情報論理工学研究室

第8回:
ミニマックス法

1

「強い手」の選択

- 「強い手」の選択
  - 強い手とは？
    - 大きな得点が得られる手
    - 相手の得点を下げる手
    - 価値の高い駒を取る手
    - 価値の高い駒を守る手
    - 有利な地点を取る手
    - 相手に不利な地点を取らせる手
    - 有利な選択ができるようになる手
    - 相手に不利な選択を強要する手

ゲームによって異なる

2

「強い手」を得る手法

- 「強い手」を得る手法
  - 局面の評価値計算
  - 定跡・定石データベースの利用
  - 先読み
  - 完全読み切り・必勝読み切り
  - モンテカルロ法
  - 機械学習

3

局面の評価値計算

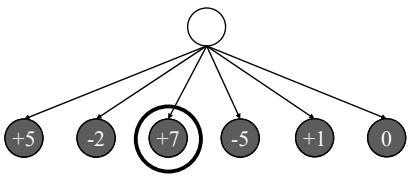
- 局面の評価値計算
  - 現在の局面からどのくらい優勢かを計算する
    - 得点を多く取っている
    - 盤上に強い駒がある
    - 強いカードを持っている
    - 有利な地点を抑えている
    - 相手を攻撃できる
    - 相手の攻撃を防げる
    - 可能な手の数が多い

:

4

評価値計算による手の選択

1. 各合法手に対する1手先の局面を生成
2. 各局面の評価値を計算
3. 最も評価値の高い手を選択



5

```
/* 最も評価値の高い手を選ぶ */
Move SelectMove () {
  ArrayList<Move> moveList = generateMoveList (); // 合法手リスト生成
  if (moveList.isEmpty()) return null; // 合法手無し
  int selectedValue = -∞; // 局面の評価値
  Move selectedMove = null; // 選択した手
  for (Move move : moveList) {
    Phase phase = nextPhase (move); // 1手先の局面を生成
    int value = phase.getValue(); // 局面の評価値を求める
    if (value > selectedValue) { // 評価値最大の手を記憶
      selectedMove = move;   selectedValue = value;
    }
  }
  return move;
}
```

6

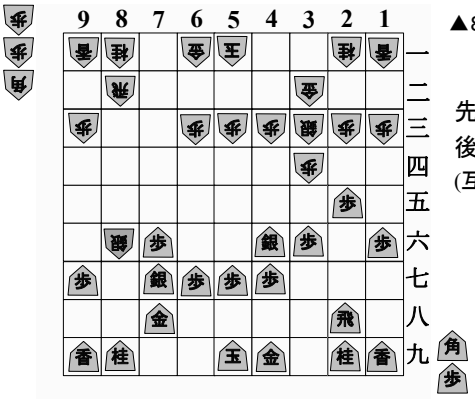
駒割りによる評価値計算

- 駒割りによる評価値計算
  - 強い駒を取れる手を高評価にする

これである程度はいい手を選べるが…

7

駒割による評価値：将棋



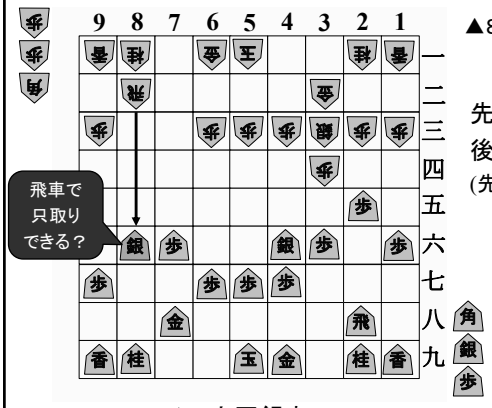
▲8六銀

先手：93点
後手：93点
(互いに駒損無し)

