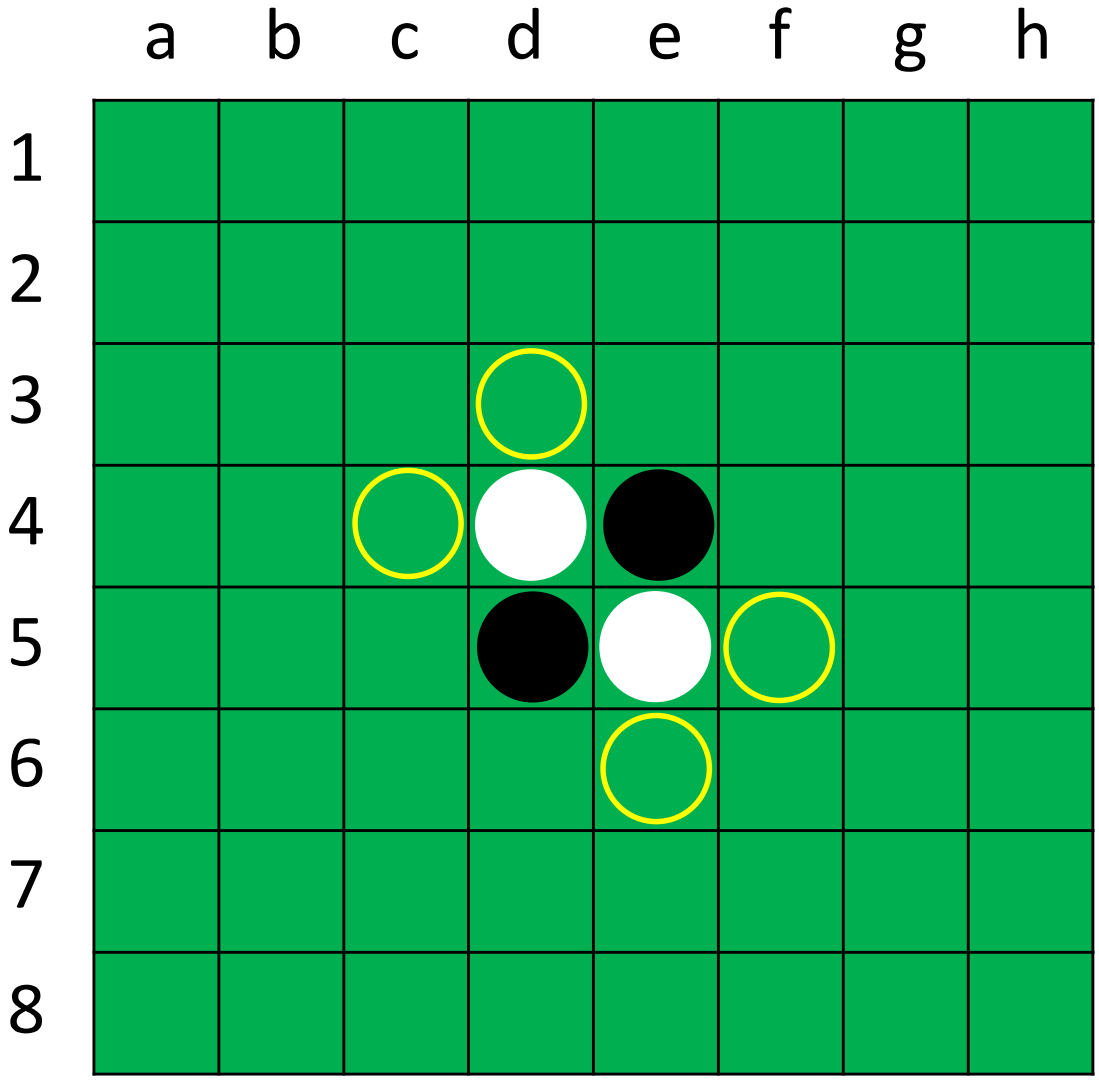
コンピュータオセロ

鶴岡 慶雅

オセロ（リバーシ）



- 黒、白が交互に置く
- 相手のディスクを自分のディスクで挟む（縦・横・斜め）と自分のディスクになる
- 初期局面での合法手は4つ
- 合法手が存在しない場合はパス
- 最終的にディスクの数が多い方が勝ち

