PyNeutron

Filip Lew

Jan 15, 2020

CONTENTS:

CHAPTER

ONE

SRC

1.1 main module

1.2 neutron module

class neutron.**Neutron**(*board*, *position*)
Bases: *neutron.Soldier*

A special case of a *Soldier*.

A *Neutron* is different from a *Soldier* only by having a unique color value.

VALUE = 1

class neutron.**NeutronBoard**(*starting_grid=None*)
Bases: object

The Neutron game board.

It is represented by a 5x5 NumPy array. The purpose of this class is to manage the array, ensuring it doesn't get into an invalid state, and to provide useful functions for the game's logic.

Parameters **starting_grid** (numpy.array) – array representing starting board data.

grid
the array containing raw data of this board. Only for testing purposes

Type numpy.array

white_soldiers
list of *Soldier* objects representing white soldiers.

Type list

black_soldiers
list of *Soldier* objects representing black soldiers.

Type list

furthest_empty_spot(*pos*, *dir*)
Get the furthest empty position one can get by moving in direction dir from position pos without colliding with anything.

Implemented as a while loop checking following conditions:

- if the position after a step in the given direction is still in the board's bounds,
- if the position after the step is empty.

While those conditions are met, the step is performed, adding direction to position. If, after executing the loop, the resulting position is different from the starting position, we return it. Else, the move could not be made, and we return `None`.

Parameters

- **pos** (`util.Vec`) – starting position.
- **dir** (*str or* `util.Vec`) – direction in which to move.

Returns position of the furthest empty spot in the line of sight of source position, or `None` if the movement cannot be made.

Return type *util.Vec*

get_soldiers(*color*)

Get all soldiers of a given color present on the board.

Parameters **color** (*int*) – color of the soldiers.

Returns a list of *Soldier* objects containing all soldiers of a given color.

Return type list

Raises **ValueError** – if the color given is not a valid soldier color.

neighbors(*pos*)

Get values of board cells neighboring cell with position `pos`.

It iterates over coordinates from x-1 to x+1 and y-1 to y+1, making sure they are not out of the bounds of the board, and appends values at those positions to the resulting list. The source position itself is not included.

Parameters **pos** (`util.Vec`) – position of the cell.

Returns list of neighboring cells' values, without the source cell

Return type list

class neutron.**NeutronGame**(*board*, *first_player*, *second_player*)

Bases: `object`

The main Neutron game class.

Parameters

- **board** (`neutron.NeutronBoard`) – the game board to be used by this game instance
- **first_player** (`player.Player`) – the player who will start the game
- **second_player** (`player.Player`) – the second player

check_won()

Checks if the game was won, updating `self.winner` variable with the color of the winning player.

Returns winning player's color

Return type int

play_round()

Plays one round, swapping players afterwards.

start()

Starts the game, playing rounds until the game is won by either of the players.

class neutron.**Soldier**(*board*, *position*, *color*)
Bases: object

Class representing a soldier on the board.

Its main task is to enforce proper movement rules, to prevent the board from getting into an invalid state from the point of view of the game's rules.

Parameters

- **board** (neutron.NeutronBoard) – home board of this Soldier.
- **position** (util.Vec) – position of this Soldier on the board.
- **color** (*int*) – color of this Soldier.

move(*direction*)
Tries to move this *Soldier* in the given direction.

This method will fail if the given direction is not in *possible_directions*.

Works by calling *NeutronBoard.furthest_empty_spot()*, setting the position returned by this function to this *Soldier*'s color, and the original position to 0.

Parameters **direction** (*str*) – direction in which to move this *Soldier*.

Raises **ValueError** – if the given direction is not in *possible_directions*.

move_to_pos(*position*)
Tries to move this :class:'Soldier' to a given position.

This method will fail if the given position is not in *possible_moves*

Parameters **position** (util.Vec) – position to which move this *Soldier*

Raises **ValueError** – if the given position is not in *possible_moves*

property neighbors
List of neighboring cells of this Soldier. A thin wrapper around *NeutronBoard.neighbors()*.

property possible_directions
List of directions this *Soldier* can move.

Works by calling *NeutronBoard.furthest_empty_spot()* for this Soldier's position and filtering out None results.

property possible_moves
List of positions this *Soldier* can be after one move.

Works by supplying *NeutronBoard.furthest_empty_spot()* with all possible directions, then filtering out None results.

1.3 player module

class player.**HumanPlayer**(*board*, *color*, *home_row*)
Bases: *player.Player*

move_neutron()
Method called by the game when it's this player's turn to move the neutron.

move_soldier()
Method called by the game when it's this player's turn to move one of their soldiers.

class player.**Player**(*board*, *color*, *home_row*)
Bases: abc.ABC

Abstract base class of all Neutron game players. Defines methods called by the game to allow players to make decisions about the next move.

Parameters

- **board** (neutron.NeutronBoard) – board of the game played by this player.
- **color** (*int*) – color of this player's soldiers.
- **home_row** (*int*) – index of this player's home row on board.

abstract move_neutron()
Method called by the game when it's this player's turn to move the neutron.

abstract move_soldier()
Method called by the game when it's this player's turn to move one of their soldiers.

class player.**RandomPlayer**(*board*, *color*, *home_row*)
Bases: *player.Player*

move_neutron()
Method called by the game when it's this player's turn to move the neutron.

move_soldier()
Method called by the game when it's this player's turn to move one of their soldiers.

class player.**StrategyPlayer**(*board*, *color*, *home_row*)
Bases: *player.RandomPlayer*

A player that tries to apply some simple rules to increase its winning chance. If no rule can be applied in the current situation, it falls back to random movement.

block_enemy_row(*soldiers*, *enemy_row*)
Tries to block an empty spot in enemy's home row by putting one of the soldiers in there.

Parameters

- **soldiers** (*list*) – list of this player's soldiers.
- **enemy_row** (*int*) – index of the enemy player's row.

Returns True if the move was successful, False otherwise

Return type bool

block_neutron(*soldiers*)
Tries to completely block neutron from moving, if there's only one direction the neutron can move.

Returns True if the move was successful, False otherwise

Return type bool

move_into_home()
Tries to move the neutron into the home row

Returns True if the move was successful, False otherwise

Return type bool

move_neutron()
Method called by the game when it's this player's turn to move the neutron.

move_soldier()
Method called by the game when it's this player's turn to move one of their soldiers.

1.4 util module

class util.**Color**
Bases: object

BLACK = 3

WHITE = 2

color_names = {2: 'white', 3: 'black'}

class util.**Vec**(*x*, *y*)
Bases: object

A very simple implementation of a 2D vector, used to facilitate operations on positions and directions.

Parameters

- **x** (*int*) – vector's x coordinate.
- **y** (*int*) – vector's y coordinate.

static fromtuple(*tuple_pos*)
Creates a *Vec* from tuple (y, x). This order of coordinates was chosen to be compatible with NumPy's way of indexing multidimensional arrays.

Parameters **tuple_pos** (*tuple*) – a (y, x) tuple representing vector's coordinates.

Returns a newly created Vec.

Return type *Vec*

x

y

PYTHON MODULE INDEX

INDEX