△8六銀まで

8

駒割による評価値：将棋



▲8六銀

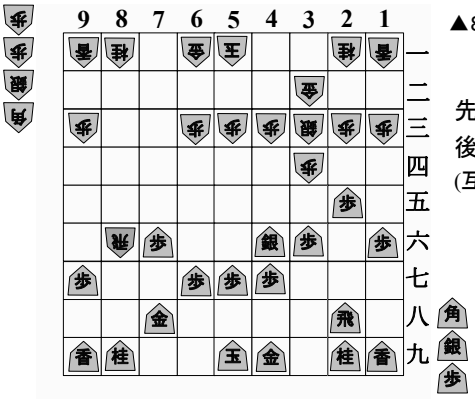
先手：101点
後手：85点
(先手の銀得)

8六の銀が
8二の飛車で
取れそう

▲8六同銀まで

9

駒割による評価値：将棋



▲8六銀　△同飛

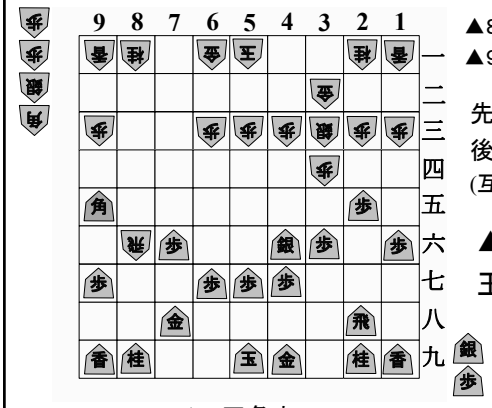
先手：93点
後手：93点
(互いに駒損無し)

しかし…

△8六同飛まで

10

駒割による評価値：将棋



▲8六銀　△同飛
▲9五角

先手：93点
後手：93点
(互いに駒損無し)

▲9五角が
王手飛車！

▲9五角まで

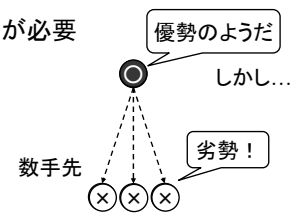
11

先読み

現在の局面は優勢でも
数手先に逆転される場合がある

⇩

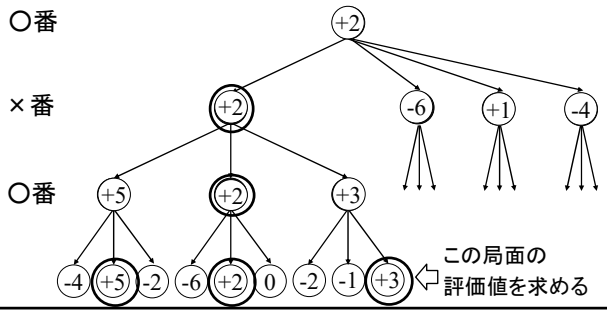
局面の先読みが必要



12

先読み

- 先読み
  - 数手先の局面を生成し、評価値を計算



13

ミニマックス(mini-max)法

- ミニマックス法
  - 自分にとっての最善手＝相手にとっての最悪手
    (二人零和ゲームの場合)

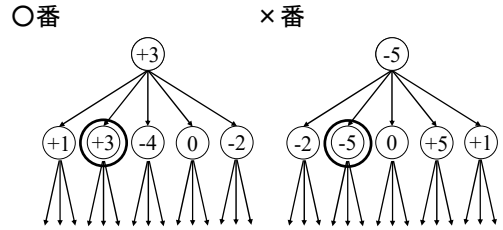
⇒ 相手が常に最善手を指してくると仮定

自分の手番：最も評価値の高い手を採用
相手の手番：最も評価値の低い手を採用

14

各頂点の得点計算

○番：最大値を選択
×番：最少値を選択



15

評価値計算

自分の手番：合法手中で最大の評価値を返す

int maxScore (int depth)

depth-1 で
呼び出し

depth-1 で
呼び出し

相手の手番：合法手中で最小の評価値を返す

int minScore (int depth)