サンプルプログラム

- コンパイル

```
$ tar xvzf reversi.tar.gz
$ cd reversi
$ make
```

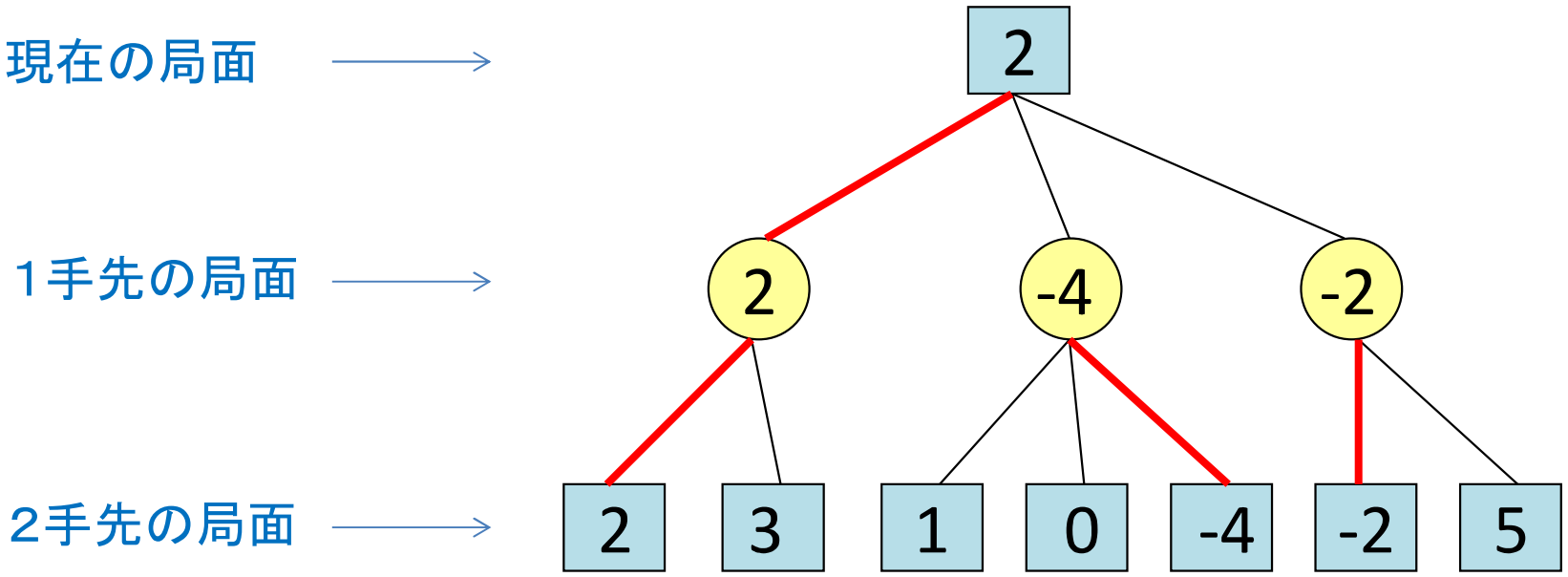
- 実行
  - コンピュータ同士の対戦

    ```
    $ ./a.out
    ```
  - 人間対コンピュータ(人間が先手)

    ```
    $ ./a.out 1
    ```
  - 人間対コンピュータ(人間が後手)

    ```
    $ ./a.out -1
    ```

ミニマックス法
(Minimax algorithm)



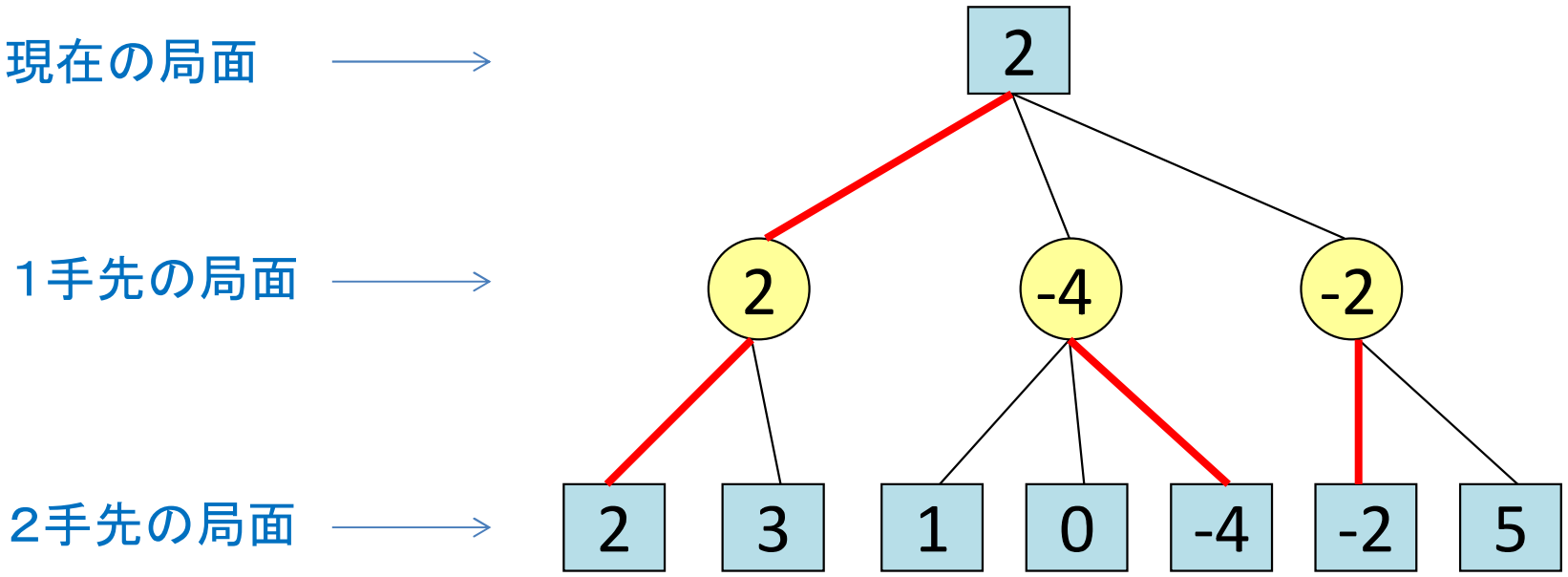
- **評価関数**によって末端局面の有利・不利の度合いを数値化
- お互いが自分にとって最も都合の良い手を選ぶと仮定して、スコア、最善手を逆算

評価関数（evaluation function）

- 局面の有利不利を数値化する関数
  - 互角ならゼロ
  - 先手（黒番）にとって有利であればプラス
  - 後手（白番）にとって有利であればマイナス

- オセロの場合
  - 盤上のディスクの数
    - （黒のディスクの数 ）ー（白のディスクの数）
  - 角のディスクにはボーナス点
  - 角の隣のディスクはちょっと減点
  - ：

