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 4 回 (3)

December 9, 2023

ネガマックス法の実装

search.cpp に以下の関数 search を定義する

```
// プロトタイプ宣言
Value search(Position& pos, int depth, int ply_from_root);
```

pos	局面．構造体 Position の参照渡し
depth	探索深さ
ply_from_root	思考開始局面からの手数

以降ではコードの一部を抜粋.

(1) 1 手読みプレイヤ作成時の探索部を改変する

```
/* 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;
  int rootDepth = 5; // 探索深さ．ここでは適当に 5 とした
  for (int i = 0; i < rootMoves.size(); ++i)
  {
    Move move = rootMoves[i].pv[0];
    pos.do_move(move, si); // 局面を 1 手進める
    Value value = -search(pos, rootDepth - 1, 0);
    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 */
```

(2) 新しく定義した関数 search の実装

```
Value search(Position& pos, int depth, int ply_from_root)
{
  if (depth <= 0)
    return Eval::evaluate(pos); // 深さ 0 では評価関数を呼ぶ

  Value maxValue = -VALUE_INFINITE; // 初期値はマイナス∞
  StateInfo si;
  int moveCount = 0; // この局面で do_move() された合法手の数
  /* 中略 */
  /* ... */
  for (ExtMove m : MoveList<LEGAL_ALL>(pos)) {
    pos.do_move(m.move, si);
    ++moveCount;
    Value value = -search(pos, depth - 1, ply_from_root + 1); // 再帰的に search() を呼び出す
    pos.undo_move(m.move);
    if (value > maxValue)
      maxValue = value;
  }
  if (moveCount == 0)
    return mated_in(ply_from_root); // 詰みの評価値を返す

  return maxValue; // 最も良い評価値を返す
}
```

- 探索深さの値 `rootDepth` を変更して，指し手がどのように変わるか調べる
- プチ将棋を使って作成したプレイヤと対局してみる
  かなり手強くなっているはず
  - Windows のみ動作確認
  - 使い方は第 3 回後半スライドの 9 枚目参照

アルファ・ベータ法の実装

後ろ向き枝刈り (**アルファベータ法**) を実装する

`search.cpp` に以下の関数 `search` を定義する

- 新たに引数を 2 つ加える

```
// プロトタイプ宣言
Value search(Position& pos, Value alpha, Value beta,
  int depth, int ply_from_root);
```

`alpha` α 値．探索範囲の**下限**

`beta` β 値．探索範囲の**上限**

関数のオーバーライド

C++では引数が異なる同名の関数を複数定義することができる

```
// 引数が異なる同名の関数を複数定義できる
Value search(Position& pos, int depth, int ply_from_root);
Value search(Position& pos, Value alpha, Value beta,
  int depth, int ply_from_root);
```

両方の実装を残しておくことができる (消してもよい)

探索の延長

1 手読みプレイヤ

- 局面を 1 手だけ指し進めて最も評価値の高い手を選択
  = 探索深さ 1 のミニマックスを行っている
- **弱点**：駒を捨ててくる．駒をとる指し手があると
  (1 手しか読めていないため) その手が最も良いと判断する

1 手読みプレイヤの改良

- 駒がとる指し手があるときはさらに先の局面を読む
  なるべく**局面が安定している** (駒がぶつかっていない) ときに評価関数を呼び出したい

この改良案は 1 手読みプレイヤでなくても採り入れることができる

このような探索末端の評価を安定させる手法を**静止探索**と呼ぶ

静止探索の実装

※ 次回詳しく説明

実装前　通常探索の末端 (探索深さ 0) で評価関数を呼び出す

実装後　通常探索の末端で静止探索を呼ぶ.
静止探索では**指し手を限定したミニマックス**を行い,
この中で評価関数を呼び出す

実装例 (決まった実装方法はない)

- 王手がかかっていないときは，**駒をとる指し手のみ**生成
- 王手がかかっているときは，通常探索と同じ
(∵ 王手を回避する手しか選べない)

参考文献

- 小谷善行，岸本章宏，芝原一友，鈴木豪「ゲーム計算メカニズム」，コロナ社 (2010)
- 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)