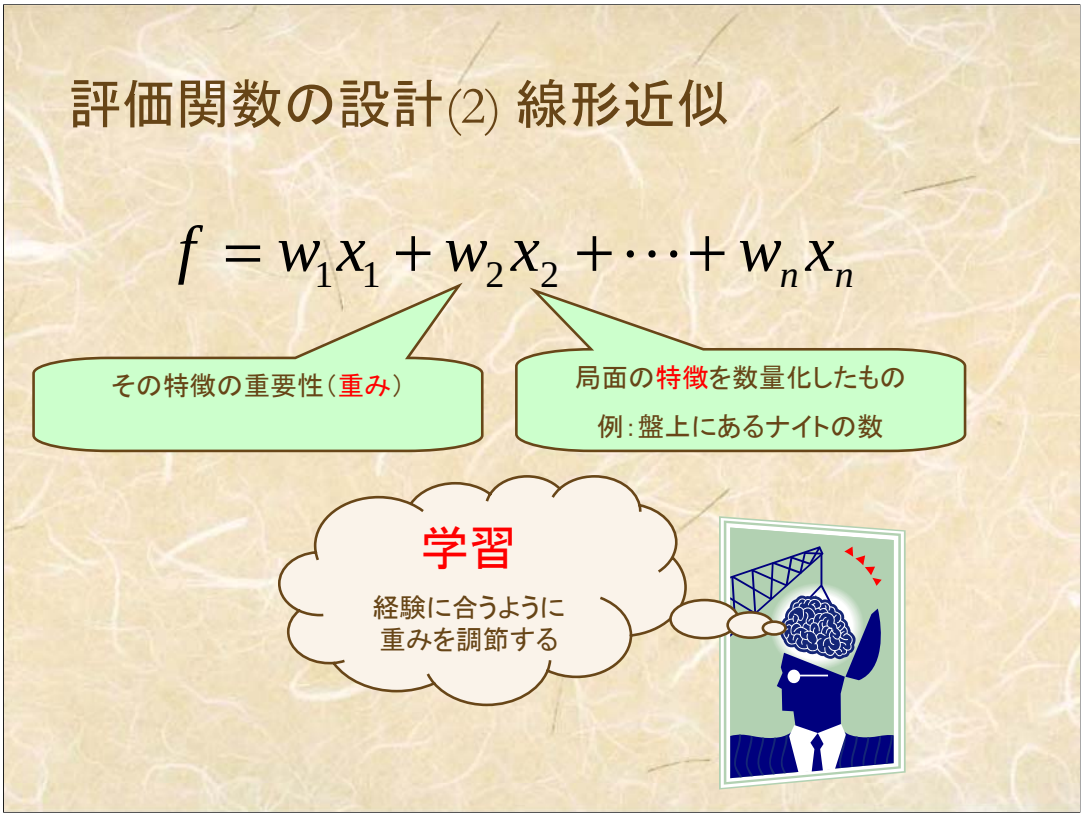
評価関数の設計(1) 基本

- 終端接点では，評価値＝効用値
- あまり長い時間かかってはいけない
- 実際に勝つ可能性を反映していること

勝つ確率	0.5	評価値 ＝1×0.5 ＋(-1)×0.25 ＋0×0.25 ＝0.25
負ける確率	0.25	
引分ける確率	0.25	

厳密である
必要はない

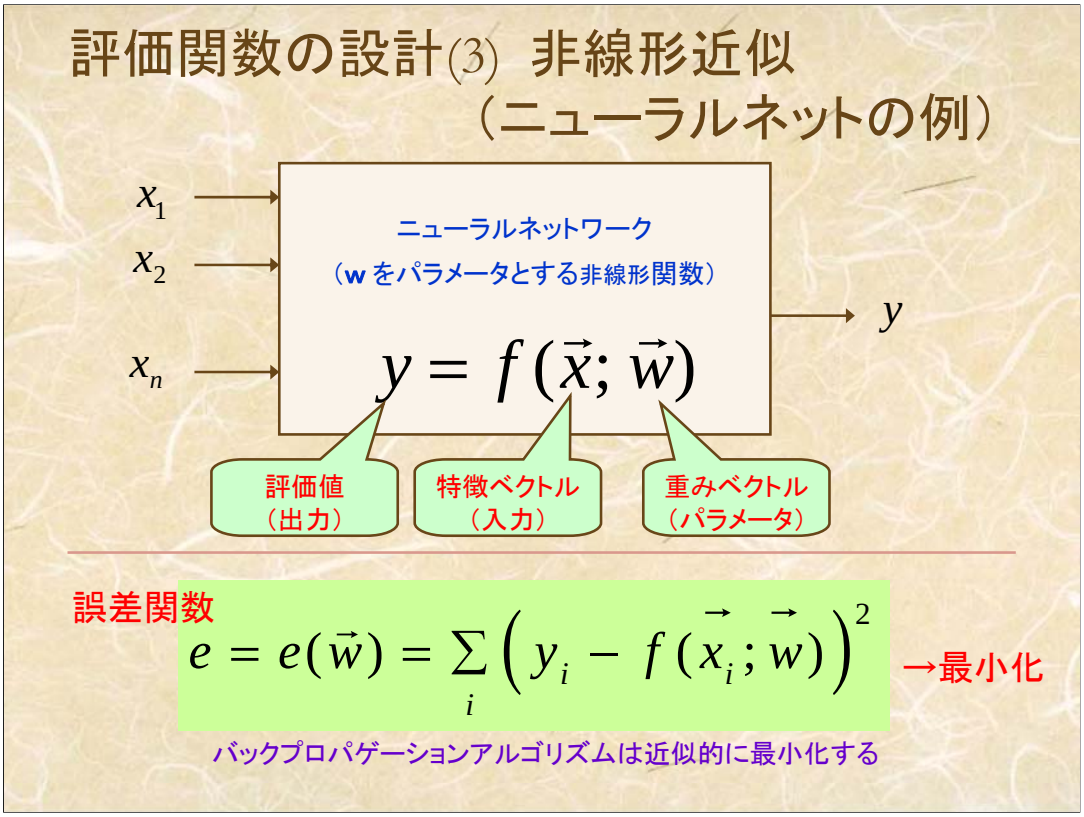
評価関数の設計は，ヒューリスティックな知識に基づいて行うので，その一般論は構成しにくいが，基本的な考え方だけを確認しておこう．当たり前のことだが，このスライドの3点は，最も基本的で重要である．



パターン認識という技術分野では，注目している情報を表す**パターン**から，ヒューリスティックによって事前に定義された**特徴**と呼ばれる(n個の)量を計算し，それらから構成される**特徴ベクトル**がn次元空間上のどこに位置するかに基づいて，そのパターンをいくつかのクラスの1つに**分類**するアルゴリズムが研究されている．ゲームプレイングでもその技術を応用できる．つまり，ゲームの局面からいくつかの特徴を抽出して特徴ベクトルを求め，これが「勝てそうな局面」か「負けそうな局面」かといういずれかのクラスに分類するわけである．

