5 五将棋をプレイするゲーム AI の強化

MI/CS 実験第二 3a
第 3 回 (後半)

December 4, 2023

今後の流れ

1. 簡単なプレイヤの作成
2. 評価関数の説明
3. (4 回目) ゲーム木探索の説明

指し手の集合

search.h から一部抜粋

```
struct RootMove
{
  // pv[0] には、コンストラクタの引数で渡された m を設定する。
  explicit RootMove(Move m) : pv(1, m) {}

  // 指し手の読み筋 (principal variation)
  std::vector<Move> pv;
};
```

RootMove は指し手の読み筋や評価値で構成される構造体

pv[0] に探索開始局面の指し手が渡される

search.h 40 行目あたり

```
typedef std::vector<RootMove> RootMoves;

// 探索開始局面で思考対象とする指し手の集合。
extern RootMoves rootMoves;
```

- RootMoves は RootMove の**可変長配列**
- rootMoves は RootMoves の**変数**
- 探索開始局面 p で合法手が N 個あるとき,
  - rootMoves.size() $= N$
  - p における各合法手は rootMoves[i].pv[0] に格納される ($0 \leq i \leq N-1$)

探索部

```
void Search::search(Position& pos)
{
  // 探索で返す指し手
  Move bestMove = MOVE_RESIGN;

  if (rootMoves.size() == 0)
  {
    // 合法手が存在しない
    Stop = true;
    goto END;
  }
  // 中略 <--- ここの部分に探索部を書く
END:;
    cout << "bestmove " << bestMove << endl;
}
```

ランダムプレイヤを作る

合法手をランダムに選択して返すプレイヤ

```
void Search::search(Position& pos)
{
  /* ここから探索部を記述する */
  {
    size_t i = rand() % rootMoves.size();
    bestMove = rootMoves[i].pv[0];
  }
  /* 探索部ここまで */
}
```

※ 探索部の記述以外は省略

- { } で囲まれた範囲を**ブロック**という
  変数の可視範囲と寿命を定める
- rand() は C 言語の標準関数ライブラリ関数．擬似乱数を返す

ランダムプレイヤ作成後，以下のコマンドを入力して動作を確かめる (コマンドの説明は次ページ参照)

```
isready
go
position startpos moves 5d5c
go
```

- position コマンド
  エンジンに思考開始局面を送る
  形式：position [局面] moves <move1> <move2> ...
    局面 以下のどちらかで局面を指定する
      sfen <SFEN 文字列> (任意局面)
      startpos (初期局面)
- go コマンド
  エンジンの思考を開始する
  形式：go byoyomi <秒読み時間 (ms)> (など)
  ※ go コマンド入力後，局面は更新されない

プレイヤとの対局方法

※ Windows のみ動作確認

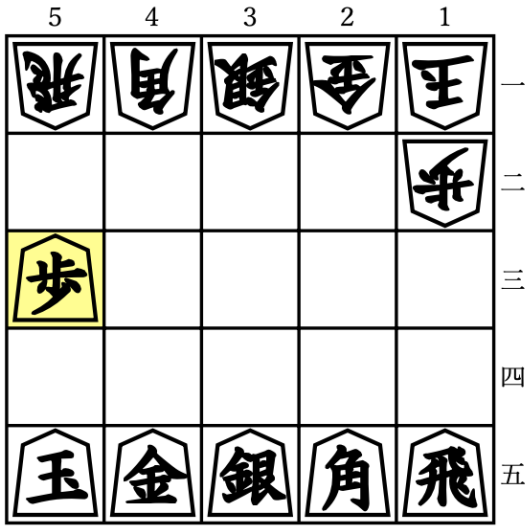
1. プチ将棋をダウンロードして展開する
2. PetitShogi.exe をダブルクリックする
3. [ツール]→[開始編集局面] の順にクリックし,「5 五」の升目をクリックする
4. [ツール]→[エンジン管理] の順にクリックし，`mics2_2d/build` 以下にある minishogi.exe を登録する
5. [対局]→[開始] の順にクリックし，対局者の一方を人間，他方をエンジンとして思考時間を適当に設定し [開始] をクリックする