深さ優先探索（depth-first search）



- 関数の再帰呼び出しで簡単に実装できる
- 木構造をつくる必要がないので省メモリ

ミニマックス法の実装

- Maxノード

```
int max_node(depth, max_depth)
    if depth == max_depth:
        return eval_func()
    best = -infinity
    for all legal moves:
        update_board(move)
        v = min_node(depth+1, max_depth)
        if v > best:
            best = v
        undo_board()
    return best
```

- Minノード

```
int min_node(depth, max_depth)
    if depth == max_depth:
        return eval_func()
    best = infinity
    for all legal moves:
        update_board(move)
        v = max_node(depth+1, max_depth)
        if v < best:
            best = v
        undo_board()
    return best
```

- 実際には最良スコアだけでなく最良手も返す必要あり
- 木の末端よりも手前で勝負がついていることも

Negamax法

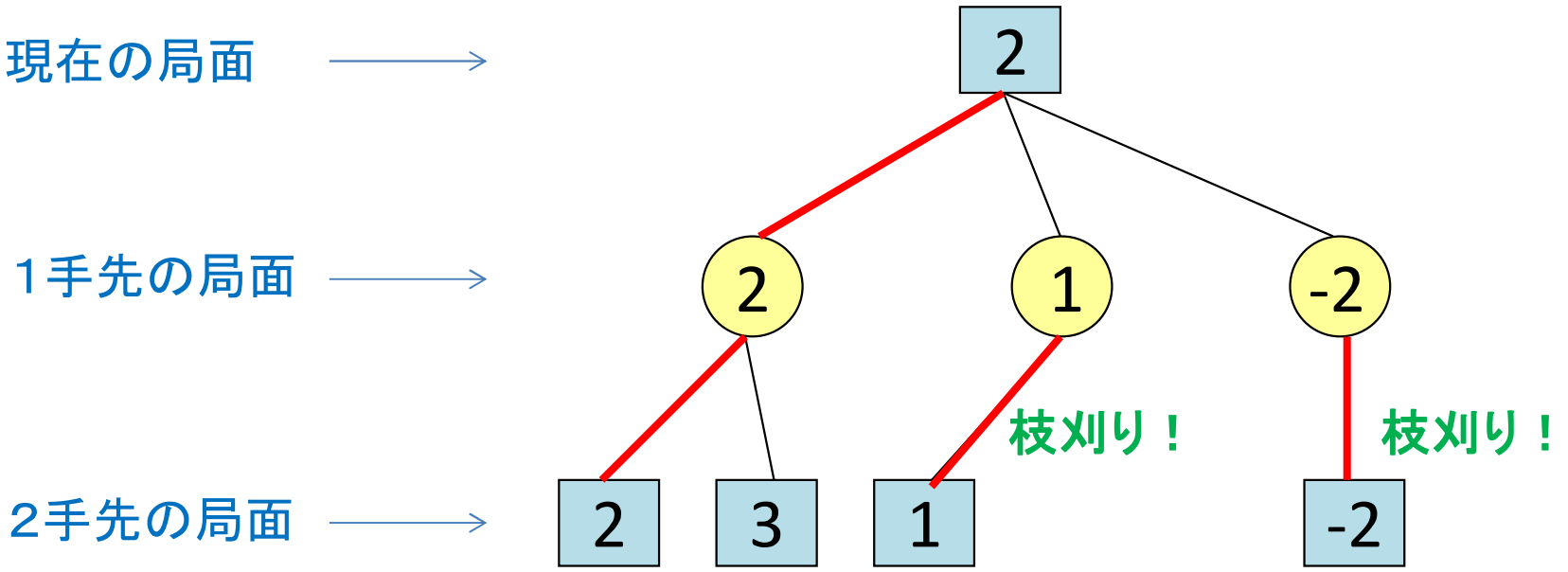
- 対称性を利用することでひとつの関数で書ける

```
int negamax(depth, max_depth)
    if depth == max_depth:
        return eval_func() * turn
    best = -infinity
    for all legal moves:
        update_board(move)
        v = -negamax(depth+1, max_depth)
        if v > best :
            best = v
        undo_board()
    return best
```