depth 先まで読む

16

```
/* 局面の評価値を計算する(自分の手番) */
/* depth 先まで読んで最も高い評価値を返す */
int maxScore (int depth) {
  if (depth == 0) // 深さ制限に達した場合
    return value; // 先読み無しの評価値を返す
  int scoreMax = -∞;
  ArrayList<Move> moveList = generateMoves(); // 合法手リスト生成
  for (Move move : moveList ) { // 全ての合法手に対して判定
    Phase phase = nextPhase (move); // 次の局面を生成
    int score = phase.minScore (depth -1); // 次局面(相手番)の評価値計算
    if (score > scoreMax) // 自分にとって良い手が見つかった場合
      scoreMax = score; // 評価値更新
  }
  return scoreMax; // 最も高い評価値を返す
}
```

17

```
/* 局面の評価値を計算する(相手の手番) */
/* depth 先まで読んで最も低い評価値を返す */
int minScore (int depth) {
  if (depth == 0) // 深さ制限に達した場合
    return value; // 先読み無しの評価値を返す
  int scoreMin = +∞;
  ArrayList<Move> moveList = generateMoves(); // 合法手リスト生成
  for (Move move : moveList ) { // 全ての合法手に対して判定
    Phase phase = nextPhase (move); // 次の局面を生成
    int score = phase.maxScore (depth -1); // 次局面(自番)の評価値計算
    if (score < scoreMin) // 相手にとって良い手が見つかった場合
      scoreMin = score; // 評価値更新
  }
  return scoreMin; // 最も低い評価値を返す
}
```

18

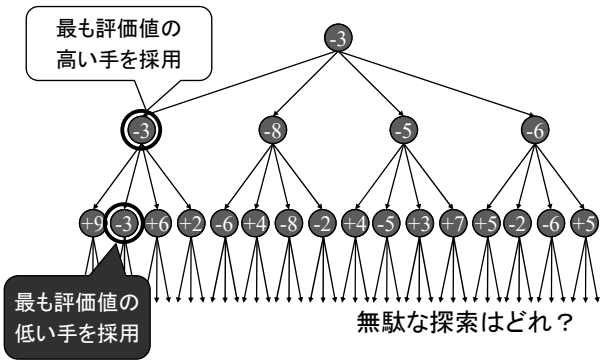
可能な局面数

		各局面での合法手数		
		2	4	8
先読み手数	10	1,000	1,000,000	10^9
	20	1,000,000	10^{12}	10^{18}
	30	10^9	10^{18}	10^{27}
	40	10^{12}	10^{24}	10^{36}
	50	10^{15}	10^{30}	10^{45}

