

1. 概要

本ドキュメントは、マインスイーパの各セルが「どのレベルのテクニック」を用いれば論理的に確定可能かを判定するプログラムの設計方針をまとめたものである。

既存の HintCountCalculator（必要ヒント数 k の算出）とは別に、「人間が解く際に使用する論理パターン（定石）」に基づいて難易度を分類する。

2. クラス設計方針

テクニック判定を行うための専用クラス TechniqueAnalyzer を作成する。このクラスは、盤面の状態を受け取り、「今の盤面で、どのテクニックを使えば次に進めるか」を分析する。

2.1. 推論エンジン (Inference Engine)

人間のように「簡単な手法から順に試す」アプローチをとる。

- Lv1-1 (埋めるだけ) を全探索し、解けるセルがあれば確定。
- 解けなければ Lv1-2/1-3 (包含) を試す。
- 解けなければ Lv1-4 (共通) を試す。
- それでも解けなければ Lv2 (背理法) を試す。

2.2. データ構造: Region (または VirtualHint)

Lv1-2以降の複雑な推論を扱うため、盤面上のヒント数字だけでなく、「推論によって得られた仮想的なヒント」を扱えるようにする。

Region:

セルの集合 (Set) と、その中に含まれる地雷数 (int) のペア。

例:

- {セルA, セルB, セルC} に地雷は残り 1個
- 初期状態では、盤面上の各数字ヒントがそれぞれ1つの Region となる。推論が進むにつれて、領域の差分や共通部分から新しい Region が生成される。

3. レベル別アルゴリズム設計

Lv1-1. 埋めるだけ (Single Hint Logic)

もっとも基本的な処理。単一のヒント数字のみを見て判断する。

入力:

開いているヒントセル H

情報:

- $\$U\$$: $\$H\$$ の周囲にある未確定セル数
- $\$F\$$: $\$H\$$ の周囲にある地雷確定セル数 (旗)
- $\$V\$$: $\$H\$$ の数字

判定ロジック:

- 残り地雷すべて: $\$V = F + U\$$ ならば、残りの未確定セル $\$U\$$ はすべて 地雷。
- 残り安全すべて: $\$V = F\$$ ならば、残りの未確定セル $\$U\$$ はすべて 安全。

Lv1-2 & 1-3. 包含関係 (Subset Logic)

2つのヒント (または Region) $\$A$, $\$B$ の関係を利用する。 $\$A\$$ の対象セル集合 $\$S_A\$$ が、 $\$B\$$ の対象セル集合 $\$S_B\$$ の部分集合である場合 ($\$S_A \subset S_B\$$) に発動する。

前提:

ヒント $\$A$, $\$B$ が共通の未確定セルを持つ (隣接している)。

判定ロジック:

- $\$S_A \subset S_B\$$ であるかチェックする (Javaの `Set.containsAll` を利用)。
- 差分領域 $\$D = S_B - S_A\$$ を計算する。
- $\$D\$$ に含まれる地雷数 $\$M_D = M_B - M_A\$$ を計算する。

Lv1-2 (確定):

- $\$M_D = |D|\$$ (差分のセル数) なら、 $\$D\$$ はすべて 地雷。
- $\$M_D = 0\$$ なら、 $\$D\$$ はすべて 安全。

Lv1-3 (情報の伝播):

- 上記以外の場合、セルは確定しないが、新たな制約「領域 $\$D\$$ には地雷が $\$M_D\$$ 個ある」が得られる。
- この新しい Region $\$D\$$ をヒントリストに追加し、他のヒントとの組み合わせ (Lv1-2やLv1-4) に再利用する。

Lv1-4. 共通部分 (Intersection Logic)

2つのヒント $\$A$, $\$B$ が共通部分を持つが、包含関係にはない場合。「1-2-1」パターンなどの解決に相当する。

領域定義:

- $\$OnlyA = S_A - S_B\$$ (Aのみにあるセル)
- $\$Common = S_A \cap S_B\$$ (共通セル)

- $\$OnlyB = S_B - S_A$ (Bのみにあるセル)

判定ロジック:

- $\$Common$ に存在しうる地雷数の 最小値($\$minC$) と 最大値($\$maxC$) を計算する。
 - $\$maxC = \min(|Common|, M_A, M_B)$
 - $\$minC = \max(0, M_A - |OnlyA|, M_B - |OnlyB|)$
- この $\$min/max$ を使って、 $\$OnlyA$ や $\$OnlyB$ の地雷数を絞り込む。
- もし $\$M_A - minC = |OnlyA|$ なら、 $\$OnlyA$ はすべて地雷確定。
- もし $\$M_A - maxC = 0$ なら、 $\$OnlyA$ はすべて安全確定。
 - ($\$OnlyB$ についても同様)

Lv2. 背理法 (Trial & Error / Contradiction)

