知能処理学 演習指示書*

演習 2-1 ゲーム木探索

大囿忠親

2023/12/12

*複製、転載または配布を禁止する。

1　概要

1.1　例題 1: 単純なゲーム木

講義中に扱った図 1 の単純なゲーム木を例題とする。

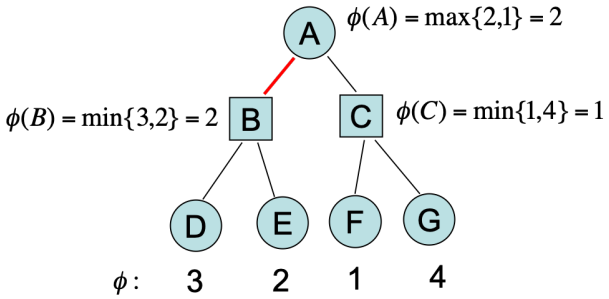

図 1: 単純なゲーム木 その 1

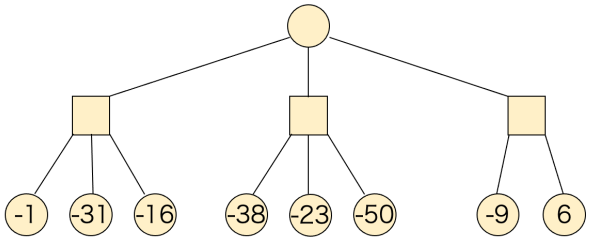

図 2: 単純なゲーム木 その 2

1.2　例題 2: 石取りゲーム

次のゲームを例題とする。

ゲームの例

場に置かれた N 個の石を 2 人のプレイヤーで交互に取り合うゲームを考える。プレイヤーは、場にある 1〜3 個の石を取り、手元に置く。2 人のプレイヤーは、交互にこれを繰り返す。場に置かれた最後の石を取ったプレイヤーを負けとする。

2　課題

以下のすべての課題に取り組むこと。

写経が指示されている場合は、次のいずれかの作業を実施せよ。プログラミングが得意でない場合は、2の作業を行うことを強く推奨する。

1. 対象のプログラムを演習指示書からコピペして、重要箇所をコメントせよ。
2. プログラムの意味を考えながらキーボードで一文字ずつプログラムを打ち込むこと。理解した内容をプログラム中にコメントとして記載すること。作業時間を必ず計測し、レポートにて報告せよ。写経の作業時間とは、入力を開始してから、プログラムを正常に実行するまでの時間を意味する。

プログラム等を提出する際は、各課題で指定されたディレクトリをルートとしたディレクトリ構成とすること。例えば、課題で指定されたディレクトリが `ex` で、提出するプログラム `Example.java` のパッケージ名が `pkg` の場合のパスは、ディレクトリ `ex` をルートとして、`ex/pkg/Example.java` となる。

課題 2-1a　（基礎・必須 ／ ex21a）

例題 1 に関連するプログラムであるリスト 1〜リスト 3 を写経せよ。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写経すること。また、リスト 3 を利用して図 2 の評価値を求めよ。また、枝刈りが発生したノードを報告すること。提出先ディレクトリを `ex21a` とする。

課題 2-1b　（基礎・必須 ／ ex21b）

例題 2 に関連するプログラムであるリスト 4〜リスト 11 を写経せよ。講義で示したアルゴリズムと見比べながら写経すること。各クラスの役割をレポートにて説明すること。また、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。提出先ディレクトリを `ex21b` とする。

課題 2-1c　（応用・必須 ／ ex21c）

例題 2 の石取りゲームの完全解析を行うプログラムを作成せよ。入力は石の数であり、出力は先手必勝・引き分け・後手必勝の何れかである。解析結果の出力時に訪問ノード数も表示すること。提出先ディレクトリを `ex21c` とする。

課題 2-1d　（応用・必須 ／ ex21d）

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。クラス名を `MyAlphaBetaPlayer` とし、ファイル名を `MyAlphaBetaPlayer.java` とすること。次に挙げるすべての機能を実現すること。提出先ディレクトリを `ex21d` とする。

1. 枝刈り発生時に、α カットが発生したのか、もしくは β カットが発生したのかを表示
2. プログラム終了時に、訪問ノード数および枝刈り回数を表示

課題 2-1e　（応用・必須 ／ ex21e）

ここまでで作成したプログラムを利用して α-β カット法による効率改善効果を調べよ。次のグラフをレポートに掲載し考察を示すこと。提出先ディレクトリを `ex21e` とする。

