知能処理学 レポート

36714029 遠藤裕人
(学生番号と氏名が必要)

2026 年 1 月 15 日

提出レポート: rep0

全体的な自己評価／作業時間: S ／ 8 時間
自己評価を S, A, B, C, D から選択する．作業時間は成績に影響しないので正直に書くこと．

評価の理由: 迷路探索アルゴリズムの実装において各処理（ゴール検査選択展開生成）を独立したメソッドに分離することで可読性と保守性を向上させた．また幅優先探索と深さ優先探索の違いを引数の順序変更のみで実現できることを示しアルゴリズムの本質的な理解を深めることができたため．

1 考察

min-max 法の実装を通じて自分の手番では評価値を最大化し、相手の手番では最小化するという交互の探索により、両者が最善手を選ぶ前提での最適解を求めることができる。しかし、探索ノード数が指数的に増加するため、実用的な深さには限界がある。

次に、α-βカット法の導入により、探索効率の大幅な改善を確認した。αカットとβカットにより、最終結果に影響しない部分木を枝刈りすることで、同じ結果を得ながら計算量を削減できた。ただし、枝刈りの効果は探索順序に依存するという課題が残った。

この課題に対し、Move Ordering を導入して各手を 1 手先の静的評価値でソートし、有望な手から探索することで、α値が早期に高くなり、枝刈りが発生しやすくなる。実験では訪問ノード数が約 79 パーセント減され、Move Ordering の有効性が実証された。

また、ネガマックス法への書き換えにより、コードを簡潔化した。評価値の符号反転という単純な操作で、max と min の 2 つの関数を 1 つに統合でき、可読性と保守性が向上した。ゲーム AI の性能は「探索の深さ」「枝刈りの効率」「評価関数の精度」の 3 要素が組み合わさっていること。

2 課題 2-1a

自己評価／作業時間: S 2 時間 23 分

3 課題 1-1

講義で示した min-max 法や α-β カット法の擬似コードと見比べながらレポートにてその実現方法を説明する

3.1 min-max 法

```
float maxSearch(State state, int depth) {
    if (isTerminal(state, depth))
        return this.eval.value(state);
    var v = NEGATIVE_INFINITY;
    for (var move: state.getMoves()) {
        var next = state.perform(move);
        var v0 = minSearch(next, depth + 1);
        v = Math.max(v, v0);
    }
    return v;
}
```

一つ下のノードに対して minSearch を呼び出している。できるだけ低い評価値の中から最大値を選んでいる。

```
float minSearch(State state, int depth) {
    if (isTerminal(state, depth))
        return this.eval.value(state);
    var v = POSITIVE_INFINITY;
    for (var move: state.getMoves()) {
            var next = state.perform(move);
            var v0 = maxSearch(next, depth + 1);
            v = Math.min(v, v0);
    }
    return v;
}
```

一つ下のノードに対して maxSearch を呼び出している。相手が最良の手を選ぶ前提である。

3.2 α-β カット法

```
float maxSearch(State state, float alpha, float beta, int depth) {葉ノード（探索
    の末端）の評価値
  //
    if (isTerminal(state, depth)) {
        float val = this.eval.value(state);
        System.out.println(" ".repeat(depth) + "Leaf " + state + " = " + val);
        return val;
    }
    float v = NEGATIVE_INFINITY;
```

```
    System.out.println(" ".repeat(depth) + "MAX at " + state + " [alpha=" +
        alpha + ", beta=" + beta + "]");
    for (var move: state.getMoves()) {
        var next = state.perform(move);
        float v0 = minSearch(next, alpha, beta, depth + 1);
        v = Math.max(v, v0);
        if (beta <= v0) {
            System.out.println(" ".repeat(depth) + "PRUNED (beta cutoff) at " +
                 state);
            break;
        }
        alpha = Math.max(alpha, v0);
    }
    return v;
}
```

親ノードより子ノードの評価値が良ければ探索を打ち切る。これがβカットである。実質的に親ノードと一つ下の子ノードの評価値だけを観察しており、それ以下のノードの評価値はminSearch内にある。

```
float minSearch(State state, float alpha, float beta, int depth) {
    if (isTerminal(state, depth)) {
        float val = this.eval.value(state);
        System.out.println(" ".repeat(depth) + "Leaf " + state + " = " + val);
        return val;
    }
    float v = POSITIVE_INFINITY;
    System.out.println(" ".repeat(depth) + "MIN at " + state + " [alpha=" +
        alpha + ", beta=" + beta + "]");
    for (var move: state.getMoves()) {
        var next = state.perform(move);
        float v0 = maxSearch(next, alpha, beta, depth + 1);
        v = Math.min(v, v0);
        if (alpha >= v0) {
            System.out.println(" ".repeat(depth) + "PRUNED (alpha cutoff) at "
                + state + " after evaluating " + next);
            break;
        }
        beta = Math.min(beta, v0);
    }
    return v;
}
```

