Quick Links

- Bitboard encoding details
- DebugState class referece
- Isolation class referece

Bitboard Encoding Overview

This section describes the bit board encoding

Bitboard Advantages:

- Very compact representation (a single long integer encodes the entire board)
- Most methods (check for legal actions, apply actions, copy state) are just bit-wise operations
- Bitboards are **fast**

Bitboard Disadvantages:

- Complex encoding makes it difficult to understand and debug

Bitboard Encoding

Why add borders? It eliminates the need to check boundaries of the board except at the lowest bits.

```
                 Grid Coordinates                  Array Index

             |-----|-----|-----|-----|       |-----|-----|-----|-----|
             |     | A 1 | A 2 |     |       | 1 1 | 1 0 |  9  |  8  |
             |-----|-----|-----|-----|       |-----|-----|-----|-----|
             |     | B 1 | B 2 |     |       |  7  |  6  |  5  |  4  |
             |-----|-----|-----|-----|       |-----|-----|-----|-----|
             |     | C 1 | C 2 |     |       |  3  |  2  |  1  |  0  |
             |-----|-----|-----|-----|       |-----|-----|-----|-----|


                               Bit-view (Flattened Array)

Coords:         (A 1) (A 2)             (B 1) (B 2)             (C 1) (C 2)
         |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
         |  0  |  1  |  1  |  0  |  0  |  1  |  1  |  0  |  0  |  1  |  1  |  0  |
         |-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
Index:     1 1   1 0    9     8     7     6     5     4     3     2     1     0
                |<-W--|                 |--E->|
                |<---------- N -----------|----------- S ---------->|
```

Important Note:

- **The actual implementation differs in one significant way from the description above.** The board can be further simplified by eliminating the left-most and right-most zeros (at indices 0 and 11 in the example above). Leading zeros are implicitly non-functional, and the single trailing zero is just a constant offset on the other indices. As a result, C2 is at index 0, B2 is at index 4, and indices 10 & 11 do not exist.

The final bit string has a length of (WIDTH + 2) * HEIGHT - 2, (the width of each row, times the height of the board, subtracting one leading & one trailing zero bit).

Movement

Pieces are moved on the bitboard by setting the appropriate bit (typically AND with a mask)

- Moving west from A2->A1 increases the index by 1 from 9 to 10
- Moving east from B1->B2 decreases the index by 1 from 6 to 5
- Moving north from B1->A1 increases the index by the width of the board plus the two border cells (2 + 2 = 4) from 6 to 10
- Moving south from B1->C1 decreases the index by the width of the board plus the two border cells (2 + 2 = 4) from 6 to 2

DebugState class

(subclass `isolation.Isolation`)

Subclass of `isolation.Isolation` with methods for visualizing the state of the game.

Constructor

DO NOT USE THE CONSTRUCTOR--use `.from_state()` instead

Attributes

See Also: `isolation.Isolation`

bitboard_string : str

A binary representation of the bitboard. By default, python formats the standard string for the Isolation class as the long integer (base-10) representation of the board.

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> DebugState.from_state(state).bitboard_string
'11111111111001111111111100111111111110011111111111001111011111100111111111110011111111111001
```

Class Methods

ind2xy(index)

Convert from board index value to xy coordinates

Example:

```
>>> from isolation import Isolation, DebugState
>>> DebugState.ind2xy(57)
```

```
(5, 4)
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> state.actions()[0]
<Action.NNE: 25>
>>> DebugState.ind2xy(state.actions()[0])
(12, 1)
```

Static Methods

from_state(gamestate)

Create and return a DebugState instance with the same attributes as the source game state

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> dbstate = DebugState.from_state(state)
>>> dbstate
DebugState(board=41523161203939121938568444148443134, ply_count=2, locs=(57, 0))
```

Public Methods

str

Generate a string representation of the current game state, marking the location of each player and indicating which cells have been blocked, and which remain open.

