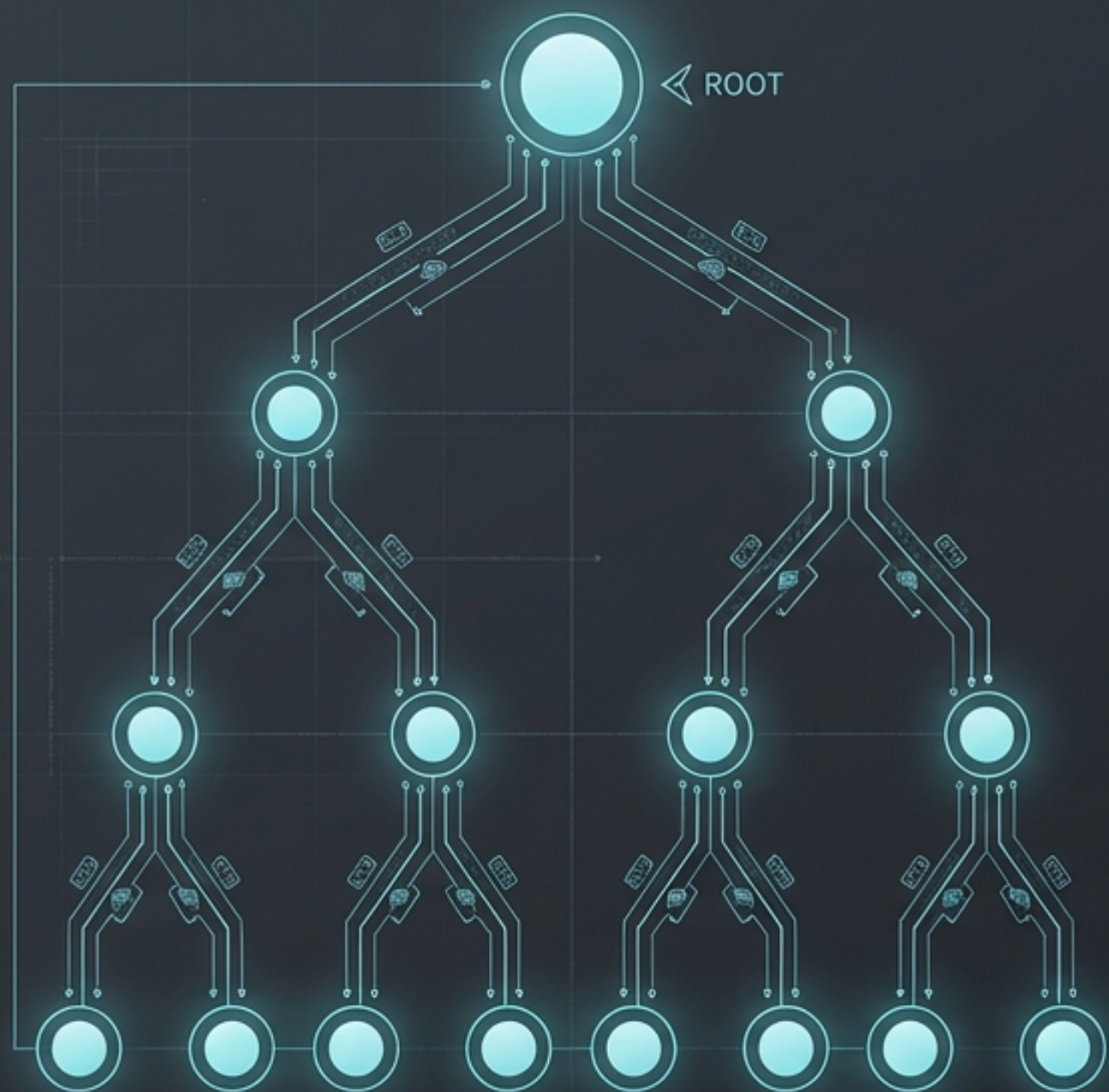
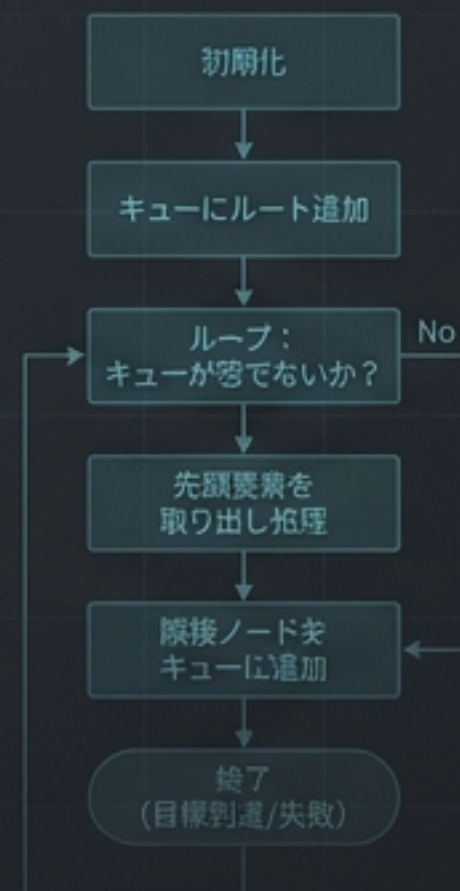