turn の値は、今の局面が
先手番であれば +1
後手番であれば -1

頭にマイナスがついていることに注意

枝刈り

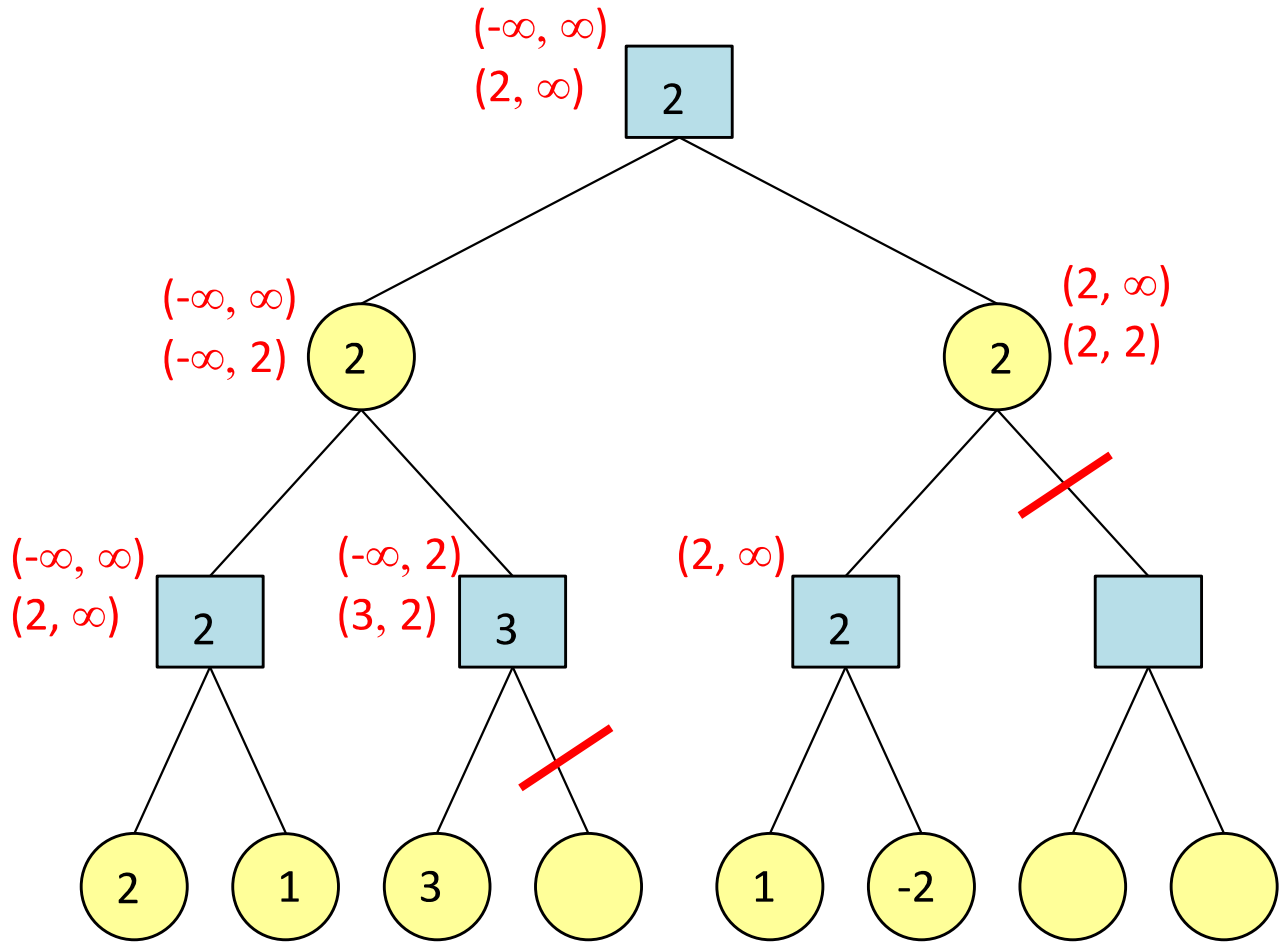


- 探索ノード数を大幅に減らせる
- 現在局面（ルート局面）で選択する手と評価値は変わらない

ミニマックス＋αβ枝刈り

- 各ノードで、ルートノードの値に影響し得る評価値の範囲（ウィンドウ）をαとβの2つの変数で保持
  - α 以下もしくは β 以上の評価値はルートノードに影響しない
  - Maxノードでは α が上がっていく
  - Minノードでは β が下がっていく
  - ウィンドウが閉じたら（α≧βになったら）カット

ミニマックス＋αβ枝刈り



ミニマックス＋αβ枝刈り

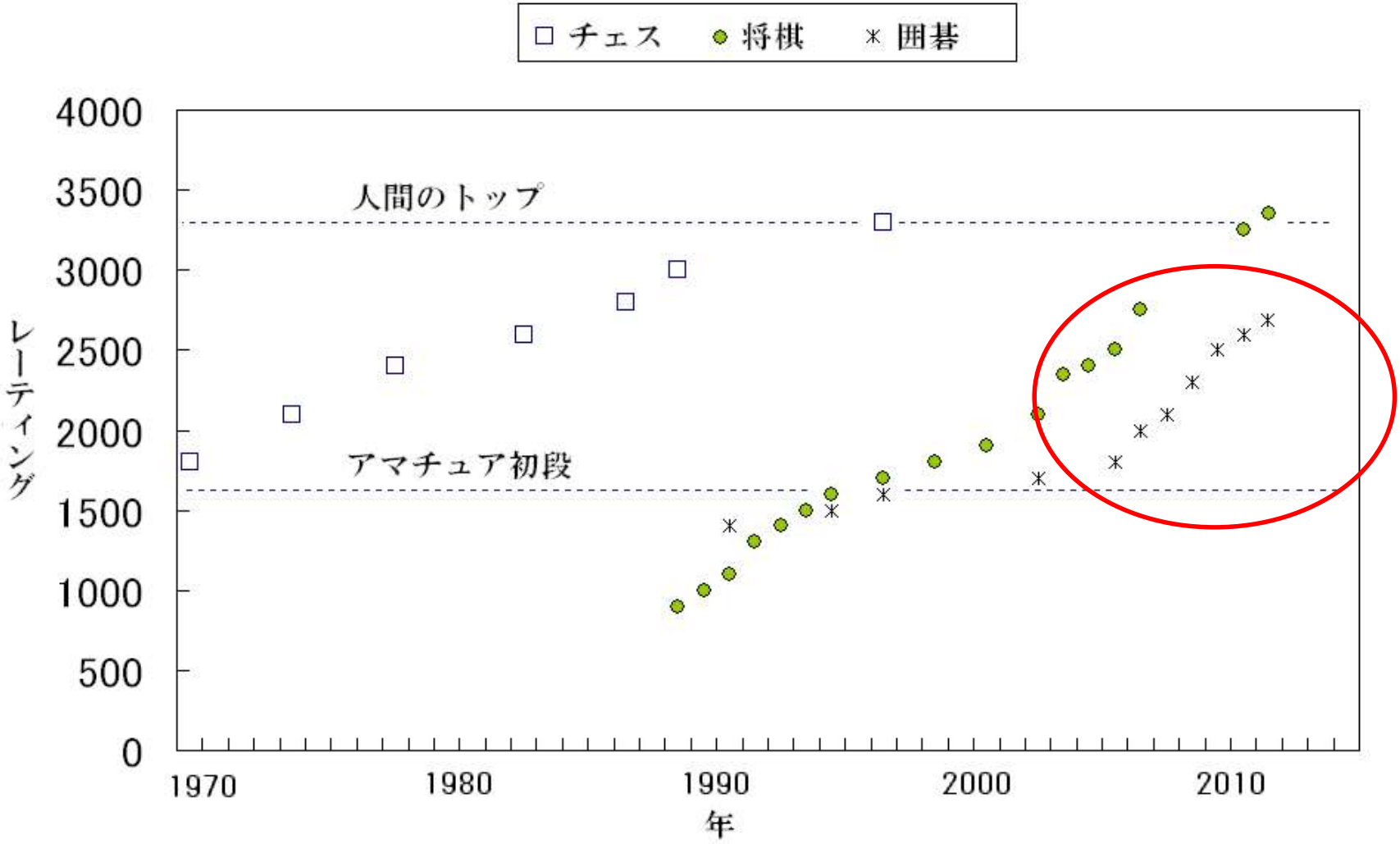
- Maxノード

```
function max_node(board, d, α, β)
    if node is terminal or d = 0:
        return eval_func(board)
    for all legal moves:
        board.update(move)
        v = min_node(board, d – 1 , α, β)
        board.undo(move)
        if v > α :
            α = v
            if α ≥ β :
                break
    return α
```