局面を進める

```
#include "../position.h"
void user_test(Position& pos, std::istringstream& is)
{
  StateInfo si;
  ExtMove m = MoveList<LEGAL_ALL>(pos).at(0); // 合法手の 0 番目
  pos.do_move(m.move, si);                    // 1 手進める
  std::cout << m.move << std::endl; // 指し手を出力
  std::cout << pos << std::endl;    // 局面を出力
}
```

- StateInfo は局面情報を持つ構造体
- do_move() は指し手で局面を 1 手進める関数
  引数には Move と StateInfo が必要

`isready` コマンド後に `user` コマンドを入力すると,
初期局面で 5 四の歩が 5 三に移動する



局面を戻す

```
#include "../position.h"
void user_test(Position& pos, std::istringstream& is)
{
  StateInfo si;
  ExtMove m = MoveList<LEGAL_ALL>(pos).at(0); // 合法手の 0 番目
  pos.do_move(m.move, si);                    // 1 手進める
  std::cout << m.move << std::endl; // 指し手を出力
  std::cout << pos << std::endl;    // 局面を出力

  pos.undo_move(m.move);                  // 1 手戻す
  std::cout << pos << std::endl; // 局面を出力
}
```

- undo_move() は指し手で局面を 1 手戻す関数
  引数には Move が必要

`isready` コマンド後に `user` コマンドを入力すると，
初期局面で 5 四の歩が 5 三に移動したのち，**初期局面に戻る**

評価関数

局面の有利不利を近似的に計算する関数

`evaluate.cpp` の evaluate() が評価関数本体

サンプルコードでは**駒の損得**のみで評価値を計算している

※ **手番側から見た評価値**を計算する

- 評価値がプラスのとき：手番側が有利
- 評価値がマイナスのとき：手番側が不利

駒の価値 (点数) は `evaluate.h` で定義されている

歩	銀	金	角	飛
100	640	690	890	1040

と金	成銀	馬	龍	玉
420	670	1150	1300	15000

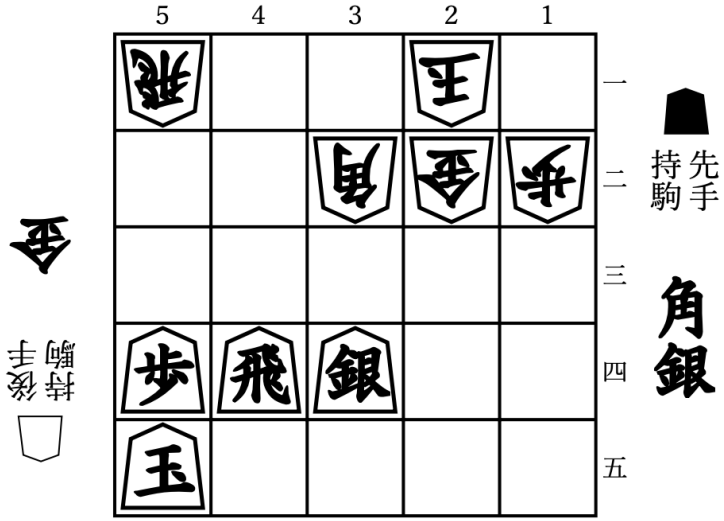
※ **玉の点数**は考慮しなくてもよい

(∵ 玉が互いに 1 枚ずつの局面を考えればよい)