先読み手数が増えると可能な局面数は指数的に増える
⇒適当な枝刈りが必要

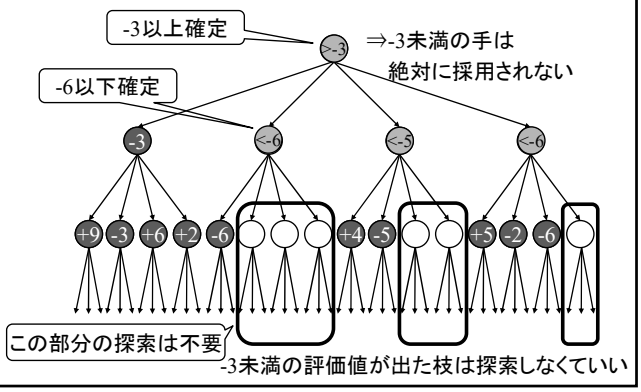
19

ミニマックス法



20

ミニマックス法



21

アルファベータ(alpha-beta)法

- アルファベータ法
  - ミニマックス法の改良アルゴリズム
    - 必要の無い探索は行わない
    - 絶対に採用されない手は読まない

α：それまでに発見した自番で最も大きな評価値
β：それまでに発見した相手番で最も小さい評価値

相手の手番：α よりも小さい評価値になれば探索打ち切り
自分の手番：β よりも大きい評価値になれば探索打ち切り

α 以上 β 以下の手を探索する

22

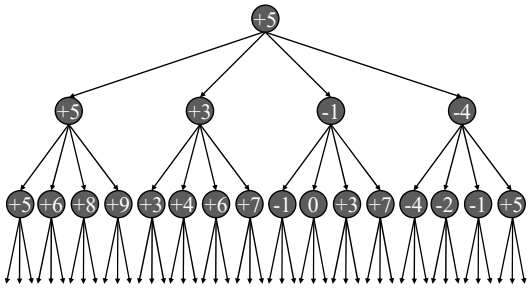
```
/* αβ法により局面の評価値を計算する(自分の手番) */
/* depth 先まで読んで最も高い評価値を返す */
int maxScore (int depth, int alpha, int beta) {
  if (depth == 0) // 深さ制限に達した場合
    return value; // 先読み無しの評価値を返す
  int scoreMax = -∞;
  ArrayList<Move> moveList = generateMoves(); // 合法手リスト生成
  for (Move move : moveList ) { // 全ての合法手に対して判定
    Phase phase = nextPhase (move); // 次の局面を生成
    int score = phase.minScore (depth -1, alpha, beta); // 次局面の評価値計算
    if (score≧beta) // β値を上回ったら探索中止
      return score; // この値は採用されることは無い
    if (score > scoreMax) // 自分にとって良い手が見つかった場合
      scoreMax = score; // 評価値更新
      if (scoreMax > alpha) alpha = scoreMax; // α値更新
  }
  return scoreMax; // 最も高い評価値を返す
}
```

23

```
/* αβ法により局面の評価値を計算する(相手の手番) */
/* depth 先まで読んで最も低い評価値を返す */
int minScore (int depth, int alpha, int beta) {
  if (depth == 0) // 深さ制限に達した場合
    return value; // 先読み無しの評価値を返す
  int scoreMin = +∞;
  ArrayList<Move> moveList = generateMoves(); // 合法手リスト生成
  for (Move move : moveList ) { // 全ての合法手に対して判定
    Phase phase = nextPhase (move); // 次の局面を生成
    int score = phase.maxScore (depth -1, alpha, beta); // 次局面の評価値計算
    if (score≦alpha) // α値を下回ったら探索中止
      return score; // この値は採用されることは無い
    if (score > scoreMin) // 相手にとって良い手が見つかった場合
      scoreMin = score; // 評価値更新
      if (scoreMin < beta) beta = scoreMin; // β値更新
  }
  return scoreMin; // 最も低い評価値を返す
}
```

