パズルで鍛える

アルゴリズム力

第4回

4×4盤オセロの完全解析をとおして、ゲーム探索を学ぶ

これまで3回にわたって、さまざまなグラフ探索アルゴリズムを取り上げてきました。今回もグラフ探索の応用としてゲーム探索を取り上げます。具体例として4×4盤オセロの解析を行います。

Author けんちょん(大槻 兼資、Kensuke Otsuki)　株式会社NTTデータ数理システム　E-mail dr.ken.1215@gmail.com　Twitter @drken1215

4×4盤オセロ

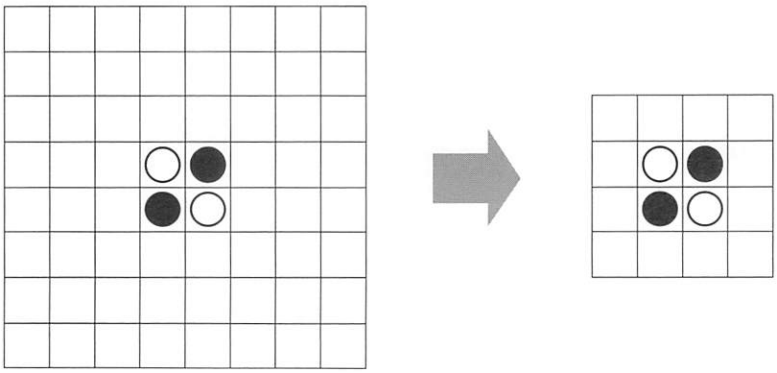
オセロは、2人のプレイヤーが交互に石を打ちながら、相手の石を自分の石で挟むことによって自分の石へと換えていき、最終的な盤上の石の個数を競うボードゲームです。ゲームのルールは単純明快ですが、多数の戦術が生み出され、日々戦略的な進歩を続けています。このことを端的に表した「覚えるのに一分、極めるのに一生」という言葉がキャッチフレーズになっています。

オセロはその複雑さから、現在まで完全解析には至っていません。しかしながら、図1のように盤の大きさを8×8から4×4に縮小したオセロならば、比較的容易に完全解析することができます。本記事では、4×4盤オセロの完全解析を行いつつ、ゲーム探索の一般論を解説していきます。なお、本記事のソースコードはC++で記述していますが、基本的な機能のみを用いているため、ほかの多くの言語でも同様の処理を実現できます。