駒の損得は

手番側の駒の価値の総和 − 非手番側の駒の価値の総和

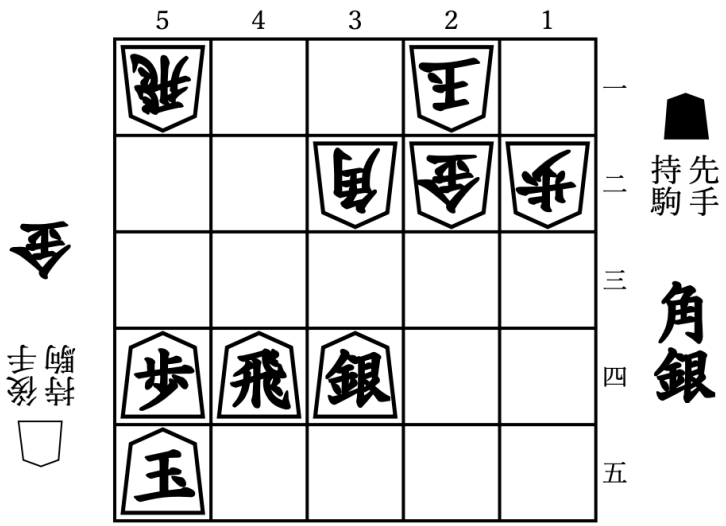
で計算される



先手番の局面

- SFEN 文字列は

- 駒の損得は



先手番の局面

- SFEN 文字列は
  r2k1/2bgp/5/PRS2/K4 b BSg 1
  (手数の自然数は任意でよい)
- 駒の損得は
  先手が 100+640x2+890+1040=3310
  後手が 100+690x2+890+1040=3410
  3370–3410=–100

`eval` コマンドを入力すると関数 evaluate() を呼び出す

- 以下のコマンドを入力して動作を確かめる

```
isready
d
eval
position sfen r2k1/2bgp/5/PRS2/K4 b BSg 1
d
eval
```

```
Value evaluate(const Position& pos) // 評価関数本体
{
  Value score = VALUE_ZERO; // 評価値の初期値 0

  // 盤上の駒
  for (Square sq : SQ)
    score += PieceValue[pos.piece_on(sq)];

  // 持ち駒
  for (Color c : COLOR)
    for (Piece pc : { PAWN,SILVER,BISHOP,ROOK,GOLD })
    {
      int cnt = hand_count(pos.hand_of(c), pc); // 駒の枚数
      score += (c == BLACK ? 1 : -1) * Value(cnt * PieceValue[pc]);
    }

  // 手番側から見た評価値を返す
  return pos.side_to_move() == BLACK ? score : -score;
}
```

1手読みプレイヤを作る

参考：連載やねうら王 mini で遊ぼう！の 11 日目

1. 各合法手で以下を実行する
   - 局面を 1 手指し進めて評価値を計算する
2. (手番側から見て) 最も評価値が高い指し手を選択する

```
void Search::search(Position& pos)
{
  /* ここから探索部を記述する */ {
    Value maxValue = -VALUE_INFINITE; // 初期値はマイナス∞
    StateInfo si;
    for (int i = 0; i < rootMoves.size(); ++i) {
      Move move = rootMoves[i].pv[0]; // 合法手の i 番目．スライド 4 枚目参照
      pos.do_move(move, si);          // 局面を 1 手進める
      Value value = (-1) * Eval::evaluate(pos); // 評価関数を呼び出す
      pos.undo_move(move);            // 局面を 1 手戻す
      if (value > maxValue) {
        maxValue = value;
        bestMove = move;
      }
    }
  } /* 探索部ここまで */
}
```

※ 探索部の記述以外は省略

実装の注意点

`Value value = (-1) * Eval::evaluate(pos);` について

現在の局面が先手番のとき

- 1 手指し進めた局面は後手番

→ 評価関数は後手 (**手番側**) から見た評価値を計算する

→ 先手から見た評価値が欲しいときは符号を**反転**すればよい

参考文献

- C++入門(かいてい.net)
- cpprefjp - C++日本語リファレンス
- やねうら王オープンソースプロジェクト(やねうら王開発者による解説記事)
- SFEN 文字列について(Qiita)