24

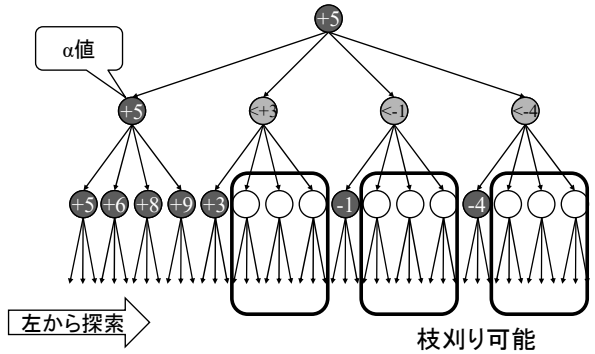
アルファベータ法の効率



枝刈りできるのはどこ？

25

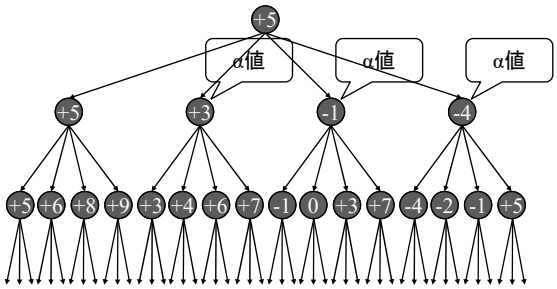
アルファベータ法



枝刈り可能

26

アルファベータ法

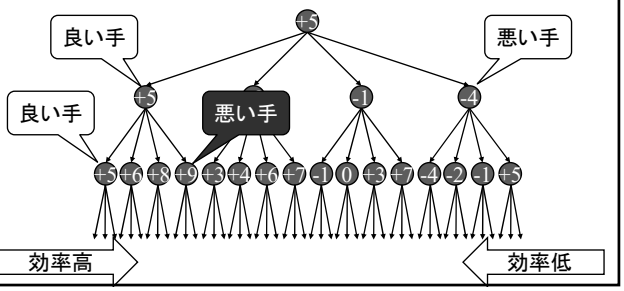


全く枝刈りできない！

27

アルファベータ法の効率

- アルファベータ法の効率
  - 良い手から先に探索すると効率がいい



28

アルファベータ法の効率

- アルファベータ法の効率
  - 良い手から先に探索すると効率がいい

でもどうやって「良い手」を探す？

そもそも「良い手」がわかるなら探索の必要無し

「良さそうな手」から先に探索

29

効率のいい探索

- まず予備探索を行う
  1. ミニマックス法(先読み数：小)で探索
  2. 評価値順に手をソートする
  3. アルファベータ法(先読み数：大)で探索

探索が2回必要だが、
枝刈りできるメリットの方が大きい

30

ネガマックス(negamax)法

- ネガマックス法
  - 関数maxScore()とminScore()を一つにまとめる
    - この2つはほぼ同じ関数(最大値か最小値かの違い)

⇒評価値の符号を反転させれば同一

自分の手番：評価値そのまま使用
相手の手番：評価値*-1を使用
⇒手番に関係なく評価値が最大の手を探せばいい

31

```
/* ネガマックス法により局面の評価値を計算する */
/* depth 先まで読んで最も高い評価値を返す */
int negaMaxScore (int depth, int alpha, int beta) {
 if (depth == 0) // 深さ制限に達した場合
   return value; // 先読み無しの評価値を返す
 int scoreMax = -∞;
 ArrayList<Move> moveList = generateMoves(); // 合法手リスト生成
 for (Move move : moveList ) { // 全ての合法手に対して判定
   Phase phase = nextPhase (move); // 次の局面を生成
   int score = -phase.negaMaxScore (depth -1, -beta, -alpha); // 次局面
   if (score≧beta) // β値を上回ったら探索中止
     return score; // この値は採用されることは無い
   if (score > scoreMax) // 良い手が見つかった場合
     scoreMax = score; // 評価値更新
     if (scoreMax > alpha) alpha = scoreMax; // α値更新
 }
 return scoreMax; // 最も高い評価値を返す
}
```

32

ネガマックス法

```
int negaMaxScore (int depth, int alpha, int beta)
```

```
int score = -phase.negaMaxScore (depth-1, -beta, -alpha)
```

相手番の評価値計算
符号を入れ替え、α:=-β、β:=-αとして計算する

評価値 v が $\alpha < v < \beta$
⇔評価値 –v が $-\beta < -v < -\alpha$

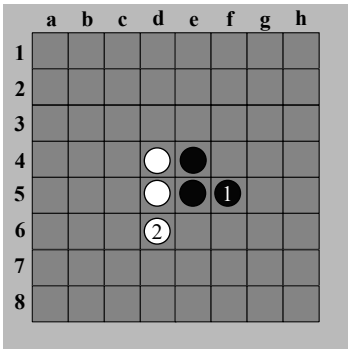
33

同一局面の処理

- 探索中同一局面が現れるケース
  - 手順前後の同一局面
    - 手順が違っても同一局面となる
      ⇒局面の評価値を再利用できる
  - 千日手
    - 手順中で同一局面が現れる(千日手)
      ⇒探索の無限ループを回避する必要がある