グラフ 1: 横軸: 石の個数（2, 3, ..., 20）、縦軸: 訪問回数（min-max 法の場合および α-β カット法の場合）

グラフ 2: 横軸: 石の個数（2, 3, ..., 20）、縦軸: α-β カット法における枝刈り発生回数

課題 2-1f　（発展・必須 ／ ex21f）

課題 2-1d で作成した MyAlphaBetaPlayer.java を改変してネガマックス探索を実現せよ。先攻および後攻の両方で正常に動作することを示すこと。クラス名およびファイル名を、それぞれ MyNegaMaxPlayer および MyNegaMaxPlayer.java とすること。提出先ディレクトリを ex21f とする。

（ヒント）静的評価関数に注意せよ。

3 プログラム

リスト 1: ex3a/State.java

```
1  package ex3a;
2
3  import java.util.*;
4
5  class State {
6    static Map<String, List<String>> childNodeLists = Map.of(
7        "A", List.of("B", "C"),
8        "B", List.of("D", "E"),
9        "C", List.of("F", "G"));
10   static Map<String, Float> values = Map.of(
11       "D", 3.0f,
12       "E", 2.0f,
13       "F", 1.0f,
14       "G", 4.0f);
15
16   String current;
17
18   State(String current) {
19     this.current = current;
20   }
21
22   public String toString() {
23     return this.current.toString();
24   }
25
26   boolean isGoal() {
27     return getMoves().isEmpty();
28   }
29
30   List<String> getMoves() {
31     return State.childNodeLists.getOrDefault(this.current, new ArrayList<>());
32   }
33
34   State perform(String move) {
35     return new State(move);
36   }
37 }
38
39 class Eval {
40   float value(State state) {
41     return State.values.getOrDefault(state.current, Float.NaN);
42   }
43 }
```

リスト 2: ex3a/MinMaxSearch.java

```
package ex3a;

import static java.lang.Float.*;

class MinMaxSearch {
  public static void main(String[] args) {
    var player = new MinMaxSearch(new Eval(), 2);
    var value = player.search(new State("A"));
    System.out.println(value);
  }

  Eval eval;
  int depthLimit;

  MinMaxSearch(Eval eval, int deapthLimit) {
    this.eval = eval;
    this.depthLimit = deapthLimit;
  }

  float search(State state) {
    return maxSearch(state, 0);
  }

  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;
  }

  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;
  }

  boolean isTerminal(State state, int depth) {
    return state.isGoal() || depth >= this.depthLimit;
  }
}
```

リスト 3: ex3a/AlphaBetaSearch.java

```
package ex3a;

import static java.lang.Float.*;

class AlphaBetaSearch {
  public static void main(String[] args) {
    var player = new AlphaBetaSearch(new Eval(), 2);
    var value = player.search(new State("A"));
```

```
    System.out.println(value);
  }

  Eval eval;
  int depthLimit;

  AlphaBetaSearch(Eval eval, int deapthLimit) {
    this.eval = eval;
    this.depthLimit = deapthLimit;
  }

  float search(State state) {
    return maxSearch(state, NEGATIVE_INFINITY, POSITIVE_INFINITY, 0);
  }

  float maxSearch(State state, float alpha, float beta, 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, alpha, beta, depth + 1);
      v = Math.max(v, v0);

      if (beta <= v0) {
        break;
      }

      alpha = Math.max(alpha, v0);
    }

    return v;
  }

  float minSearch(State state, float alpha, float beta, 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, alpha, beta, depth + 1);
      v = Math.min(v, v0);

      if (alpha >= v0) {
        break;
      }

      beta = Math.min(beta, v0);
    }

    return v;
  }

  boolean isTerminal(State state, int depth) {
    return state.isGoal() || depth >= this.depthLimit;
  }
}
```

リスト 4: ex3b/Game.java

```
1  package ex3b;
2
3  import static ex3b.Color.*;
4
5  import java.util.*;
6
7  public class Game {
8    public static void main(String[] args) {
9      for (int numStones = 1; numStones <= 20; numStones++) {
10       var p0 = new RandomPlayer();
11       var p1 = new MinMaxPlayer(new Eval(), 20);
12       Game g = new Game(numStones, p0, p1);
13       g.play();
14       g.printResult();
15     }
16   }
17
18   State state;
19   Map<Color, Player> players;
20
21   public Game(int numStones, Player black, Player white) {
22     black.color = BLACK;
23     white.color = WHITE;
24     this.state = new State(numStones);
25     this.players = Map.of(BLACK, black, WHITE, white);
26   }
27
28   void play() {
29     System.out.printf("==== %d stone(s) ====\n", state.numStones);
30
31     while (this.state.isGoal() == false) {
32       var player = this.players.get(this.state.color);
33       var move = player.think(this.state.clone());
34
35       var next = this.state.perform(move);
36       System.out.printf("%s -> %s | %s %s.\n", state, next, player, move);
37       this.state = next;
38     }
39   }
40
41   void printResult() {
42     System.out.println("Winner: " + this.players.get(this.state.winner()));
43     System.out.println();
44   }
45 }
```