親ノードより子ノードの評価値が悪ければ探索を打ち切る。これがαカットである。

リスト3を利用して図2の評価値を求めよ

MAX at Root [alpha=-Infinity, beta=Infinity] MIN at L1 [alpha=-Infinity, beta=Infinity] Leaf LL1 = -1.0 Leaf LL2 = -31.0 Leaf LL3 = -16.0 MIN at L2 [alpha=-31.0, beta=Infinity] Leaf ML1 = -38.0 PRUNED (alpha cutoff) at L2 after evaluating ML1 MIN at L3 [alpha=-31.0,

beta=Infinity] Leaf RL1 = -9.0 Leaf RL2 = 6.0 AlphaBeta evaluation: -9.0

よって評価値は-9.0 となる。

また ML1 の評価値-38.0 は L2 ノードの評価値-31.0 より小さいためαカットが発生し RL1 RL2 の評価値は計算されていない。

3.3 考察・感想

考察:

感想:

4 課題 2-1b

自己評価／作業時間: S 2 時間 00 分

5 課題 2-1

講義で示したアルゴリズムと比較すること．各クラスの役割をレポートにて説明すること

Eval クラス状態の評価値を計算する役割を持つ。value メソッドは State オブジェクトを受け取り、その状態がゴール状態であれば無限大の評価値を返し、そうでなければ石の数に基づいて評価値を計算する。静的評価関数を採用しており、先を読むことなく現在の状態のみを評価する。講義で示したアルゴリズムと同一である。

Player クラスゲームプレイヤーの抽象クラス。nextMove メソッドは State オブジェクトを受け取り、次に取るべき手を決定して返す。このクラスを継承して具体的な戦略を実装する。また、色や名前を保持するためのフィールドとコンストラクタも含まれている。

RandomPlayer クラス Player クラスを継承し、ランダムに石を取り除く戦略を実装する。nextMove メソッドは State オブジェクトを受け取り、可能な手の中からランダムに一つを選択して返す。

MinMaxPlayer クラス Player クラスを継承し、min-max 法を用いて最適な手を選択する戦略を実装する。nextMove メソッドは State オブジェクトを受け取り、可能な手をすべて評価し、最も高い評価値を持つ手を選択して返す。評価には Eval クラスを使用する。講義と違い深さ制限がデフォルトで設定されていたが、無制限に設定することも可能である。

State クラスゲームの状態を表現する役割を持つ。現在の石の数、現在のプレイヤー、各プレイヤーが獲得した石の数などの情報を保持する。getMoves メソッドは現在の状態から可能な手をリストとして返し、perform メソッドは指定された手を実行した後の新しい状態を返す。

石の個数を変えたり，先行後攻を入れ替えたりして RandomPlayer と MinMaxPlayer を対戦させた結果を報告せよ

石の数が 5 のとき、MinMaxPlayer が 3 つの石を獲得したため勝利した。==== 5 stone(s) ==== 5 -¿ 4 — Random(o) took 1 stone(s). 4 -¿ 1 — MinMax20(x) took 3 stone(s). 1 -¿ 0 — Random(o) took 1 stone(s). Winner: MinMax20(x)

石の数が 20 のとき、MinMaxPlayer が 11 つの石を獲得したため勝利した。==== 20 stone(s) ==== 20 -¿ 18 — Random(o) took 2 stone(s). 18 -¿ 17 — MinMax20(x) took 1 stone(s). 17 -¿ 14 — Random(o) took 3 stone(s). 14 -¿ 13 — MinMax20(x) took 1 stone(s). 13 -¿ 12 — Random(o) took 1 stone(s). 12 -¿ 9 — MinMax20(x) took 3 stone(s). 9 -¿ 8 — Random(o) took 1 stone(s). 8 -¿ 5 — MinMax20(x) took 3 stone(s). 5 -¿ 4 — Random(o) took 1 stone(s). 4 -¿ 1 — MinMax20(x) took 3 stone(s). 1 -¿ 0 — Random(o) took 1 stone(s). Winner: MinMax20(x)

6 課題 2-1c

自己評価／作業時間: A 1 時間 00 分

7 課題 2-1

課題 2-1b で作成した MinMaxPlayer を改変して，α-β カット法を実装せよ

=== 探索統計情報 === 訪問ノード数: 39 枝刈り回数: 8 αカット: 2 βカット: 6
[βカット] 枝刈り発生箇所: 3 理由: v(-1000.0) ¿= β (-1000.0)

8 課題 2-1d

自己評価／作業時間: S 3 時間 16 分

9 課題 2-1

d が無制限の場合に比べて MinMaxPlayer の強さがどのように変化したのか報告せよ.