34

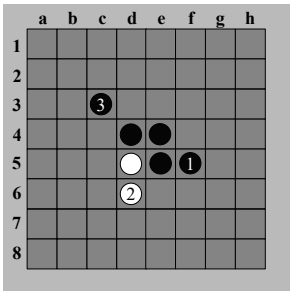
手順前後の同一局面



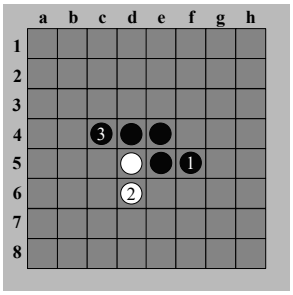
1:f5, 2:d6

35

手順前後の同一局面



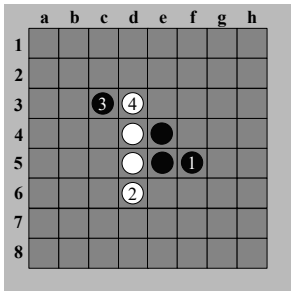
1.f5, 2.d6, 3.c3



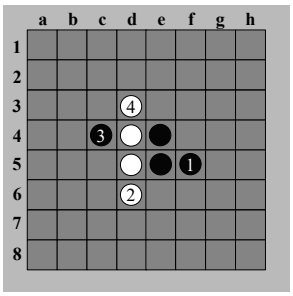
1.f5, 2.d6, 3.c4

36

手順前後の同一局面



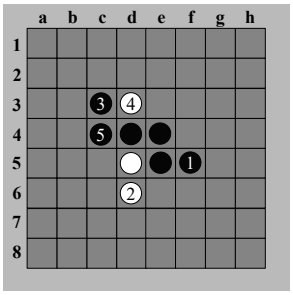
1.f5, 2.d6, 3.c3, 4.d3



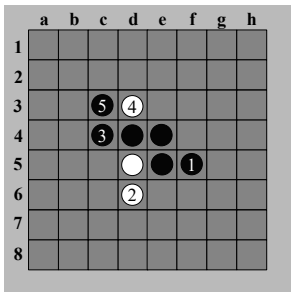
1.f5, 2.d6, 3.c4, 4.d3

37

手順前後の同一局面



1.f5, 2.d6, 3.c3, 4.d3, 5.c4



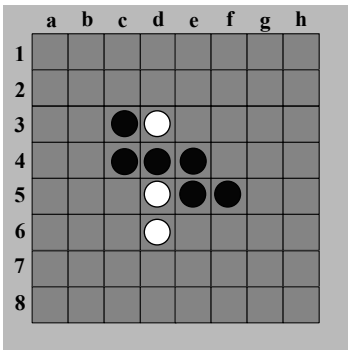
1.f5, 2.d6, 3.c4, 4.d3, 5.c3

38

手順前後の同一局面

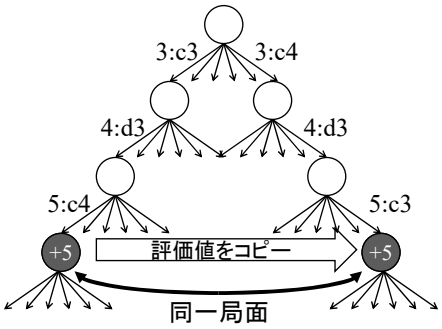
手順1
1:f5, 2:d6, 3:c3, 4:d3, 5:c4
手順2
1:f5, 2:d6, 3:c4, 4:d3, 5:c3