リスト 5: ex3b/Move.java

```
1  package ex3b;
2
3  public class Move {
4    int removal;
5    Color color;
6
7    public Move(int removal, Color color) {
8      this.removal = removal;
9      this.color = color;
10   }
11
12   public String toString() {
13     return String.format("took %d stone(s)", this.removal);
14   }
15 }
```

リスト 6: ex3b/Color.java

```
1  package ex3b;
2
3  public enum Color {
4    BLACK(1, "o"),
5    WHITE(-1, "x"),
6    NONE(0, " ");
7
8    private int sign;
9    private String symbol;
10
11   private Color(int sign, String symbol) {
12     this.sign = sign;
13     this.symbol = symbol;
14   }
15
16   public int getSign() {
17     return this.sign;
18   }
19
20   public Color flipped() {
21     switch (this) {
22     case BLACK: return WHITE;
23     case WHITE: return BLACK;
24     default: break;
25     }
26     return NONE;
27   }
28
29   public String toString() {
30     return this.symbol;
31   }
32 }
```

リスト 7: ex3b/Player.java

```
package ex3b;

public abstract class Player {
  String name;
  Color color;

  public Player(String name) {
    this.name = name;
  }

  public String toString() {
    return String.format("%s(%s)", this.name, this.color);
  }

  public Move think(State state) {
    Move move = search(state);
    move.color = this.color;
    return move;
  }

  abstract Move search(State state);
}
```

リスト 8: ex3b/RandomPlayer.java

```
package ex3b;

public class RandomPlayer extends Player {
  public RandomPlayer() {
    super("Random");
  }

  Move search(State state) {
    var moves = state.getMoves();
    int index = new java.util.Random().nextInt(moves.size());
    return moves.get(index);
  }
}
```

リスト 9: ex3b/State.java

```java
package ex3b;

import static ex3b.Color.*;

import java.util.*;
import java.util.stream.*;

public class State implements Cloneable {
  int numStones;
  Color color = BLACK;
  Move move;

  public State(int numStones) {
    this.numStones = numStones;
  }

  public State clone() {
    State other = new State(this.numStones);
    other.color = this.color;
    other.move = this.move;
    return other;
  }

  public String toString() {
    return String.format("%2d", this.numStones);
  }

  public boolean isGoal() {
    return this.numStones == 0;
  }

  public Color winner() {
    return isGoal() ? this.color : NONE;
  }

  public List<Move> getMoves() {
    var n = Math.min(3, this.numStones);
    return IntStream.rangeClosed(1, n)
        .mapToObj(i -> new Move(i, this.color))
        .toList();
  }

  public State perform(Move move) {
    var next = clone();
    next.numStones -= move.removal;
    next.color = this.color.flipped();
    next.move = move;
    return next;
  }
}
```

リスト 10: ex3b/Eval.java

```java
package ex3b;

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

リスト 11: ex3b/MinMaxPlayer.java

```
1  package ex3b;
2
3  import static java.lang.Float.*;
4
5  public class MinMaxPlayer extends Player {
6    Eval eval;
7    int depthLimit;
8    Move move;
9
10   public MinMaxPlayer(Eval eval, int depthLimit) {
11     super("MinMax" + depthLimit);
12     this.eval = eval;
13     this.depthLimit = depthLimit;
14   }
15
16   Move search(State state) {
17     this.move = new Move(Math.min(3, state.numStones), state.color);
18
19     if (this.color == Color.BLACK) {
20       maxSearch(state, 0);
21     } else {
22       minSearch(state, 0);
23     }
24
25     return this.move;
26   }
27
28   float maxSearch(State state, int depth) {
29     if (isTerminal(state, depth)) {
30       return this.eval.value(state);
31     }
32
33     var v = NEGATIVE_INFINITY;
34
35     for (var move : state.getMoves()) {
36       var next = state.perform(move);
37       var v0 = minSearch(next, depth + 1);
38
39       if (depth == 0 && v0 > v) {
40         this.move = move;
41       }
42
43       v = Math.max(v, v0);
44     }
45
46     return v;
47   }
48
49   float minSearch(State state, int depth) {
50     if (isTerminal(state, depth)) {
51       return this.eval.value(state);
52     }
53
54     var v = POSITIVE_INFINITY;
55
56     for (var move : state.getMoves()) {
57       var next = state.perform(move);
58       var v0 = maxSearch(next, depth + 1);
59
60       if (depth == 0 && v0 < v) {
61         this.move = move;
62       }
63
64       v = Math.min(v, v0);
65     }
66
67     return v;
68   }
```

```
69
70    boolean isTerminal(State state, int depth) {
71      return state.isGoal() || depth >= this.depthLimit;
72    }
73  }
```