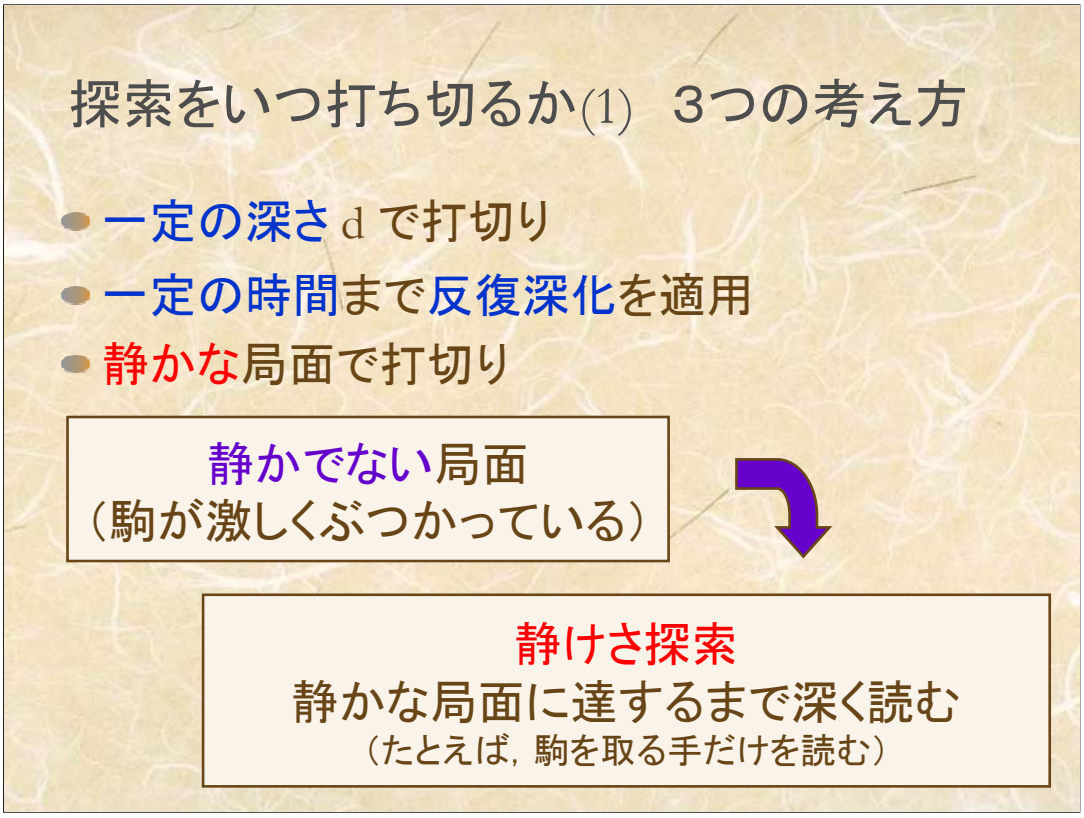
たとえば，勝敗に関係するような局面の特徴を数量化し，それにその特徴の重要性を表す**重み**を掛けたものの総和として評価関数を設計することがある．これを評価関数の**線形近似**という．それを数式で表したのがこのスライドである．

重みの値を事前に決めるのはなかなか難しいことが多い．そのような場合には，何度も何度もゲームの対戦を繰り返し，人工知能で**学習**と呼ばれている技術によって，入力ベクトル x に対する出力 f が，実戦データにできるだけ小さな誤差で合致するように重みベクトル w を調節することができる．学習といっても，すごい技術とは限らず，線形近似の場合には，数値解析で学ぶような最小二乗法によって決定することができる．



評価関数を非線形関数で近似する方法としては，**ニューラルネット**がある．

これは，**ニューロン**と呼ばれる神経細胞を数理モデル化した非線形素子のネットワークによって関数を表現し，たとえば，**バックプロパゲーション（誤差逆伝播）**と呼ばれるアルゴリズムによって，最適な重みベクトルを近似的に決定することができる．この問題は，望ましい入出力関係を表すデータ (x_i,y_i)（**訓練例**という）が何組も与えられたときに，重みベクトル w に応じて誤差の2乗和 を表す非線形関数 e(w) を最小にするための w を求める最適化問題として理解できる．その場合，バックプロパゲーションの本質は，**勾配降下法**という貪欲（グリーディ）な方法で，目先の誤差が小さくなる方向に重みを修正し，（最適値とは限らない）局所最適値によって近似的に最適な重みベクトルを求めるものである．



つぎに，探索をいつ打ち切るかを考える．

少なくとも次の3つの考え方がある．

- 一定の深さdで打ち切り
- 一定の時間まで反復深化を適用
- 静かな局面で打ち切り

一定の深さdで打ち切るというのは，最も初歩的な方法である．この深さdはパラメータであり，プログラマや利用者が事前に設定するものである．

一定の時間で打ち切るというのは，もっと現実的である．いつタイムアウトになってもよいように，探索(2)の授業で学んだ「反復深化」を使って，深さ制限を逐次的に1ずつ深くしながら深さ優先探索をする．そしてタイムアウトになった時点でのベストな指し手を出力する．

「**静かな局面**」で打ち切るということを理解するには，「静かでない局面」を理解するのがよい．これは，駒が激しくぶつかっているような局面で，一手先を読む毎に優劣が激しく変動するような局面である．