どちらも同じ局面になる



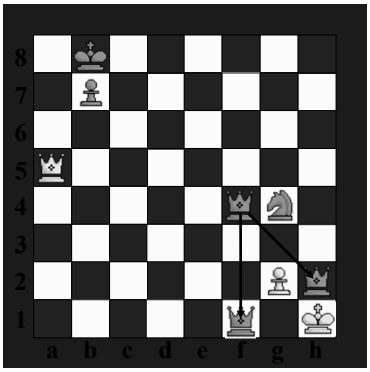
39

手順前後の同一局面



40

千日手による同一局面



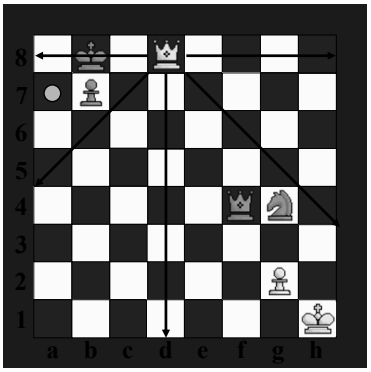
白番

次に黒が以下の
どちらかを指すと
チェックメイト
... Qf1#
... Qh2#

白は攻めるしかない

41

千日手による同一局面

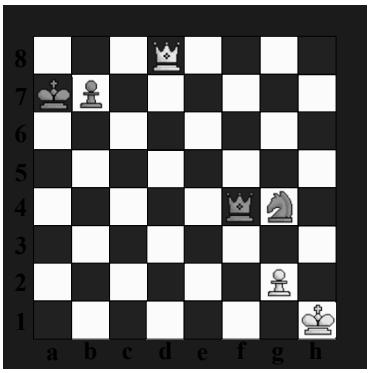


黒番

1. Qd8+

42

千日手による同一局面

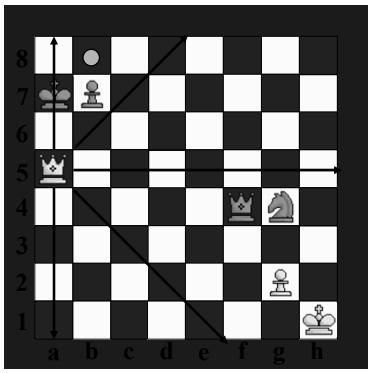


白番

1. Qd8+ Ka7

43

千日手による同一局面

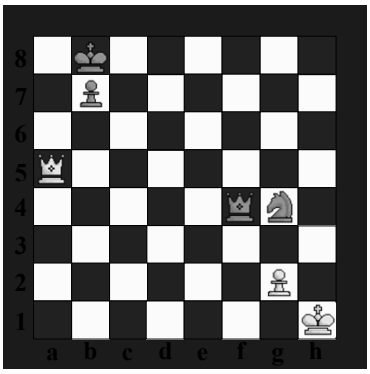


黒番

1. Qd8+ Ka7
2. Qa5+

44

千日手による同一局面



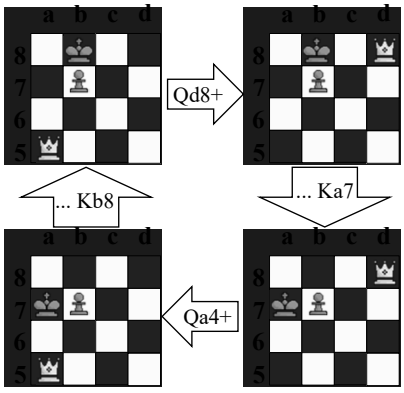
白番

1. Qd8+ Ka7
2. Qa5+ Kb8

最初の局面と同じ

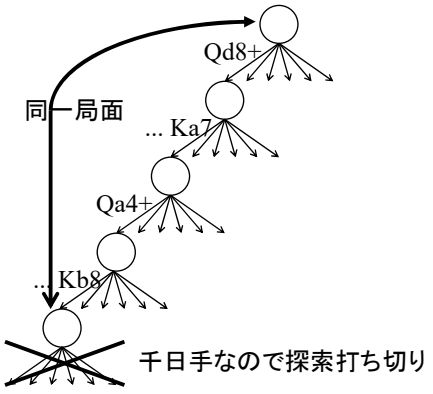
45

千日手による同一局面



46

千日手による同一局面



47

千日手の評価値

- チェスの場合
  - 千日手は引き分け ⇒ 評価値を 0 にする
    - 優勢ならば千日手を避け、劣勢ならば千日手に
- 将棋の場合
  - 千日手は先手後手入れ替えて指し直し
    ⇒ 評価値は後手側やや有利にする
    - 優勢ならば千日手を避け、劣勢ならば千日手に
    - 互角ならば、先手は千日手を避け、後手は千日手に
  - ただし、連続王手の千日手は攻め手の負け
    ⇒ 連続王手の千日手は評価値最低に