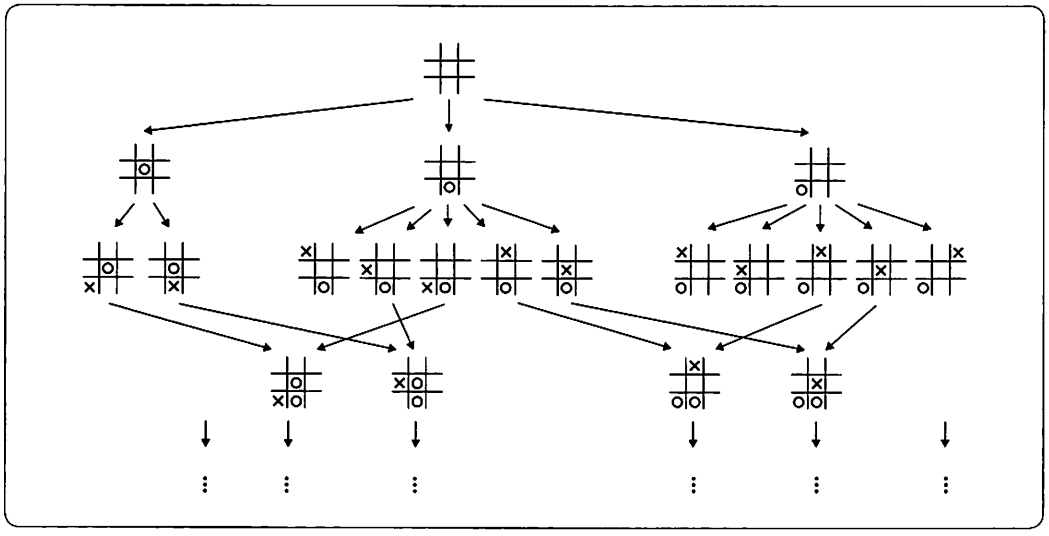
ゲームを完全解析するとは

オセロは、勝敗に運の要素がなく、2人のプレイヤーが互いに知恵を絞って実力だけを頼りに勝敗を決するゲームです。引き分けとなる場合もありますが、一方が勝てば他方は敗北となります。さらに、有限回の手順で必ず勝敗が決まりますし、トランプの大富豪のように互いの手札を隠すこともありません。このようなゲームは「二人零和有限確定完全情報ゲーム」と呼ば

▼図1　4×4盤オセロ



▼図2 マルバツゲームの局面遷移を表すグラフ



れます。オセロだけでなく、チェスや将棋、囲碁もそれに含まれます。二人零和有限確定完全情報ゲームには、次の特徴があります。

- 2人のプレイヤーが互いに最善を尽くした場合において、「先手必勝」「後手必勝」「引き分け」のいずれになるかが原理的にはあらかじめ決まっている
- 無尽蔵の計算時間と計算機資源があるならば、必勝手順を解析することができる

つまり、オセロもチェスも将棋も囲碁も、理論上はすべての局面について完全な先読みができます[注1]。たとえばマルバツゲーム(三目並べ)は、その局面遷移を図2のように表せます。図2では、回転や裏返しによって重なる局面は同一のものとみなしていることに注意してください。どのゲームも同様に局面遷移グラフを考えることができ、局面遷移グラフを探索することで、必勝手順が求められます。

このようにゲームが「先手必勝」「後手必勝」「引き分け」のいずれになるかを求め、必勝手順を見いだすことを完全解析すると言います[注2]。

各ゲームの解析状況

本記事では4×4盤オセロの完全解析を行います。それに先立って、伝統的な各ゲームの完全解析状況を表1にまとめます。たとえば4×4盤オセロ、6×6盤オセロは後手必勝であることがわかっています。また連珠は、通常の五目並べにおいて先手の着手に制限を加えたものですが、それでも最短47手で先手が勝てることが示されました。

将棋は完全解析に至っていませんが、将棋を簡略化したゲームであるどうぶつしょうぎ[注3]は後手必勝(78手)であることがわかっています。なお、将棋やどうぶつしょうぎでは、同一局面が何度も繰り返し現れることがあります。その

注1) 厳密には、将棋には現在「勝ちか負けか引き分けかがルール上あいまいな局面」が存在することがわかっています。「最後の審判」と呼ばれる詰将棋作品で示されています。

注2) ゲームの解析にも段階があります。初期盤面の勝敗のみがわかることを超弱解決と呼び、それを証明するのに必要な各局面の最善手もわかることを弱解決と呼び、すべての局面について勝敗も最善手もわかることを強解決と呼んで区別することもあります。本記事では弱解決までを考えます。

注3) どうぶつしょうぎは、将棋を簡略化したルールを北尾まどか女流棋士が考案し、藤田麻衣子女流棋士がデザインすることで2008年に生まれました。使用する駒は「ライオン」「ぞう」「きりん」「ひよこ」の4種類であり、3×4の盤面上でプレイします。