- Minノード

```
function min_node(board, d, α, β)
    if node is terminal or d = 0:
        return eval_func(board)
    for all legal moves:
        board.update(move)
        v = max_node(board, d – 1 , α, β)
        board.undo(move)
        if v < β :
            β = v
            if α ≥ β :
                break
    return β
```

コンピュータチェス・将棋・囲碁



FPGAで将棋プログラムを作ってみるブログ
http://blog.livedoor.jp/yss_fpga/archives/53897129.html

MCTS

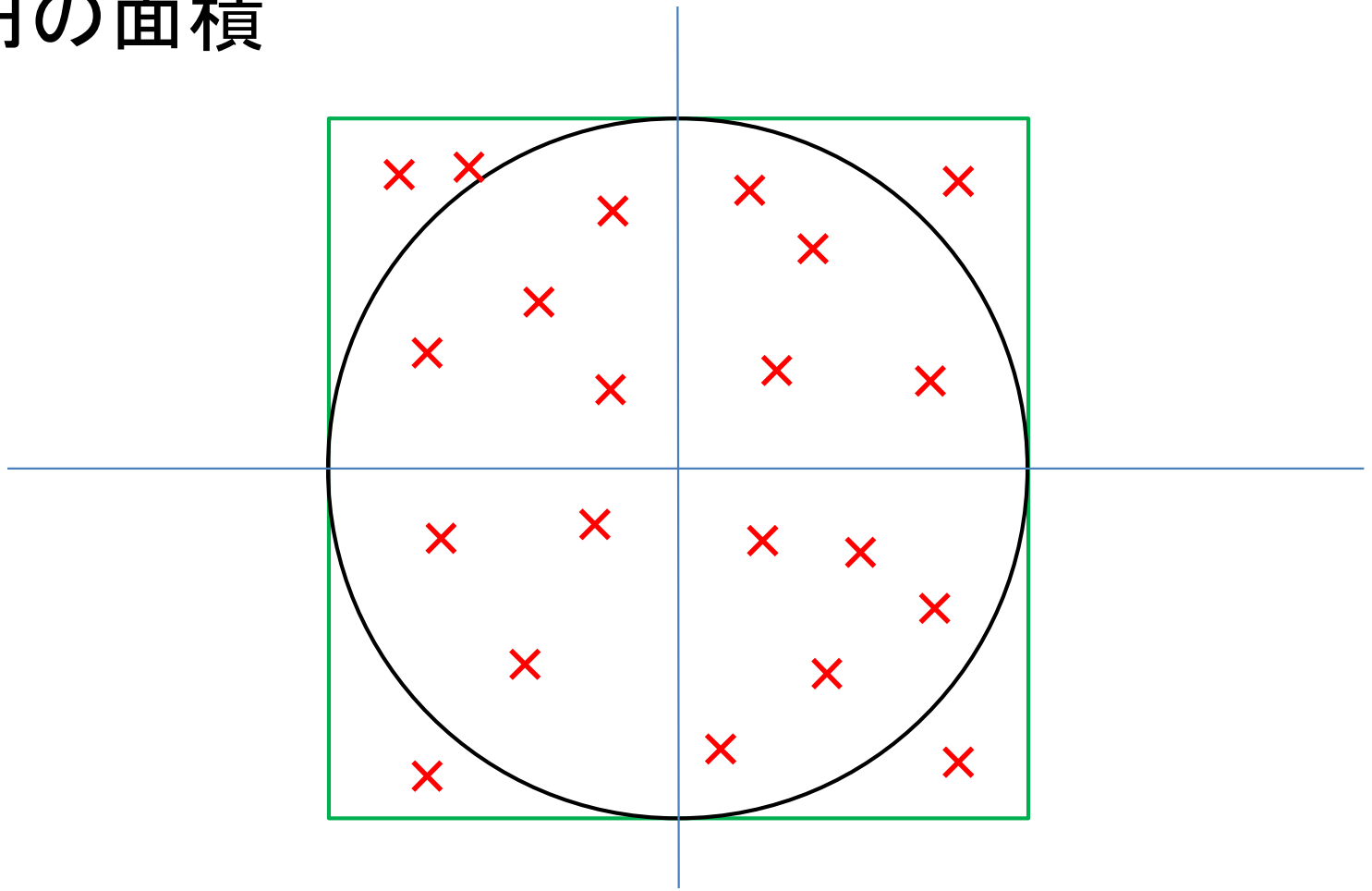
- モンテカルロ木探索（Monte Carlo Tree Search, MCTS）
  - AI 研究に大きな影響
    - 囲碁で大成功
    - 他のゲーム、プランニング、制御、最適化問題などへの応用が進む
  - 特長
    - ドメイン知識が不要
    - 他の手法がうまくいかない難しい問題で成功
    - 計算パワーの向上がそのまま性能の向上につながる

