MY ROBOT

Status

* Gold coins (start: 100)
* Health (start: 100 %)

Rounds

* beginning : + 1 gold coin
* if health < 100% : + 10% health

Sight

* range to
  ⇒ other robots
  - health
  - gold coins

  ⇒ walls

Actions

* move : $\uparrow, \downarrow$
  $\leftarrow, \rightarrow$
  $\nwarrow, \searrow$
  $\nearrow, \swarrow$

* set mines:
  1. YES/NO ?
  2. Coordinates (width, height) ?
     ... must be empty, otherwise gold coins will be charged w/o placing a mine

Cost of actions

* move :
  # moves = k
  $\text{cost}(k) = \text{sum}\{i \mid i = 1, .., k\}$
  Penalties :
  - hit wall ⇒ − 25% health
  - hit other robot ⇒ − 20 − 15 % health
  (... any penalty also cancels all following moves)
  - hit out-of-board ⇒ − 25% health
* mines : direct distance = k
  $\text{cost}(k) = k$

* Are gold coins charged at once or per turn ?
* In other words, if following moves are cancelled after penalty, is robot still charged for all planned moves ?

1) game_utils.py
line 127
<u>self</u>. is_blocked()

2) test_RobotRace.py
line 16

Pot of GOLD = GOAL

* initialized with 100 gold coins
  ⇒ + 1 gold coin / action of any robot
* after chosen # rounds emptied & newly initialized at new coordinate (100 gold coins)
* Other GAME-MODE :
  after chosen # rounds
  ⇒ − 1 gold coin / action of any robot

Robot with the most gold coins
⇒ WINNER

PROGRAM - related notes

1, __init__(self):
initialize player class => done before game-simulation is started

2, reset(self): (called within Simulator.play(self))
initialize player class variables => done after game-simulation was started

3, round_begin(self):
called within Simulator._begin_round(self)
=> tell robot what to do after
- expired mines have been removed
- timed-out gold pots have been removed & new ones relocated
- gold & health have been added to each Player`s status

4, set_mines(self):
called within Simulator._handle_setting_mines(self)
=> tell function whether and where to set mines
using a list of coordinates as tuples, i.e. (x, y)

5, move(self):

called within Simulator._handle_moving(self). askPlayer._askPlayerForMoves(self).ask()

=> tell function _handle_moving(self) how to move
using a list of move directions as Enums, i.e. name: 'right' etc...
value: 3 => indicate position in list of Direction to move

ad 1, my choice of parameters

ad 2, player_id, max_players, width, height
=> # of players

ad 3, round

ad 4, status

ad 5, status

class Status
- .player ≙ player ID
- .x ≙ x-pos. of player
- .y ≙ y-pos. of player
- .health
- .gold
- .map ≙ limited info about map
- .others ≙ limited info about other players
  - -> list containing class Status of other visible players
- .goldPots ≙ dictionary: (x, y) => amount of gold

- .params ≙ info about GameParameters
  - maxNumGoldPots
  - initial Gold Pot Amount
  - gold Per Round
  - gold Pot Time Out
  - gold Decrease
  - gold Decrease Time
  - move Timeout
  - mine Expiry Time
  - health Per Round
  - min Move Health
  - max Health
  - visibility
  - health Per Wall Crash
  - –"– Player Crash
  - –"– Random
  - ~cost

class Map
- width ≙ max length x-coordinate (note: pos. starting from 0)
- height ≙ vice versa for y-coordinate
- _data ≙ list of Tile(TileStatus()) for each square on the map
  => accessed via) .map[x, y]
  - .status ≙ status of Tile, i.e.
  - .obj ≙ object on Tile, i.e.

class TileStatus(Enum):

Unknown	= 0	"_"
Empty	= 1	"."
Wall	= 2	"#"
Mine	= 3	"&"

class TileObject(object):
```
__init__(i):
self._i = i
```
=> i ≥ 0 ≙ player where i ≙ PID
=> i = -1 ≙ gold Pot