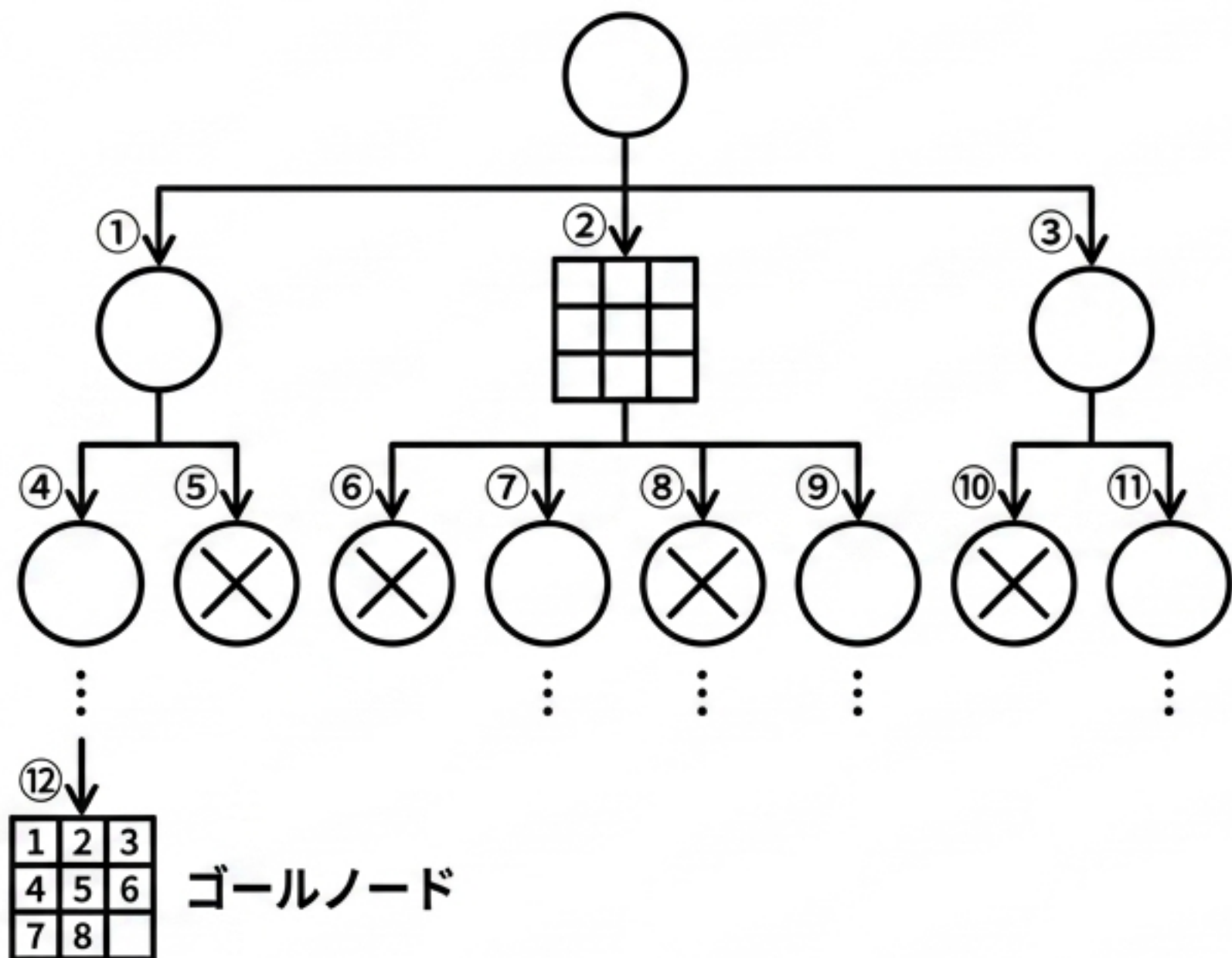
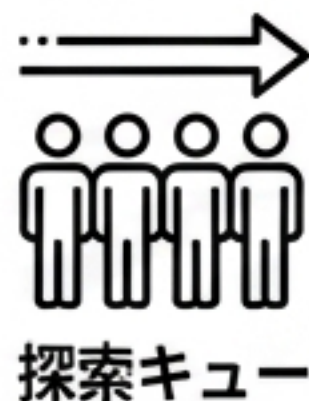