コンピュータ囲碁

- コンピュータ囲碁
  - 初段手前でしばらく停滞
  - MCTS の登場（2006年ごろ）
  - アマトップクラス？
  - 難しさ
    - 合法手が多い
    - 評価関数の設計が難しい
      - 地が確定するのは最後
      - 石の生死の判定
      - 離れた場所にある石の影響
      - etc

モンテカルロ法

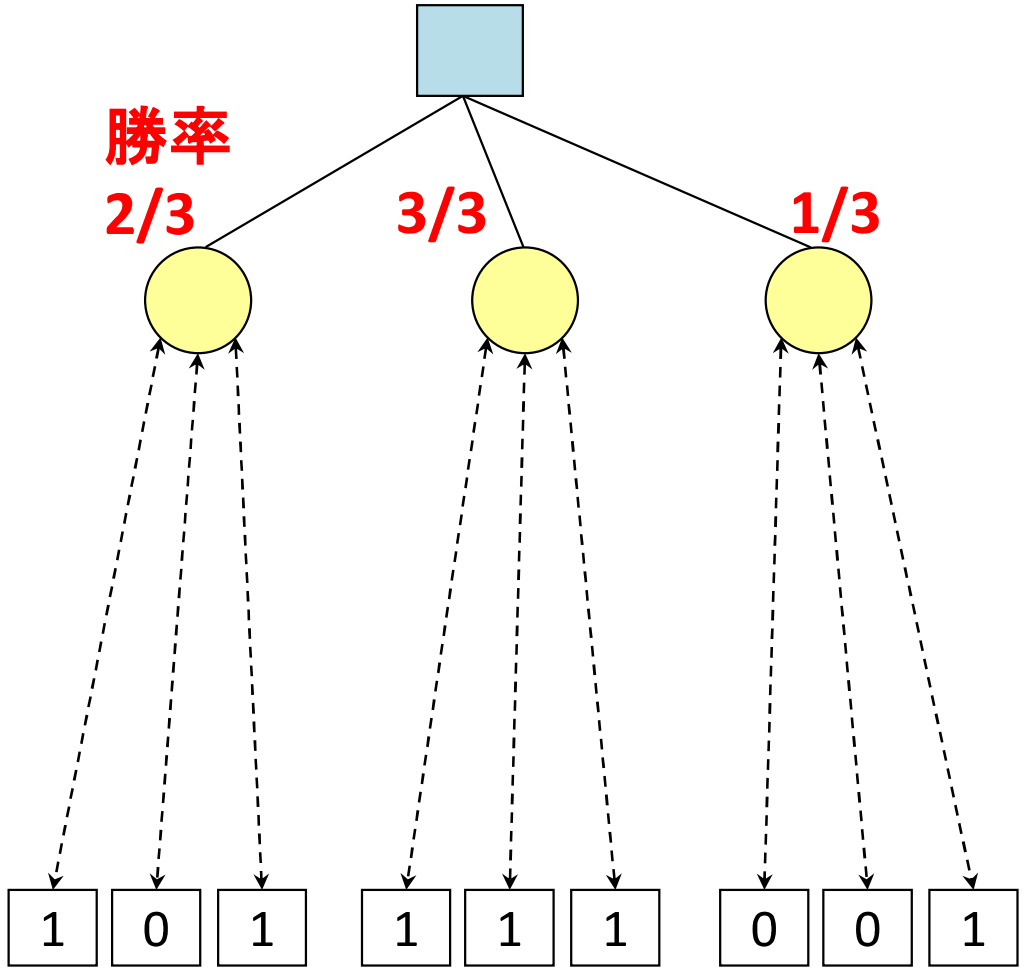
- 円の面積



原始モンテカルロ法

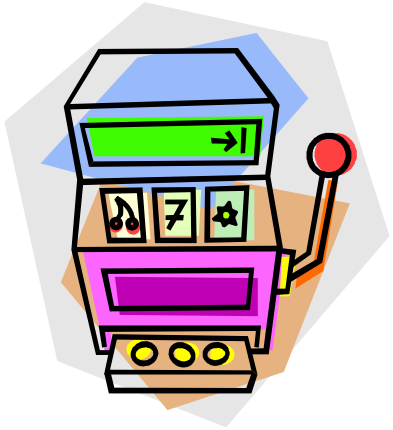
- 各合法手からランダムプレイ
  - 評価関数不要

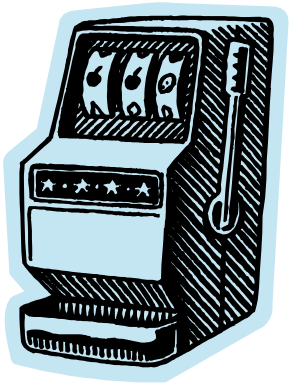
- 勝率の一番高い手を選ぶ
- ダメな手に対しも多くの試行を行うので効率が悪い



多腕バンディット問題

- 多腕バンディット問題（multi-armed bandits）



- どのスロットマシンにお金をつぎこむべきか？
  - 儲かるマシンに集中したい（exploitation）
  - 儲かるマシンを見つけたい（exploration）