Lv1の手法ですべて詰まった場合に発動する。「仮定」をおいて矛盾を探すシミュレーション。

対象:

盤面上のすべての未確定セル (あるいは、情報の多いフロンティア上のセル) 。

判定ロジック (仮定法):

- ある未確定セル $\$X$ を選ぶ。
- 仮定1: $\$X$ が「地雷」であると仮定する。
- 仮定した状態で Lv1-1 (埋めるだけ) のロジックを連鎖的に走らせる (軽い推論) 。

矛盾チェック:

- あるヒント数字に対し、周囲の地雷数が数字を超えた。
- あるヒント数字に対し、残り全セルを地雷にしても数字に足りない。

結論:

矛盾が発生したら、仮定は誤りである。→ $\$X$ は 安全 で確定。(安全と仮定して矛盾すれば、地雷で確定)

4. 実装のステップ

いきなり全てを実装せず、以下の順序でクラスを拡張していくことを推奨する。

Phase 1: 基盤作成

- TechniqueAnalyzer クラスの作成。
- 盤面データを受け取り、コピーして操作できる仕組み。

- Region (または HintConstraint) クラスの定義 (Set cells, int - mines)。

Phase 2: Lv1-1 の実装

- もっとも単純なロジックを実装し、テストする。
- これだけで解ける簡単な盤面で動作確認。

Phase 3: Lv1-2/1-3 の実装

- 2つのヒントを比較するロジックを追加。
- Set演算 (containsAll, removeAll) を実装。
- 新しい制約をリストに追加する仕組み。

Phase 4: Lv2 の実装

- 盤面を一時的に変更して検証する try-catch 的なロジックの実装。
- 再帰深さは1 (仮定は1手のみ) とするか、深くするかを検討 (まずは深さ1推奨)。

Phase 5: Lv1-4 の実装 (オプション/高難度)

- 共通部分のmin/max計算ロジック。
- Lv1-2で多くのケースはカバーできるため、必要に応じて実装する。

5. 既存コードとの連携

- `HintCountCalculator` は「ヒント数 \$k\$」という純粋な計算量的指標。
- `TechniqueAnalyzer` は「テクニックレベル」という定性的指標。

最終的には、各セルに対して (必要なヒント数 k , 必要なテクニック Lv) のペアで評価を出力する形を目指す。

6. Phase 1 詳細設計 (基盤作成)

Phase 1では、アルゴリズムの器となるクラス構造を定義する。

6.1. データクラス `Region`

盤面上の「ある範囲」と「その中の地雷数」を表現する不変 (Immutable) クラス。`HintCountCalculator` では盤面配列だけで計算したが、Lv1-2以降の「飛び地」や「仮想ヒント」を扱うために必須となる。

フィールド

- `Set<Integer> cells`: この領域に含まれる未確定セルのインデックス集合。
- `int mines`: この領域内に存在する地雷の数。

メソッド

- `isSubsetOf(Region other)`: 自分が相手の部分集合かどうか判定。
- `subtract(Region other)`: 差分領域（自分 - 相手）を新しい `Region` として返す。
- `toString()`: デバッグ用（例: "{10, 11, 12} = 1"）。

6.2. 解析クラス TechniqueAnalyzer

盤面全体の状態を管理し、推論のステップを進めるコントローラー。

フィールド

- `int[] board`: 現在の盤面状態（常に最新を維持）。
- `int[] completeBoard`: 正解盤面（推論結果の検証用）。
- `int[] difficultyMap`: 各セルの確定にかかったテクニックレベルを記録。
- `List<Region> activeRegions`: 現在有効な推論の手がかりリスト。

メソッド

- `analyze()`: 解析のメインループ。
 - `initRegions()` で初期化。
 - 変化がなくなるまでループ:
 1. `solveLv1_1()`
 2. `solveLv1_2()` ...
- `initRegions()`: 盤面上の数字ヒントを走査し、初期の `Region` リストを生成する。
 - 数字セルを見つけ、その周囲の「未確定セル」を集めて `Region` を作成。
- `applyResult(Map<Integer, Integer> deduced)`: 推論で確定したセル（Safe/Mine）を盤面に反映する。
 - 盤面 `board` を更新。
 - `activeRegions` 内の各 `Region` から、確定したセルを除去し、地雷なら `mines` を減らす処理を行う。

6.3. 解析フロー (Phase 1版)

まずは Lv1-1 などの実装は空で行い、データの流れだけを確認する。

1. コンストラクタで盤面を受け取る。
2. `analyze()` を呼ぶ。
3. 盤面から `Region` のリストを作成する（まだ中身は数字ヒント由来のみ）。
4. ループ内で「何も確定しなかった」として即終了する。
5. 結果（まだすべて -1）を出力する。

この基盤があれば、Phase 2 以降は `solveLv1_1` メソッドの中身を書くだけで機能拡張ができる。