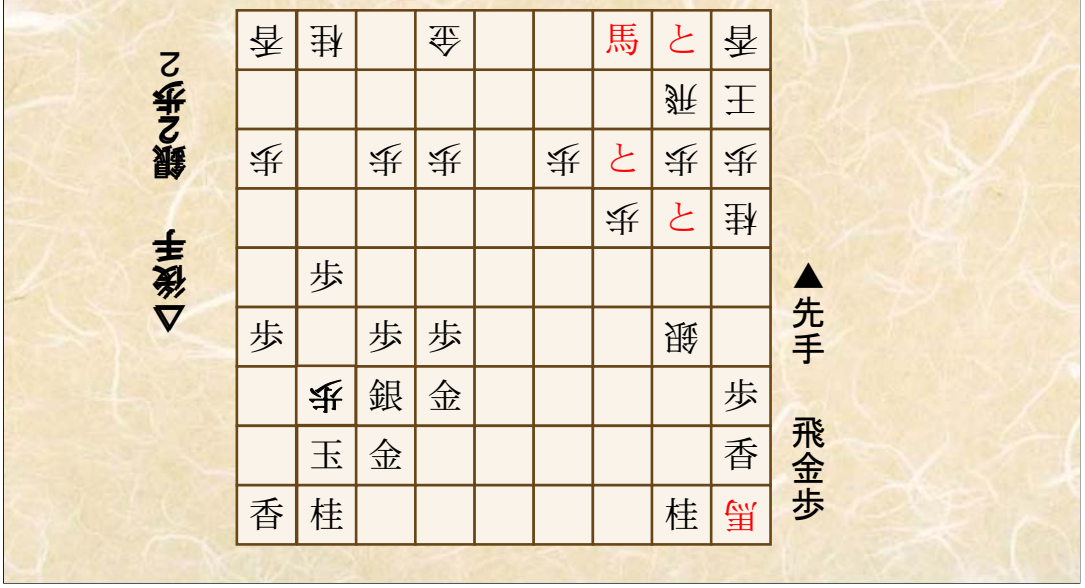
この考え方をとる探索は，**静けさ探索**と呼ばれる．これはある一定の深さ(あるいは時間)まで先読みすると，そこで探索を打ち切るのではなく，そこから先は，たとえば駒を取る手だけを読むなど，局面が静かでない限りは激しい手だけを先読みして，静かな局面に達したら探索を打ち切るものである．

探索をいつ打ち切るか(2)　静かでない局面

静かでない局面の例として，このスライドのように，素人どうしで将棋を3手まで指した局面を考える．これは，駒どうしが激しくぶつかりあっているので，静かでない局面である．もし，ここで探索を打ち切って評価関数を適用すると，明らかに先手が大優勢となる．しかし，もう1手まで先を読むと，後手の銀が先手の馬を取り，ほぼ互角な静かな局面となる．その時点で探索を打ち切るべきである．