▼表1　さまざまなゲームの解析状況

ゲーム	結論	備考	参考文献
マルバツゲーム	引き分け		
4×4盤オセロ	後手必勝	後手の10石勝ち	
6×6盤オセロ	後手必勝	後手の4石勝ち	J. Feinstein: Amenor Wins World 6x6 Championships!, Forty billion noted under the tree, pp.6-8, British OthelloFederation's newsletter. (1993)
8×8盤オセロ	未解決		
連珠	先手必勝	47手で先手の勝ち	J. Wagner and I. Virag: Solving renju, ICGA Journal, Vol.24, No.1, pp.30-35 (2001)
チェッカー	引き分け	完全解析に約18年もの計算を要したと言われている	M. Muller, R. Lake, P. Lu, and S. Suphen: Checkers is solved, Science Vol.317, No,5844, pp.1518-1522 (2007)
どうぶつしょうぎ	後手必勝	78手で後手の勝ち	田中哲郎「『どうぶつしょうぎ』の完全解析」『情報処理学会研究報告』Vol.2009-GI-22 No.3、pp.1-8、2009年
将棋	未解決		
4×4盤囲碁	引き分け		清槙一、川嶋俊「探索プログラムによる四路盤囲碁の解」『研究報告ゲーム情報学(GI)』Vol. 2000-GI-004、pp.69-76、2000年
5×5盤囲碁	先手必勝	先手の24目勝ち	E. van der Werf, J. van den Herik, and J. Uiterwijk: Solving Go on Small Boards, ICGA Journal, Vol.26, No.2, pp.92-107 (2003)
囲碁	未解決		

ような場合は「引き分け」と考えます[注4]。このように局面がループする可能性のあるゲームでは「後退解析」と呼ばれる手法がよく用いられます。

なお、本記事で扱うオセロでは、「局面が進むと盤上の駒数が単調増加する」というゲームの性質上、局面がループすることはありません。後退解析について関心のある方は、たとえばどうぶつしょうぎの完全解析を実施した論文(表1中の参考文献)を読んでみてください。

ゲーム探索

ゲーム探索の考え方

ゲーム探索の考え方を体感するために、簡単なゲーム(石取りゲーム)を実際に解析してみましょう。石取りゲームとは、最初に何個かの石が積まれた山から、先手と後手が交互に1、2、3個のいずれかの個数の石を取っていくゲームです。先に石が取れなくなったほうが負け(最後の石を取ったほうが勝ち)です。先手と後手が互いに最善を尽くした場合、どちらが勝つでしょうか？　結論は次のようになります。

・石の個数が4の倍数のとき：後手必勝
・石の個数が4の倍数ではないとき：先手必勝

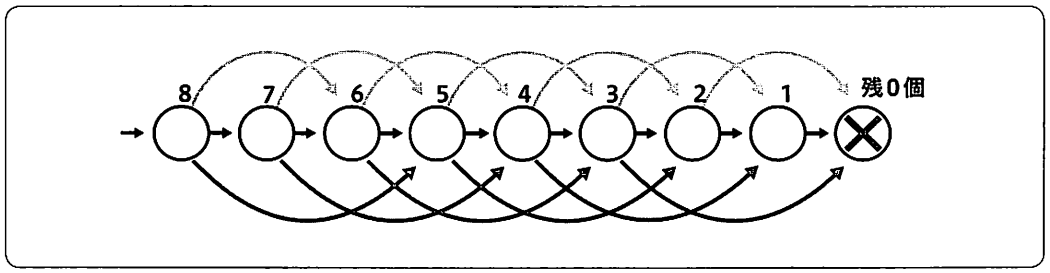
たとえば初期状態で22個の石があったとしましょう。このとき先手はまず2個の石を取ることで石の個数を20個にすることができます(20は4の倍数です)。それ以降は次のようにします。

・後手が1個の石を取ってきたならば、その次のターンで先手は3個の石を取る
・後手が2個の石を取ってきたならば、その次のターンで先手は2個の石を取る
・後手が3個の石を取ってきたならば、その次のターンで先手は1個の石を取る