応用情報技術者試験 プログラミング ロジック解体新書

幅優先探索における状態管理と終了判定の
思考プロセス(令和7年春期 問3 設問2)

設問2の空欄[ア],[カ],[キ],[ク]に隠されたアルゴリズムの構造を読み解く。





幅優先探索（BFS）の全体像とマスターループ

幅優先探索は、「次に調べる候補（キュー）」と「既に調べた履歴（リスト）」を管理し、条件を満たすまでループを回し続けるエンジンである。

- 本解説では、プログラムの中核となる while ループの制御ロジックを解剖する。
- 空欄はそれぞれ「終了監視」「ゴール判定」「状態の記録」という最重要パーツを担う。



空欄 [ア] の論理：探索の「限界」を知る

20行目:

```
while ( [ ア ] がfalseと等しい )
```

【状況】

ゴールが見つかればループ内で return される。つまり、この while 条件は「ゴールが見つからず、調べるべき盤面が尽きた」場合を判定している。

【着眼点】

「調べるべき盤面」は変数 explore_queue に格納されている。

【仕様の確認】

クラス Queue のメソッド一覧から、状態を調べるための「戻り値が論理型」のメソッドを探す。

メソッド	戻り値	説明
add(BoardState: st)	なし	キューに BoardState のインスタンスへの参照 st を追加する。
isEmpty()	論理型	キューが空の場合は true を返し、空でない場合は false を返す。
poll()	BoardState	先頭の BoardState のインスタンスへの参照を取り出して返す。
peek()	BoardState	先頭の BoardState のインスタンスへの参照を返す。

空欄 [力] の論理：勝利条件（ゴール）の検知

```
33: if ( [ 力 ] がtrueと等しい )
34:     printResult(new_state)
35:     return
```