探索をいつ打ち切るか(2) 水平線効果

無意味な手の連続で，不利な局面を見つけることのできない水平線の向こうへ追いやって安心する

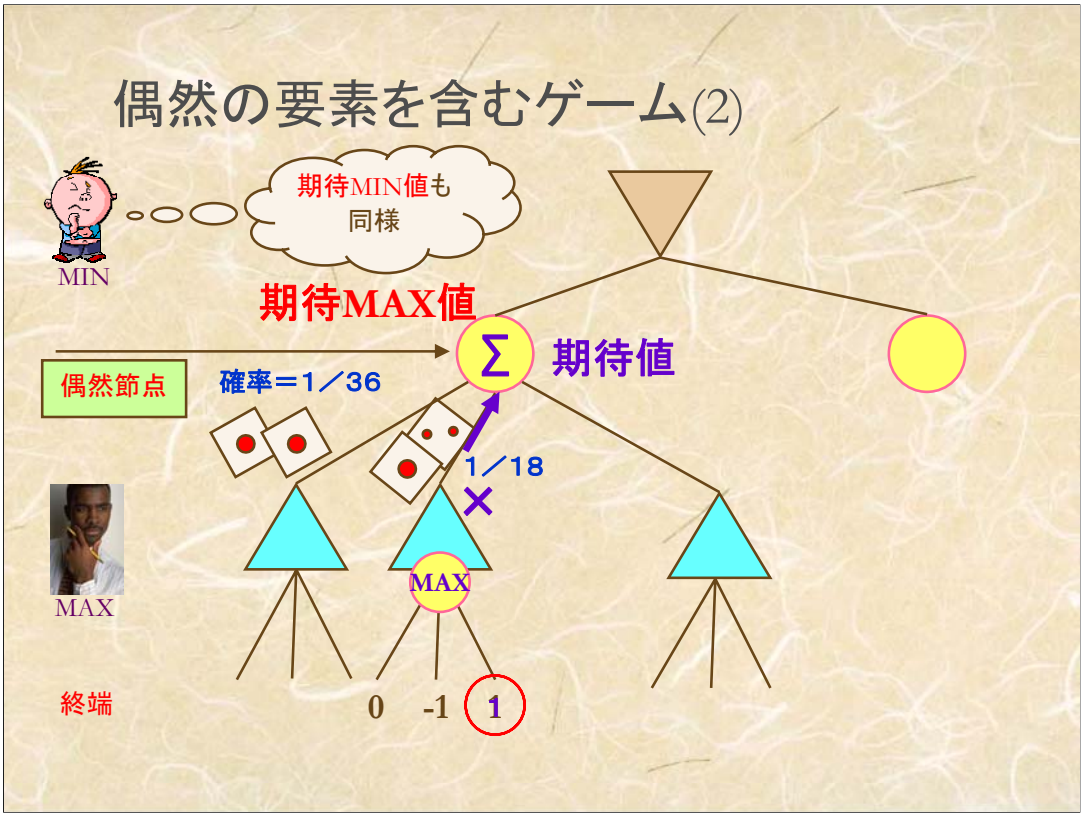


先読みの深さを固定したときには，**水平線効果**と呼ばれる現象が生じるかもしれない．たとえば，コンピュータは5手先まで読むとしよう．コンピュータが不利な局面になってくると，このスライドで後手が先手玉の頭に歩を打っているように，コンピュータは無意味な手を指すことがある．それは，そのような無意味な手を連続して指すことによって，ゲーム木の中で不利な局面が生じる深さが5手よりさらに先になるからである．コンピュータはそれ以上先を読まないので，あたかも不利な局面を回避できたかのように考え，その無意味な手がすばらしい好手であるかのように判断してしまうのである．この5手という先読み深さを海に見える水平線までの距離にたとえて，無意味な手によって不利な局面を水平線の向こうに追いやり，見えなくなって安心してしまう様子を表現しているネーミングである．

偶然の要素を含むゲーム(1)

サイコロ(dice)の目によって取りうる手が制限される

チェスや囲碁・将棋は偶然に左右されないが，バックギャモンのように，サイコロを使うゲームは偶然の要素を含んでいる．



偶然の要素を含むゲームでも，これまでの考え方を少し拡張すれば，ほぼ同様のアルゴリズムを構成できる．具体的には，MAXやMINのノードの間に**偶然ノード**というものを設け，そこで子ノードの評価値の期待値を計算するのである．MAXはその期待値が最大の指し手を選び，MINは期待値が最小の指し手を選ぶ．

ゲームプログラムの現在

チェス	1997年にコンピュータ（Deep Blue）が名人Kasparovに勝利
チェッカー	コンピュータが世界チャンピオン
オセロ	ふつうのプログラムでも人間より強い
バックギャモン	トップレベルの実力
囲碁	徹底的な研究開発が必要
将棋	2005年プロに角落ちで勝利






ゲームプログラムの強さは，現在，およそこの表のようになっている．

1997年にチェス名人に勝った**Deep Blue** は，チェス専用に開発した超並列マシンであり，現在は使われていない．現在は，パソコン上に実装されたプログラムでも，十分に名人クラスの腕前のようである．私はチェスはルールを知っている程度なのだが，航空機の国際線の座席で提供されているミニのチェスプログラムにでさえも，10手くらい指すともう必敗の局面にされてしまう．

囲碁は，チェスよりはるかに可能な指し手の数が多く，探索空間は膨大である．現在の囲碁プログラムは，そこそこのアマチュアにも負けてしまう．強い囲碁プログラムを作るには，かなり新しい発想の人工知能アルゴリズムが必要かもしれない．