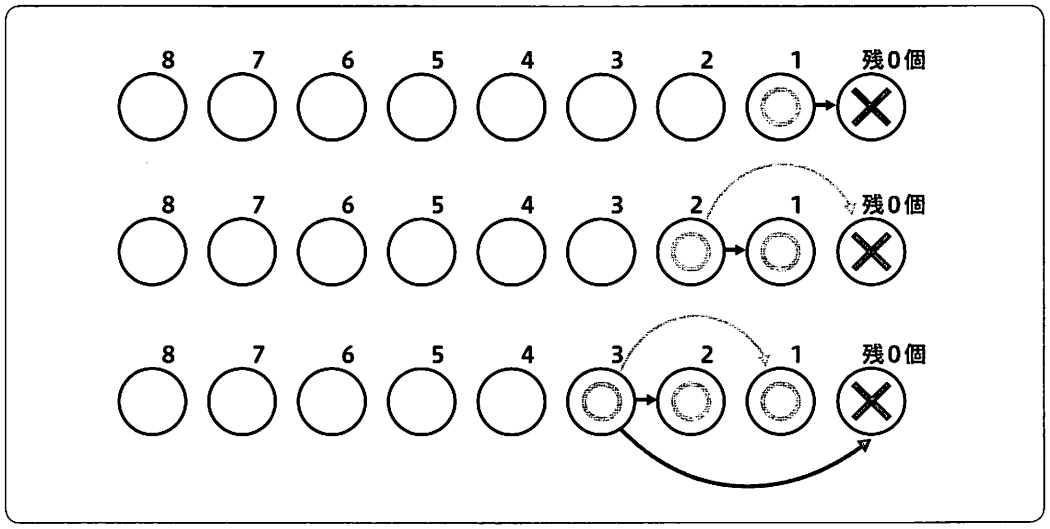
これにより、先手は常に石の個数を4個ずつ減らしながら後手にターンを渡すことができます。最後は石が4個の状態で後手にターンを回して、後手がどのように応対したとしても残った石をすべて取って勝てます。まとめると、「石の個数を4の倍数にして相手に渡せたら勝ち」「石の個数が4の倍数で手が回ってきたら負け」

注4) 将棋の公式ルールでは、同一局面が4回以上現れた場合は先後を入れ替えて指しなおす(千日手)という規定が設けられています。

▼図3 石取りゲームの局面遷移を表すグラフ



▼図4 石の個数が1、2、3の場合は「勝ち」



と言えます。

以上の結論を、グラフ探索に基づいて導いてみましょう。石取りゲームの局面遷移は図3のように表せます。右端が終局(石が残り0個の状態)を表します。それ以外のノードに記した数値は石の個数を表しています。たとえば8と記したノードからは、7、6、5と記したノードに遷移できます(これ以降、石の個数がNの状態を表すノードをノードNと呼ぶことにします)。石が0個の状態で手番となったら負けですので、ノード0は「負け」となります(図3では「×」を記入しています)。

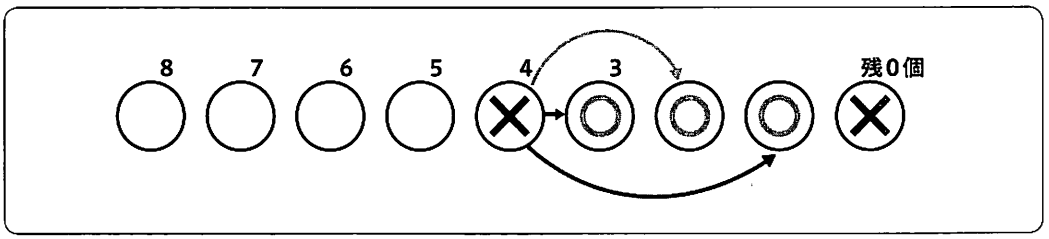
ゲーム解析では「勝敗が決している局面を出発点として、さかのぼるように解析する」という考え方が基本となります。たとえばノード1、2、3は、いずれも「負け」であるノード0の状態にして相手に手番を渡せますので、「勝ち」となります(図4ではそれぞれ「○」を記入しています)。一方ノード4は、そこから遷移できるノード1、2、3はすべて「勝ち」ですので、どの手を打っても相手が勝ちとなります。つまりノード4は「負け」となります(図5)。ノード5は、今度は「負け」であるノード4へと遷移して相手に手番を渡せますので、「勝ち」となります(図6)。