UCB

- Upper Confidence Bounds (UCBs)
  - 多腕バンディット問題の近似解法
  - Regret が O(ln *n*)

$$\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

総試行回数

腕 *j* の平均報酬

腕 *j* の試行回数

利用（exploitation）

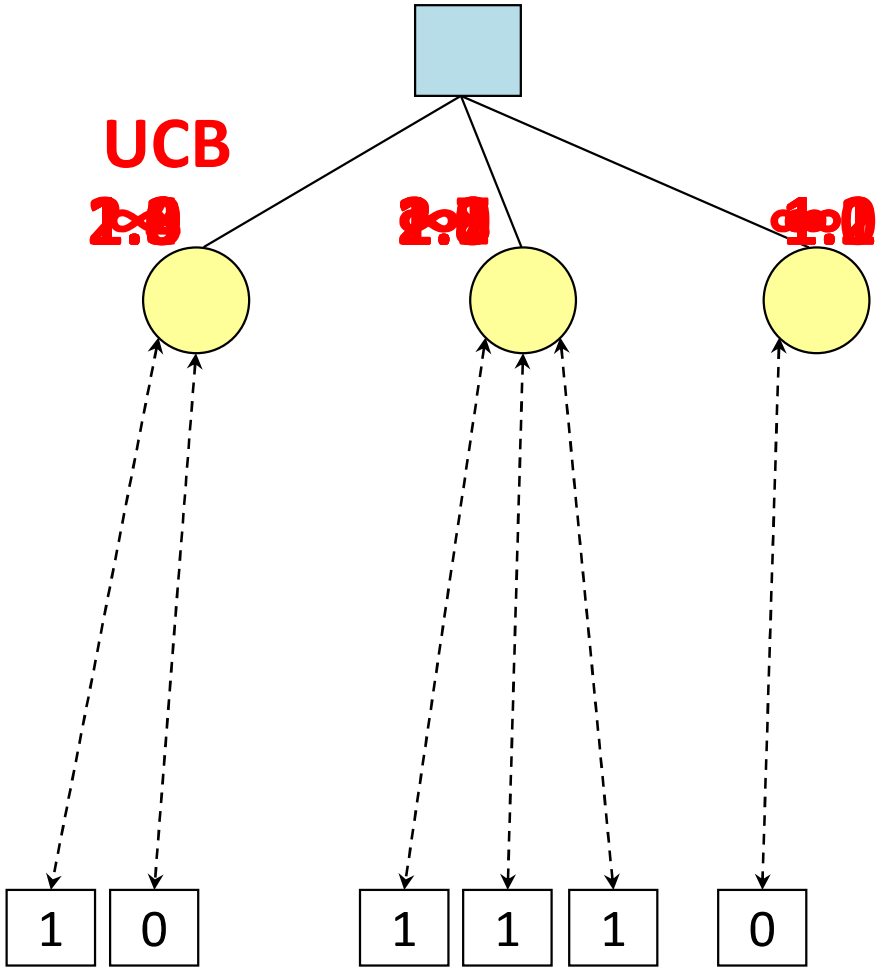
探索（exploration）

UCB 例

- 各イテレーションで UCB 値が最も高い手を選ぶ

$$\mathrm{UCB1} = \overline{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

- 有望な手に関して多くの試行



UCT

- UCB の問題
  - ２手目以降のプレイアウトに無駄が多い
  - 相手の悪手に期待する

- UCT (UCB applied to Trees)
  - Kocsis & Szepesvari (ECML 2006)
  - UCB を各ノードで適用
  - 勝率等を各ノードに保存した木を成長させる
  - MINMAX値に収束