N=20 より==== 20 stone(s) ==== 20 -¿ 17 — Random(o) took 3 stone(s). 17 -¿ 14 — MinMax20(x) took 3 stone(s). 14 -¿ 11 — Random(o) took 3 stone(s). 11 -¿ 9 — MinMax20(x) took 2 stone(s). 9 -¿ 6 — Random(o) took 3 stone(s). 6 -¿ 5 — MinMax20(x) took 1 stone(s). 5 -¿ 4 — Random(o) took 1 stone(s). 4 -¿ 1 — MinMax20(x) took 3 stone(s). 1 -¿ 0 — Random(o) took 1 stone(s). Winner: MinMax20(x)

N=3 より==== 3 stone(s) ==== 3 -¿ 0 — Random(o) took 3 stone(s). Winner: MinMax20(x)

N=2 より==== 2 stone(s) ==== 2 -¿ 1 — Random(o) took 1 stone(s). 1 -¿ 0 — MinMax20(x) took 1 stone(s). Winner: Random(o)

10 課題 2-2

なぜそのように変化したのかを Eval.java のコードと min-max 法の仕組みに基づき論理的に説明せよ

```
public class Eval {
    float value(State state) {
      float s = state.winner().getSign(); // +1, -1, 0
      return state.isGoal() ? Float.POSITIVE_INFINITY * s : s / state.numStones;
    }
}
```

残り石数が少ないほど評価値の絶対値が大きくなるため、終盤になるほど評価値の差が大きくなる。そのため、探索深さが制限されている場合でも、終盤では正確な評価が可能となり MinMaxPlayer の強さが維持される。一方、序盤では評価値の差が小さいため、浅い探索深さでは最適な手を選択できず、MinMaxPlayer の強さが低下する。

11 課題 2-3

深さが浅くてもある程度賢く振る舞うためには、Eval クラスをどのように改良すべきかを考察し，具体的なアイデア（ヒューリスティック）を提案せよ

```
float value(State state) {
if (state.isGoal()) {
        float s = state.winner().getSign();
        return Float.POSITIVE_INFINITY * s;
    }
    float s = state.winner().getSign();
    int mod4 = state.numStones % 4;

    // の倍数を相手に残せている 4 = 有利
    // の倍数が自分のターン 4 = 不利
    float modBonus = (mod4 == 0) ? -1.0f : 1.0f;

    // 終盤ほど評価値を大きく
    float endgameWeight = 1.0f / (state.numStones + 1);

    return s * (modBonus * 10.0f + endgameWeight);
}
```

自分のターン終了時に相手に 4 の倍数の石を残すことができれば有利であるため、その状況を評価値に反映する。また、終盤になるほど評価値の差が重要になるため、残り石数に応じて評価値を調整する。

12　課題 2-4

上のアイデアを実際に実装し，ランダムプレイヤーとの勝率の変化を検証せよ（常勝は NG）
提出先ディレクトリを rep21d とする.

ヒューリスティック関数適用前勝った回数：19 回負けた回数：1 回ヒューリスティック関数適用後勝った回数：18 回負けた回数：2 回

もともと高い勝率だったためヒューリスティック関数の効果はあまり現れなかった。

13　課題 2-2a

自己評価／作業時間:　S 2 時間 00 分

評価関数の仕様を答えなさい.

public class Eval float value(State state) return -1025 * state.b3 + 511 * state.a3 - 63 * state.b2 + 31 * state.a2 - 15 * state.b1 + 7 * state.a1;

相手と自分の盤面のラインの数を評価値に反映している。具体的には、3 つ連続したラインを持つことが最も高く評価され、次に 2 つ連続したライン、最後に 1 つのラインが評価される。相手のラインは負の重みで評価され、自分のラインは正の重みで評価される。相手の負の評価値のほうが大きいため、相手のラインを阻止することが優先される。