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> dbstate = DebugState.from_state(state)
>>> print(dbstate)

+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   | 1 |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   | 2 |
+ - + - + - + - + - + - + - + - + - + - + - +
```

Isolation class

Class representing the current state information for the game Isolation on an 11x9 rectangular grid with tokens that move in L-shaped patterns (like knights in chess). The class subclasses `NamedTuple`, which makes the states (effectively) immutable and hashable. Using immutable states can help avoid subtle bugs that can arise with in-place state updates. Hashable states allow states to be used as dict keys (e.g., for an opening book).

Constructor

```
Isolation.__init__(self, board: int=_BLANK_BOARD, ply_count: int=0, locs: tuple=(0, 0))
```

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> initial_state
Isolation(board=41523161203939122082683632224299007, ply_count=0, locs=(None, None))
```

Attributes

board : int

Bitboard representation of isolation game state. Bits that are ones represent open cells; bits that are zeros represent blocked cells. By default, python prints the board in base-10; you can inspect the bitboard representation using the `DebugBoard.bitboard_string` method.

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> state.board
41523161203939121938568444148443134
```

ply_count : int

Cumulative count of the number of actions applied to the board

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> state.ply_count
2
```

locs : tuple

A pair of values defining the location of each player. Default for each player is None while the player has not yet placed their piece on the board; otherwise an integer.

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> state.locs
(57, 0)
```

Public Methods

actions(self)

Return a list of the legal actions in the current state. Players can choose any open cell on the board for their opening move, but all later moves MUST be one of the values in isolation.Actions. (This method automatically handles returning the appropriate values.)

On open boards, the method returns a list of cell indices that are open; but during play it returns a list of `Action` values (defined in isolation.py).

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> initial_state.actions()
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 26, 27, 28, 29
>>> state = initial_state.result(57).result(0)  # p1 takes center; p2 takes bottom right corn
>>> state.actions()
[<Action.NNE: 25>, <Action.ENE: 11>, <Action.ESE: -15>, <Action.SSE: -27>, <Action.SSW: -25>,
```

player(self)

Return the id (zero for first player, one for second player) of player currently holding initiative (i.e., the active player)

Example:

```
>>> from isolation import Isolation
>>> initial_state = Isolation()
>>> initial_state.player()  # p1 has initiative
0
>>> state = initial_state.result(57)  # p1 takes center; p2 has initiative
>>> state.player()
1
```

result(self, action)

Return a new game state (no in-place updates) that results from applying the `action` argument to the current state. In any state, players should always choose an action from the list returned by the `actions()` method.

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()  # empty board
>>> print(DebugState.from_state(initial_state))

+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +

>>> state = initial_state.result(57).result(0)  # p1 takes board center; p2 takes bottom righ
>>> print(DebugState.from_state(state))

+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   | 1 |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   |   |
+ - + - + - + - + - + - + - + - + - + - + - +
|   |   |   |   |   |   |   |   |   |   | 2 |
+ - + - + - + - + - + - + - + - + - + - + - +
```

terminal_test(self, action)

Return True if either player has no legal moves, otherwise False

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()  # empty board
>>> initial_state.terminal_test()
False
>>> state = initial_state
>>> while not state.terminal_test(): state = state.result(state.actions()[0])  # play til end
...
>>> state
Isolation(board=36349881425809777517250506995532800, ply_count=84, locs=(32, 21))
>>> state.terminal_test()
True
```

utility(self, player_id)

Returns the utility of the current game state from the perspective of the player specified by the 0-indexed `player_id` argument (i.e., 0 for player 1 or 1 for player 2). The utility is defined as +∞ if the player specified by player_id is the winner, -∞ if the player specified by player_id is the loser, and 0 otherwise.

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()  # empty board
>>> initial_state.utility(initial_state.player())  # utility to player 1
0
>>> state = initial_state
>>> while not state.terminal_test(): state = state.result(state.actions()[0])  # play til end
...
>>> state
Isolation(board=36349881425809777517250506995532800, ply_count=84, locs=(32, 21))
>>> state.utility(initial_state.player())
-inf
>>> state.utility(1)  # player 2
inf
```

liberties(self, loc)

Return a list of liberties in the neighborhood of the index specified by the argument `loc`. "Liberties" are locations on the board that are not blocked in the current state. The "neighborhood" of a location is the set of cells that can be reached by the L-shaped movements of the player's token.

Example:

```
>>> from isolation import Isolation, DebugState
>>> initial_state = Isolation()  # empty board
>>> initial_state.liberties(57)
[82, 68, 42, 30, 32, 46, 72, 84]
```