したがって、次のように考えることができます。

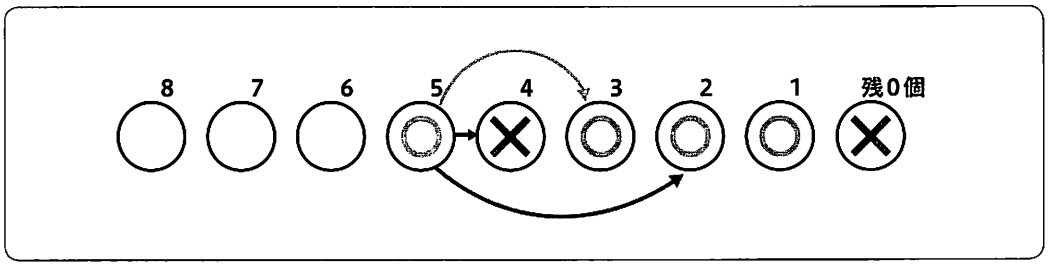
- 今のノードから遷移できるノードのうち、「負け」が1つでも存在するならば、今のノードは「勝ち」
- 今のノードから遷移できるすべてノードが「勝ち」ならば、今のノードは「負け」

以上の考え方に基づいてゲーム探索を続ける

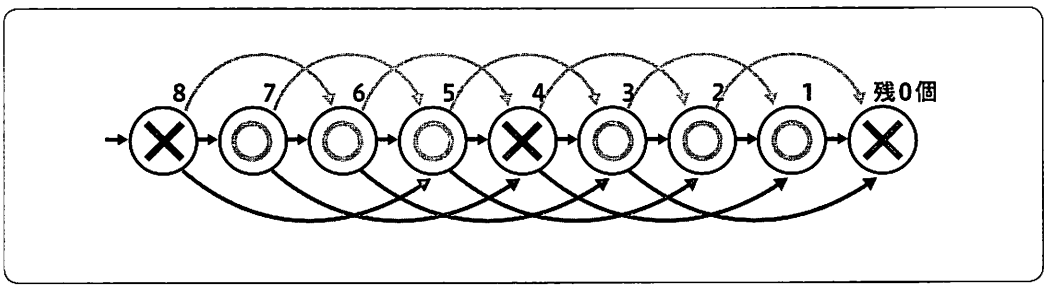
▼図5　石の個数が4の場合は「負け」



▼図6　石の個数が5の場合は「勝ち」



▼図7　石取りゲームの解析結果



と、各局面の勝敗は図7のように求められます。

ゲーム探索の実装

ゲーム探索は、再帰関数を用いることでリスト1のように実装できます[注5]。ここでは、盤面の状態や手番に関する情報が、クラスNodeとして表現されている状況を想定しています。また、勝敗のみではなくスコアも考慮した実装を行っています[注6]。これにより、たとえばオセロの場合は、勝敗だけではなく石の個数の差も最大にする最善手が求められます。

なお、勝敗のみを考える場合には、次のように修正します。

・「勝ち」を表すスコア：1
・「負け」を表すスコア：－1

置換表

図2のマルバツゲームの局面遷移グラフを振り返ると、異なる局面から同一の局面へと合流する箇所があることが見て取れます。このような局面遷移グラフに対しては、とくに工夫せず

注5）リスト1の実装はネガマックス法と呼ばれるものとなっています。
注6）相手にとってN石差で勝ち(スコアN)であるとき、自分にとってはN石差で負け(スコア－N)となります。逆に、相手にとって－N石差で負け(スコア－(－N))であるとき、自分にとってはN石差で勝ち(スコアN)となります。このように、自分側のスコアは相手側のスコアを符号反転したものと考えます。