将棋は持ち駒を打つことができるので，やはりチェスよりもはるかに探索空間が広いが，囲碁ほどではない．そのため，将棋プログラムはまだプロには負けるが，アマの初段～3段程度よりは強いらしい．「**激指**」（げきさし）というプログラムは，2005年にプロ棋士が角を落とした持ち時間25分の対戦でプロに勝っている．そのため，コンピュータがプロに勝利する日は近いと考えられている．

「激指」がプロ棋士に角落ちで勝利(2005)

開始日時：2005/05/05，持ち時間：25 分切れ負け
場所：木更津市かずさアーク，手合割：角落ち
△：勝又清和五段，▲：激指

△６二銀	▲７六歩	△４二玉	▲２六歩	△３二玉	▲２五歩
△５四歩	▲２四歩	△同　歩	▲同　飛	△５三銀	▲２六飛
△２三歩	▲６八玉	△６四歩	▲３八銀	△５二金右	▲７八銀
△７四歩	▲７九玉	△６三金	▲７七角	△６五歩	▲５八金右
△６四金	▲２七銀	△２二銀	▲３六銀	△４二金	▲３五銀
△５五歩	▲１六歩	△９四歩	▲１五歩	△８四歩	▲９六歩
△８五歩	▲３六歩	△５四銀	▲５九角	△７五歩	▲同　歩
△７二飛	▲３七桂	△７五飛	▲４六歩	△７四飛	▲２七飛
△７五金	▲７七歩	△８六歩	▲同　歩	△８四飛	▲２四歩
△同　歩	▲４八角	△６六歩	▲５六歩	△同　歩	▲６六歩
△９五歩	▲同　歩	△９七歩	▲２三歩	△同　銀	▲２四銀
△同　銀	▲同　飛	△２三歩	▲２五飛	△６五歩	▲５五歩
△６三銀	▲６五歩	△６六歩	▲７六銀	△同　金	▲同　歩
△５七銀	▲同　金	△同歩成	▲同　角	△５六金	▲３九角
△９五香	▲９六歩	△同　香	▲４五銀	△６七銀	▲８七銀
△９八歩成	▲同　香	△同香成	▲同　銀	△８六飛	▲８七金
△８四飛	▲５六銀	△同銀成	▲８五歩	△同　飛	▲２二歩
△同　玉	▲２四歩	△同　歩	▲２三歩	△３二玉	▲２四飛
△６七歩成	▲２二歩成	△４一玉	▲３二と	△同　玉	▲２五香
△４四歩	▲２一飛成	△４三玉	▲５一龍	△５二歩	▲３五桂
△５三玉	▲４五桂　まで激指の勝ち				

棋譜出典：小谷善行，"コンピュータ将棋の歴史的瞬間：プロ棋士に角落ちで勝利"，情報処理，vol.46，no.7，pp.811-813(2005)

実現確率打ち切りアルゴリズム

プロ棋士ならどのくらいの確率でその手を指すかを推定し，
それを親ノードから子ノードへの遷移確率とし，
ノードごとに「実現確率」を計算して，
探索の深さを制御する．

これが激指がプロ棋士に角落ちで勝利した棋譜である．将棋がわかる人は盤面で並べてみてほしい．

激指では「**実現確率打ち切りアルゴリズム**」を考案してそれを実装している．このアルゴリズムは，プロ棋士ならどのくらいの確率でその手を指すかを，大量の棋譜から推定（学習）し，それを親ノードから子ノードへの遷移確率とし，ノード毎に

子ノードの実現確率　＝　親ノードの実現確率　×　親ノードから子ノードへの遷移確率

で計算される**実現確率**を計算する．そして実現確率が事前に決めておいた値より小さくなったらそれ以上の深い読みを打ち切るようにしている．

この激指の例もそうであるが，最近の傾向として，強いゲームプログラムを作る秘訣は，いかにゲームの戦略を数理的にモデル化して優れた探索アルゴリズムを考案するかにあるようだ．単にプログラミング技術に優れていて効率良く実装できればよいという時代は過ぎ去っている．また，Deep Blue や激指の開発者がチェスや将棋に強いわけではないことから，そのゲームに強い必要もなさそうなのである．