 **【状況】**
新しい盤面状態 (new_state) が生成された直後、結果を出力してプログラムが終了 (return) している。

 **【着眼点】**
ここでは「生成された盤面がゴールと一致したか」を判定している。

 **【仕様の確認】**
関数一覧から、「2つの盤面配列を比較して論理型を返す関数」を特定する。

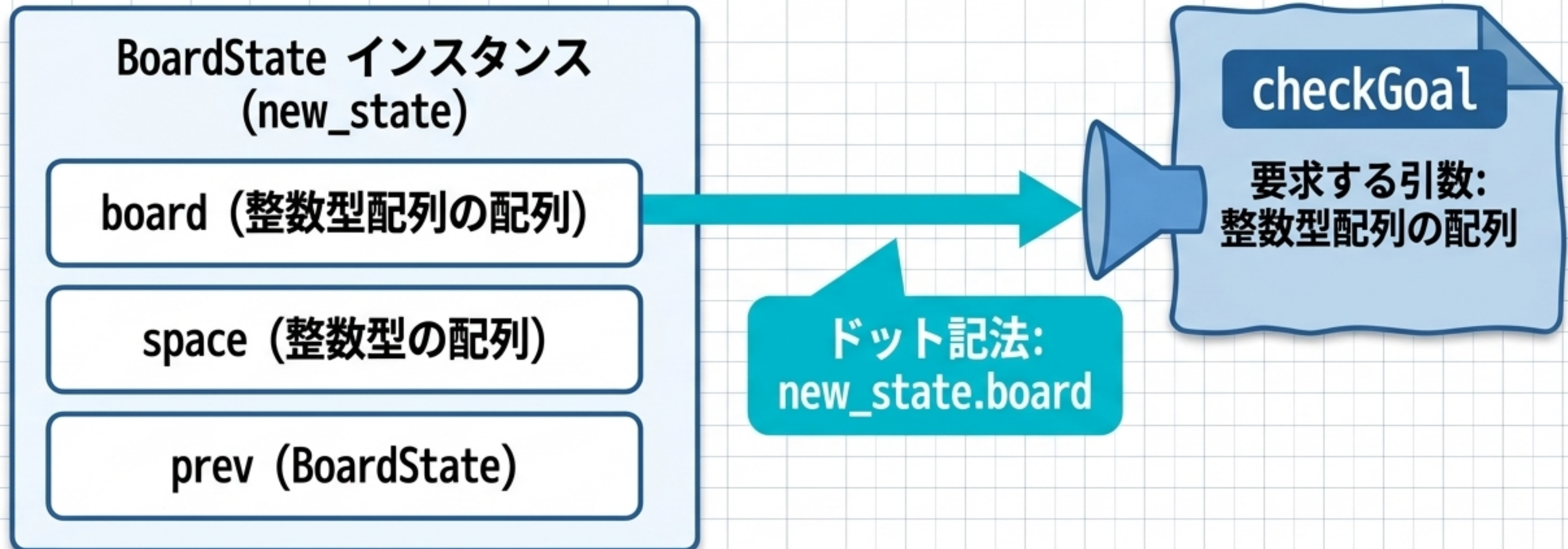
主な関数（抜粋）

名称	戻り値	説明
createGoal(整数型: board_size)	整数型配列の配列	引数 board_size の値を一辺のマス数としたスライドパズルのゴールの盤面を表す配列（以下、ゴールの盤面配列という）を作成して返す。
checkGoal(整数型配列の配列: b, 整数型配列の配列: g)	論理型	第一引数の盤面配列 b と第二引数のゴールの盤面配列 g を用いて両者が一致しているかどうかを判定して結果を返す。一致する場合は true を返し、一致しない場合は false を返す。
printResult(BoardState: st)	なし	引数として与えられた BoardState のインスタンスへの参照 st を用いて、開始時点の盤面からの遷移及び駒の移動回数を出力する。

引数の「型」を合わせる：インスタンスからのデータ抽出

判定関数が要求しているのは「状態オブジェクト」全体ではなく、その内部にある「盤面配列」のデータである。

- ・ゴール判定関数は、現在の状態とゴールの「盤面配列」を引数に取る。
- ・new_state の中から適切なメンバ変数をドット記法で指定する必要がある。



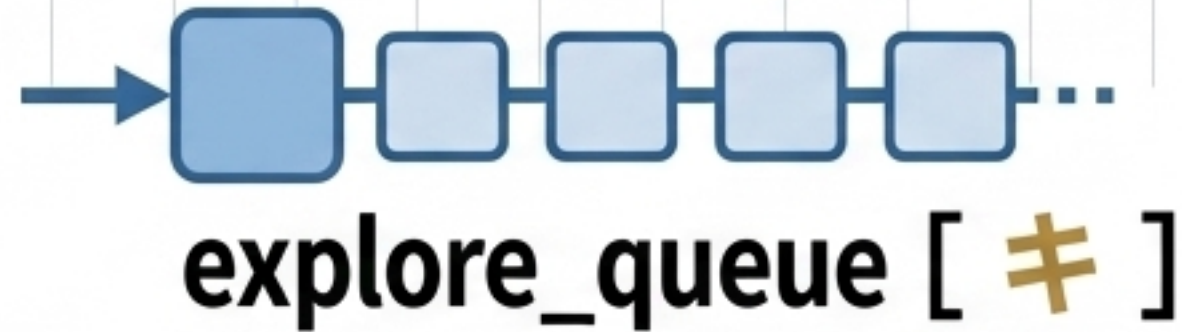
空欄 [キ] と [ク] の論理：重複排除と探索の継続

【状況】

新しい盤面が「ゴールではなく」「過去の履歴とも重複していない」ことが確認された。

【アクション】

この未知の状態に対して、次に2つのアクションを起こす。



未来への布石

さらに深く探索するために、
探索キューの末尾に追加する。



check_list [ク]

過去の記憶

無限ループを防ぐため、
探索済みリストに追加する。

```
37: if (checkSameBoard(new_state.board, check_list)がfalseと等しい)
38:     explore_queue.[ キ ]
39:     check_list.[ ク ]
```

格納先が要求する「型」を見極める

コレクション（キューとリスト）が似た役割を持っていたとしても、クラスの仕様によって追加すべきデータの「型」が異なる。

- [キ]: クラスQueueの仕様に基づき、状態オブジェクトへの参照をそのまま渡す。
- [ク]: クラスListの仕様に基づき、状態オブジェクトから「盤面配列」のみを抽出して渡す。

explore_queue.[キ]

add(BoardState: st)



check_list.[ク]

add(整数型配列の配列: b)



ピースがはまり、アルゴリズムが動き出す

クラスや関数の「仕様（型と戻り値）」を正確に読み解くことで、アルゴリズムの挙動はパズルのように論理的に導き出せる。

幅優先探索（BFS）は、これらの部品が連動することで、最短手数を確実に導き出す堅牢なシステムである。