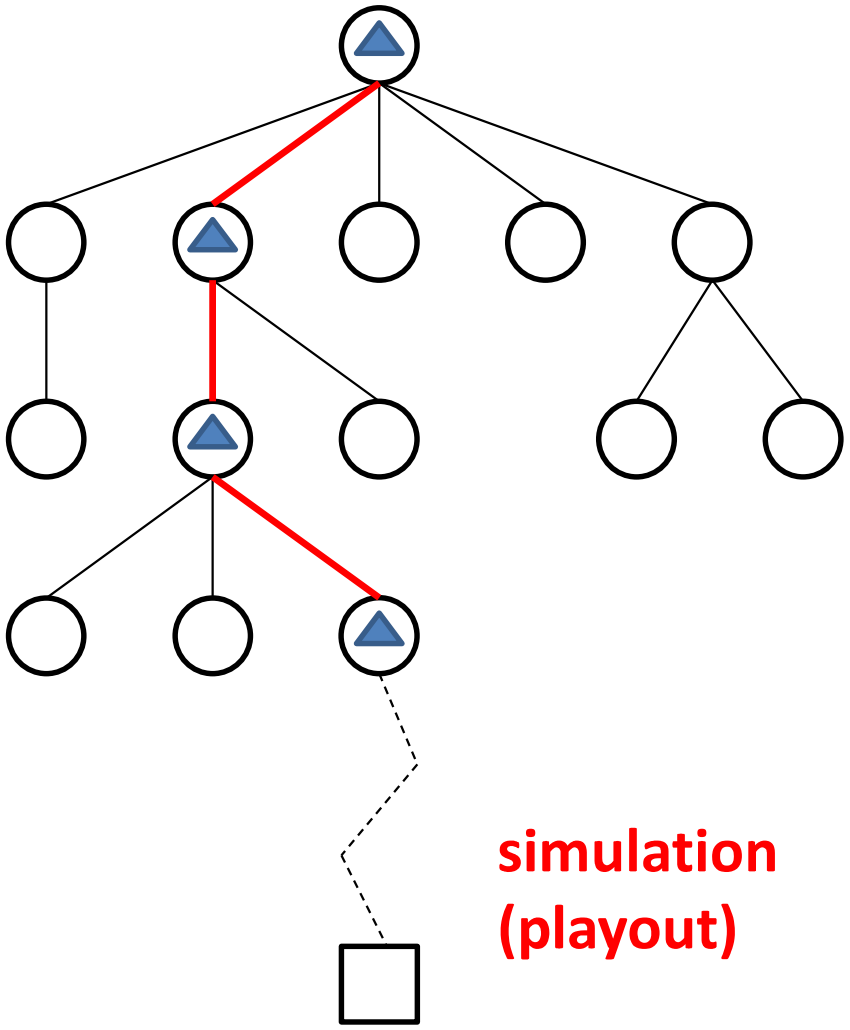
MCTSの基本動作

- 各イテレーション

1. Selection
2. Expansion
3. Simulation
4. Backpropagation

- UCT 値が最大の子ノードを再帰的に選択

$$\mathrm{UCT} = \overline{X}_j + 2C_p\sqrt{\frac{2\ln n}{n_j}}$$

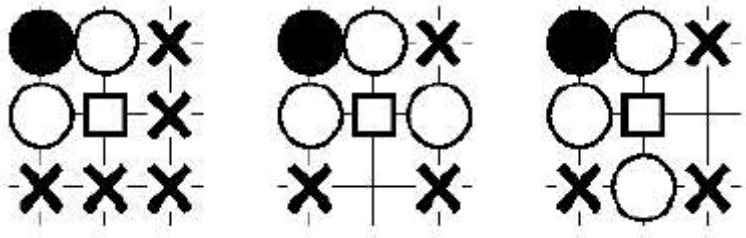


MCTSの改良（主に囲碁）

- Tree policy
  - Progressive Widening
    - 各ノードで考慮する合法手を徐々に増やす
  - All Moves As First (AMAF)
    - プレイアウト中の手の統計情報を木にも反映
  - Rapid Action Value Estimation (RAVE)
    - AMAF の重みの自動調整

Playout policy の改善

- 棋譜による手法
  - Log-linear model + 局所パターン等の特徴量



- Simulation balancing
  - Silver and Tessauro (ICML 2009)
  - プレイアウトによる期待値が教師値と等しくなるように policy のパラメータを調整

References

- Coulom, Efficient Selectivity and Backup Operations in Monte-Carlo Tree Search, CG 2006
- Kocsis & Szepesvari, Bandit based Monte-Carlo Planning, ECML 2006
- Gelly et al., Modification of UCT with patterns in Monte-Carlo Go, TechReport, 2006
- Coulom, Computing Elo Ratings of Move Patterns in the Game of Go, 2007
- Silver and Tessauro, Monte-Carlo Simulation Balancing, ICML 2009
- Browne et al., A Survey of Monte Carlo Tree Search Methods, IEEE Trans. Comput. Intell. AI Games, 2012

C++ 超入門

- C++
  - C言語を拡張
  - オブジェクト指向プログラミング
  - 充実した標準ライブラリ
  - ：
- 使い方
  - コンパイル：gcc のかわりに g++
  - ソースファイルの拡張子は .cc や .cpp などにすることが多い

メンバ関数

- 構造体(クラス)のメンバとして関数を書ける
- クラスを単位として機能をまとめて記述できるので、プログラムの保守性・可読性が高くなる

```
struct student
{
  int id;
  int age;
  void increment_age() {
    age++;
  }
};

int main()
{
  struct student x = {123, 20};

  x.increment_age();
}
```

標準ライブラリ

- 便利なクラスがいろいろ
- string クラス
  - 文字列を普通の変数のように扱える
  - 代入、結合なども簡単にできる
  - メモリの確保、解放も意識しなくてよい

```
#include <iostream>
#include <string>
#include <cstdio>

using namespace std;

int main()
{
  string x = "abc";
  string y = x + "def";
  string z = y.substr(0, 4);

  cout << z << endl;
  printf("%s¥n", z.c_str());
}
```

実行結果

```
$ ./a.out
abcd
abcd
```

標準ライブラリ

- list クラス
  - 双方向リストの実装
- 他にも多数のコンテナ
  - vector: 可変長配列
  - map: 連想配列
  - set: 集合
  - :

```
#include <iostream>
#include <list>

using namespace std;

int main()
{
  list<int> x;
  x.push_back(1);
  x.push_back(2);
  x.push_back(3);

  cout << x.front() << endl;
  cout << x.back() << endl;
}
```

実行結果

```
$ ./a.out
1
3
```

標準ライブラリ

- map クラス
  - 連想配列の実装
  - 出し入れの際の計算コストは O(log *n*)
- 連想配列
  - 任意のデータ型をインデックスとして使える配列
  - 「辞書」と呼ばれることもある

```
#include <iostream>
#include <map>

using namespace std;

int main()
{
  map<string, int> x;
  x["abc"] = 13;
  x["xyz"] = 25;

  cout << x["abc"] << endl;
  cout << x["xyz"] << endl;
}
```

実行結果

```
$ ./a.out
13
25
```

課題

1. 対局終了時に勝敗を判定・表示

2. 手をランダムに選択するように修正

3. 黒白双方ともパスしたら終局するように改良

4. 思考エンジンを実装
   - 思考中の最善手・形勢の表示
   - 「待った」機能

研究室内オセロ大会

- 日時
  - 4月9日　15:00～
- リーグ戦
- 思考時間
  - 一手3秒以内