にゲーム探索を行うと、同一局面を何度も解析してしまう可能性があります。それを効率化するためには、一度解析した局面のスコアをメモ化する手法が考えられます。これをゲーム探索の文脈では「置換表」と呼びます[注7]。ただし本記事では簡単のため、置換表については省略します。

α-β法

ゲーム探索を効率化する代表的手法としてα-β法と呼ばれる枝刈り手法があります。「こちら側に絶妙手があるような局面を相手は渡してこない」という考え方に基づいてゲーム探索を効率化するものです。この考え方を具体的に実現してみましょう。

▼リスト1　ゲーム探索の実装の一般形

```
int rec(Node &node) {
    // 終局の場合は石差を数える
    if (nodeが終局) return (nodeのスコア)

    // 各遷移を考える
    int result = -INF; // 無限小を表す値
    for (Node node2 : (nodeの遷移先)) {
        // 遷移局面のスコアを符号反転して受け取る
        int score = -rec(node2);
        result = std::max(result, score);
    }
    return result;
}
```

▼リスト2　α-β法

```
int rec(int alpha, int beta, Node &node) {
    // 終局の場合は石差を数える
    if (nodeが終局) return (nodeのスコア)

    // 各遷移を考える
    for (Node node2 : (nodeの遷移先)) {
        // 遷移局面のスコアを符号反転して受け取る
        int score = -rec(-beta, -alpha, node2);
        alpha = std::max(alpha, score);

        // βカット
        if (alpha >= beta) return alpha;
    }
    return alpha;
}
```

まず、相手がいくつかの手を検討済みであり、すでにβ以上のスコアを獲得可能だとわかっているものとします。その状態で、仮に相手がnodeという局面をこちらに渡してきたとします。そして局面nodeに良い手が存在して、こちらのスコアをα以上にできることがわかったとします。このとき$\alpha \geqq \beta$であるならば、相手がそもそも局面nodeをこちらに渡してくることはありません。よって、$\alpha \geqq \beta$となった時点で局面nodeにおける探索を打ち切ることができます。このような枝刈り(βカット)を取り入れた探索法をα-β法と言い、**リスト2**のように実装できます。

4×4盤オセロの解析

以上で解説したゲーム探索を4×4盤オセロに対して適用します。まずオセロの各局面の状態を、

- 黒石の配置を表す整数bl
- 白石の配置を表す整数wh
- 手番の先後を表す変数co(黒番を1、白番を0とする)

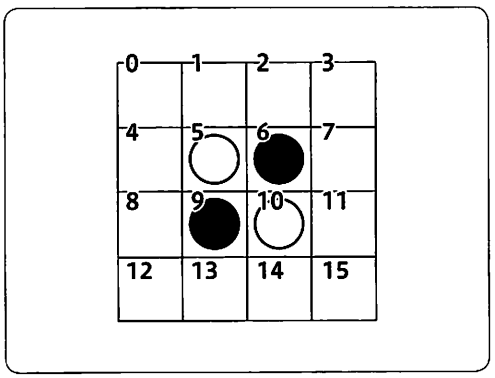
によって表現します。

ここで、**図8**のようにオセロ盤の各マスを番号付けします。黒石の配置を表す整数blは、二進法表現したときに、黒石のあるマスの番号に対応する桁の値が1でほかの桁の値が0である整数を表します。白石の配置を表す整数whも同様です。4×4のオセロ盤において、石を置くことによる局面遷移の様子や、最終局面のスコア計算は**リスト3**のように実装できます[注8]。