48

局面の同一判定

- equals()メソッド
  - 同一の局面か判定する

```
boolean equals (Phase phase) {
  for (int i=0; i<SIZE; ++i)
    for (int j=0; j<SIZE; ++j)
      if (this.board[i][j] ≠ phase.board[i][j])
        return false; // 1箇所でも異なればfalse
  if (this.turn ≠phase.turn) return false;
                :
  return true; // 全て同じならtrue
}
```

だがこの判定は時間がかかる

49

局面の同一判定

- 探索中には多くの局面が現れる
- 局面の同一判定は時間がかかる

⇩

同一の可能性のある局面を絞り込む

ハッシュ関数による同一判定

ハッシュ関数で局面を数値化、
同一のハッシュ値を持つ局面のみ同一判定

50

ハッシュ関数の例：チェス

駒がある：1
駒が無い：0
として64ビットの
数値で表現

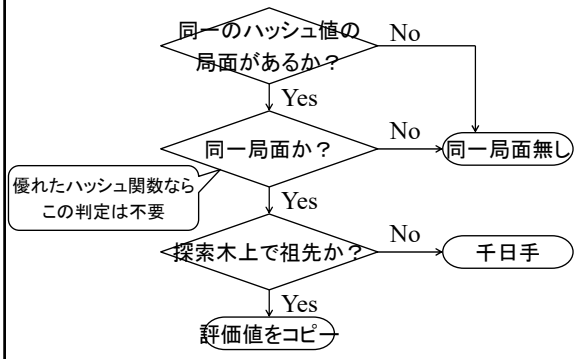
11011111
11100101
00010010
00001000
00101000
0010000
11111111
10111001

DFE51208
2820F7B9

※実際はもっと複雑なハッシュ関数を用いる

51

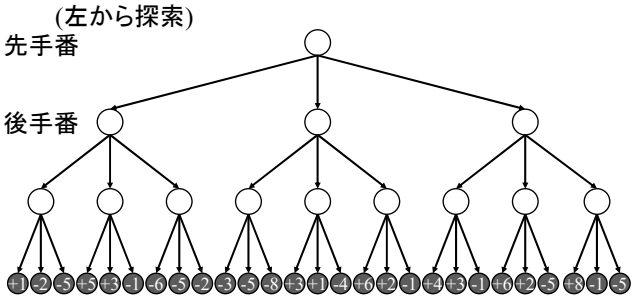
同一局面の判定



52

宿題：アルファベータ法

αβ 法で探索したときに枝刈りできる部分はどこか？
(左から探索)



53

宿題：3目並べの着手選択

- 3目並べ着手選択
  - 先手後手それぞれを人間かCPUのどちらが受け持つかを選べるようにせよ
    - CPUはαβ法を用いて着手決定
      - 探索の深さは適当な値に設定する
      - 先読み無しの評価値も適当と思われる方法を採る

54

参考：三目ならべプログラム

- Tictactoe.java
  - 実行時に Com の強さを指定
    - レベル 0 : ランダム
    - レベル 1 : 自分のリーチは見逃さない
    - レベル 2 : 相手のリーチも見逃さない
    - レベル 3 : 指定した手数先読み

http://www.info.kindai.ac.jp/~takasi-i
からダウンロードし、各自実行してみること

55

課題

- 以下のテーマから1つ選び調査してください
  - 12月21日(水) 2限 発表 (5分～10分)
  - 1月11日(水) 17:00 報告書提出
    - チェス・将棋・囲碁等の強いソフト
    - チェス・将棋・囲碁等の着手選択法
    - コンピュータチェス・将棋・囲碁の歴史
    - 完全解析されているゲーム
    - 並列計算機にはどのようなものがあるか
    - LANを用いた仮想計算機
    - クラスタ処理・グリッド処理
    - その他

56