14　課題 2-2b

自己評価／作業時間:　S 2 時間 55 分

人がコンピュータと対戦できるようにプログラムを作成せよ

HumanClass を実装した。プレイヤーの手番になると、コンソールに現在の盤面を表示し、ユーザーに行動を入力させる。入力された行動が有効であればその手を実行し、無効であれば再度入力を促す。

15　課題 2-2c

自己評価／作業時間:　A 1 時間 00 分

リスト 8 をネガマックスで実装せよ

minsearch と maxsearch を統合し、評価値の符号を反転させることでネガマックス法を実装した。関数の切り替えが不要になるため、コードが簡潔になり可読性が向上し、バグが減る。中間で生成される値も値を反転させれば相手ターンでも同じ関数を利用できる。

16　課題 2-2d

自己評価／作業時間:　S 3 時間 23 分

Move Ordering の有無による訪問ノード数の変化を具体的に報告せよ

```
=== Comparison Test: MyPlayerV1 (without Move Ordering) vs MyPlayerV2 (with Move Ordering) ===

--- Game 1: RandomPlayer vs MyPlayerV1 (without Move Ordering) ---
 | |
-+-+-
 | |
-+-+-
o| |
o: a3=0,a2=0,a1=3
x: b3=0,b2=0,b1=0
--------------------
MyPlayerV1_6 visited nodes: 3008
 |x|
-+-+-
 | |
-+-+-
o| |
o: a3=0,a2=0,a1=3
x: b3=0,b2=0,b1=2
--------------------
 |x|
-+-+-
 | |
-+-+-
o|o|
o: a3=0,a2=1,a1=2
x: b3=0,b2=0,b1=1
--------------------
MyPlayerV1_6 visited nodes: 371
 |x|
-+-+-
x| |
-+-+-
o|o|
o: a3=0,a2=1,a1=1
x: b3=0,b2=0,b1=2
--------------------
 |x|
-+-+-
x|o|
-+-+-
o|o|
o: a3=0,a2=2,a1=1
```

```
x: b3=0,b2=0,b1=1
--------------------
MyPlayerV1_6 visited nodes: 32
 |x|
-+-+-
x|o|x
-+-+-
o|o|
o: a3=0,a2=2,a1=1
x: b3=0,b2=0,b1=2
--------------------
 |x|o
-+-+-
x|o|x
-+-+-
o|o|
o: a3=1,a2=1,a1=1
x: b3=0,b2=0,b1=0
--------------------
winner: Random


--- Game 2: RandomPlayer vs MyPlayerV2 (with Move Ordering) ---
 | |
-+-+-
 |o|
-+-+-
 | |
o: a3=0,a2=0,a1=4
x: b3=0,b2=0,b1=0
--------------------
Visited nodes (with Move Ordering): 633
 |x|
-+-+-
 |o|
-+-+-
 | |
o: a3=0,a2=0,a1=3
x: b3=0,b2=0,b1=1
--------------------
o|x|
-+-+-
 |o|
-+-+-
 | |
o: a3=0,a2=1,a1=3
x: b3=0,b2=0,b1=0
--------------------
Visited nodes (with Move Ordering): 196
o|x|
```

```
-+-+-
 |o|
-+-+-
 |x|
o: a3=0,a2=1,a1=3
x: b3=0,b2=0,b1=1
--------------------
o|x|o
-+-+-
 |o|
-+-+-
 |x|
o: a3=0,a2=2,a1=3
x: b3=0,b2=0,b1=1
--------------------
Visited nodes (with Move Ordering): 26
o|x|o
-+-+-
x|o|
-+-+-
 |x|
o: a3=0,a2=2,a1=1
x: b3=0,b2=0,b1=1
--------------------
o|x|o
-+-+-
x|o|
-+-+-
o|x|
o: a3=1,a2=1,a1=1
x: b3=0,b2=0,b1=0
--------------------
winner: Random
```

Player1V と Player2V の比較では、Move Ordering を導入した MyPlayerV2 の方が訪問ノード数が大幅に削減されている。具体的には、MyPlayerV1 では 3008 ノードに対し、MyPlayerV2 では 633 ノードと約 79 パーセントの削減が達成されている。Move Ordering により、有望な手を優先的に探索することで、無駄な探索を避けることができたためである。ただ終盤は全訪問ノードが少なくなっているので、Move Ordering の効果が相対的に小さくなっている。

17 課題 2-2e

自己評価／作業時間: S 2 時間 03 分

対称（反転・回転）な状態を重複とみなす場合とみなさない場合でそれぞれ総状態数を答えよ

観測可能な状態数 Total states (without symmetry consideration): 5478 Total states (with symmetry consideration): 765