これを用いて、4×4盤オセロの解析は**リスト4**のように実装できます。ここでは初期局面のスコアを求めています。なお**リスト4**では省

注7) 置換表を用いたゲーム探索は、本誌2020年10月号で解説したダイクストラ法でも登場した動的計画法の一種とみなせます。
注8) ここではロジックのわかりやすさを重視した実装をしていますが、実際のオセロ解析では、よりビット演算を多用した実装方法がよく採られます。それによって探索を高速化できます。

▼図8　4×4盤オセロの各マスの番号



略していますが、各局面に対する最善手をメモ化することにより、具体的な最適手順を復元することもできます。解析結果を図9に示します。4×4盤オセロで双方が最善を尽くした結果は黒3－白11(空きマス2)となり、10石差で後手必勝です。日本オセロ連盟競技ルールに則り、空きマスは勝者のものとカウントしています。計算に要した時間は筆者の環境[注9]で0.050秒でした。

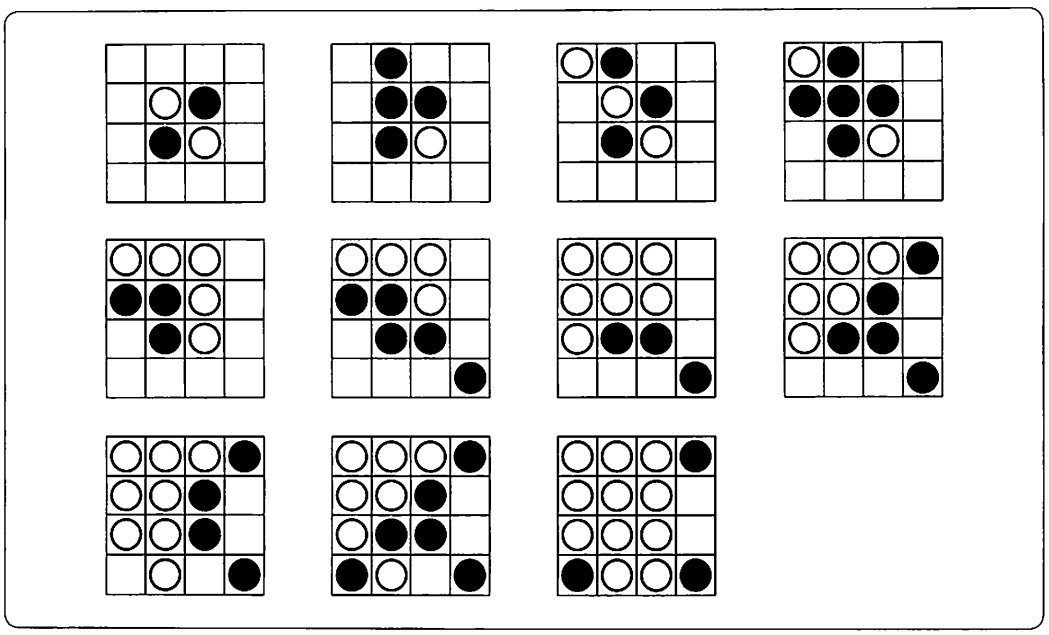
まとめ

今回は、4×4盤オセロの解析をとおして、ゲーム探索の考え方を解説しました。そしてゲーム探索で登場した考え方は、より一般的で汎用性の高い手法をゲームに対して応用したものだとみなすことができます。

- ゲーム探索：「深さ優先探索」の応用例
- 置換表：メモ化を活用する「動的計画法」
- α-β法：「分枝限定法」と呼ばれる枝刈り手法の応用例

つまりゲーム解析を学ぶことで、より汎用的な手法を自然に楽しく学ぶことができます。それはゲームに限らず、さまざまな問題を解くときに役立つものです。本記事をとおして、ゲーム探索のおもしろさを感じていただいたり、汎用手法をさまざまな問題の解決に役立てていただいたりしたならば、大きな喜びです。SD

▼図9　4×4盤オセロの最適手順



注9) MacBook Air(13-inch, Early 2015)プロセッサ：1.6 GHz Intel Core i5、メモリ：8GB

▼リスト3　4×4盤オセロの局面遷移と、最終局面のスコア計算

```
// 色
const int BLACK = 1, WHITE = 0;

// マスceを方向dに動かした結果を求める
std::vector<int> dx = {1, 0, -1, 0, 1, 1, -1, -1};
std::vector<int> dy = {0, 1, 0, -1, 1, -1, -1, 1};
int move(int ce, int d) {
    int x = ce / 4 + dx[d], y = ce % 4 + dy[d];

    // 盤外に出る場合は-1とする
    if (x < 0 || x >= 4 || y < 0 || y >= 4)
        return -1;
    else
        return x * 4 + y;
}

// 盤面が(bl, wh)のとき、マスceの色がcoかどうか
bool iscolor(int bl, int wh, int co, int ce) {
    if (ce == -1) return false;
    if (co == BLACK) return ((bl >> ce) & 1);
    else return ((wh >> ce) & 1);
}

// 盤面が(bl, wh)のとき、マスceに色coの石を置く
// 反転するマスを返す(置けないときは0を返す)
int put(int bl, int wh, int co, int ce) {
    // マスceにすでに石がある場合は0
    if (((bl | wh) >> ce) & 1) return 0;
    int result = 0;
    for (int d = 0; d < 8; ++d) {
        int rev = 0;
        int ce2 = move(ce, d);
        while (iscolor(bl, wh, 1 - co, ce2)) {
            rev |= 1 << ce2;
            ce2 = move(ce2, d);
        }
        if (iscolor(bl, wh, co, ce2))
            result |= rev;
    }
    return result;
}

// 終局時のスコア計算
int calc(int bl, int wh, int co) {
    int nbl = 0, nwh = 0, nem = 0;
    for (int ce = 0; ce < 16; ++ce) {
        if ((bl >> ce) & 1) ++nbl;
        else if ((wh >> ce) & 1) ++nwh;
        else ++nem;
    }
    // 勝利側に空きマスの個数を加算
    if (nbl > nwh) nbl += nem;
    else if (nbl < nwh) nwh += nem;
    // スコア
    if (co == BLACK) return nbl - nwh;
    else return nwh - nbl;
}
```

▼リスト4　4×4盤オセロの解析

```
#include <iostream>
#include <vector>

// 盤面が(bl, wh)、手番がcoである局面のスコア
int rec(int alpha, int beta, int bl, int wh, int co) {
    // 石の置ける場所を求める(mine:自分、opp:相手)
    std::vector<int> mine, opp;
    for (int ce = 0; ce < 16; ++ce) {
        if (put(bl, wh, co, ce))
            mine.push_back(ce);
        if (put(bl, wh, 1 - co, ce))
            opp.push_back(ce);
    }

    // 終局の場合は石差を数える
    if (mine.empty() && opp.empty())
        return calc(bl, wh, co);

    // パスの場合
    if (mine.empty())
        return -rec(-beta, -alpha, bl, wh, 1 - co);

    // 各遷移を考える
    for (auto ce : mine) {
        // 色を反転する
        int rev = put(bl, wh, co, ce);
        int bl2 = bl ^ rev, wh2 = wh ^ rev;

        // ceに置く
        if (co == BLACK) bl2 |= 1 << ce;
        else wh2 |= 1 << ce;

        // 遷移局面のスコアを符号反転して受け取り、値更新
        int score = -rec(-beta, -alpha, bl2, wh2, 1 - co);
        alpha = std::max(alpha, score);

        // βカット
        if (alpha >= beta) return alpha;
    }
    return alpha;
}

int main() {
    // 初期配置
    int bl = (1 << 6) | (1 << 9);
    int wh = (1 << 5) | (1 << 10);

    int score = rec(-16, 16, bl, wh, BLACK);
    std::cout << score << std::endl;
}
```