Robots

提出日　2018 年 6 月 19 日

15-412　及川 裕公

1.　Robots とは

古くから UNIX オペレーティングシステムに付属してきた由緒正しきゲームである。

プレイヤーは悪いロボットと戦っている。戦っている様子は以下の図 1 のようなゲームフィールドに、プレイヤーが’@’、ロボットが’+‘で表されている。

凡例:　‘@’...プレイヤー、’+’...ロボット、’-‘,’|’...壁

図 1　Robots のゲームボードのイメージ図

1.1　プレイヤーの操作

プレイヤーは 1 回の動作で、上下左右と斜めの 8 方向、同じ場所で待機、ランダムな場所にテレポートの合計 10 種類の動作が選択可能である。

ただし、以下の 3 つの条件のいずれかが当てはまる地点には、移動することができない。これはテレポートも同様で、以下の条件に当てはまらない場所からランダムに移動となる。

・　フィールドの外
・　ロボットがすでにいる場所
・　スクラップがすでにある場所

プレイヤーが移動した座標に、ロボットが重なったらゲームオーバーである。

プレイヤーの操作は動く方向を 0～9 の数字で入力する。この数字と方向は、プレイヤーがテンキーの 5 の位置にいたと仮定したときの向きにそれぞれ対応している。

0: テレポート
1: 左下
2: 下
3: 右下
4: 左
5: その場で待機

6: 右
7: 左上
8: 上
9: 右上

1.2　ロボットの動き

ロボットはプレイヤーの移動後に行動を開始する。

ロボットは、プレイヤーが移動した後の座標を基に、最接近するように行動してくる。

そして、ロボット同士が衝突すると、ロボットが壊れスクラップとなる。また、ロボットはスクラップに衝突しても壊れてしまう。スクラップは' * 'で表される。

すべてのロボットをスクラップにすると、ステージクリアとなり、次のステージが現れる。

1.3　ゲームの設定

ゲーム開始時のレベルは 1 である。ステージクリアするごとに、レベルが 1 ずつ上がっていく。

フィールドに設置されるロボットの数は、

Min (レベル * 5, 40)

とする。ただし、Min (x,y)は x, y のうち、小さい方を取る関数である。すなわち、レベル 8 以降のロボット出現数は 40 固定となる。

ロボットを 1 台スクラップにするごとに 1 点が与えられる。また、ステージクリアごとにレベル * 10 点のボーナスが与えられる。

例えば、ステージ 1 をクリアすると、

Min (1 * 5, 40) + 1 * 10 = 5 + 10 = 15

となり、15 点を得られることがわかる。

2.　Robots の使用方法

今回作成した Robots は、ターミナル上でソースコード「robots.py」があるディレクトリまで行き、「$python robots.py」と打ち込んで起動する。

起動したと同時にゲームが始まるので、1.1 で記したキーを入力することでプレイヤーが行動をする。また、操作がわからなくなったら、h キーを打つことでヘルプを閲覧することができる。

3. Robots の機能

今回作成した Robots は、ターミナルから実行することで遊ぶことができる、CLI アプリケーションである。コンティニュー機能を実装したことにより、何度も開く手間がなく遊び続けることができる。

4. Robots のデータ構造

4.1 Game クラス

Game クラスは、ゲームに関する情報をすべて管理し、このプログラムの中核とも言えるクラスである。Game クラスは以下の要素で構成される。

表 1 Board クラスのデータ構造

変数名・メソッド名	機能
height	ゲームフィールドの高さ
width	ゲームフィールドの幅
robot_reft	非スクラップ状態のロボットの数
board	ゲームフィールド
score	現在の得点
level	現在のレベル
MAX_ROBOT ※定数	フィールドに配置できるロボットの最大値
setting()	ゲームフィールドの初期状態を生成する
show()	ゲームフィールドの情報を表示
action()	ゲームフィールドを更新
teleport()	プレイヤーがテレポートする座標の算出

4.2 Object クラス

Object クラスは Game クラスの board の中に配置されるオブジェクトが継承しているクラスである。現在どの座標にいるのかを保持している。

4.3 MoveObject クラス

Objectクラスを継承して作成したクラス。座標の他に、move メソッドを持ち、引数として指定した座標に移動することができるようになっている。

4.4 Player クラス

MoveObject を継承して作成したクラス。Move メソッドに移動の制約が追加され、移動できたか否かを返り値として持てるように変更してある。

Object クラスと Moveobject と Player クラスの要素を以下に示す。

表 2　Object クラス、MoveObject クラス、Player クラスのデータ構造

変数名・メソッド名	詳細
x	オブジェクトが持つ x 座標
y	オブジェクトが特つ y 座標
Object クラスは上記の要素を持つ	
move()	x, y を指定した座標に動かす
MoveObject クラスは上記までの要素を持つ	
Player クラスは、move メソッドをオーバーライドしているだけのため、データ構造上は変わらない。	

4.5　Robot クラス

MoveObject を継承して作成したクラス。sclap という変数を加え、自分自身がスクラップ状態か否かを持てるようにしてある。また、Move メソッドをオーバーライドし、ロボット同士が衝突したかどうかの判定などもできるようになっている。それに伴い、ロボットの状態をスクラップ状態にする kill メソッドも追加した。

以下に、Robot クラスのデータ構造を示す。

表 3　Robot クラスのデータ構造

変数名・メソッド名	詳細
MoveObject クラスを継承しているため、基本は表 2 と同じである。ここでは、変わった部分のみを示す。	
scrap	自分がスクラップであるかを持つ
kill()	ロボットをスクラップにする

4.6　Getch クラス

Getch クラスは、Enter キーの入力無しで入力処理を行えるようにしたクラスである。

以下に Getch クラスのデータ構造を示す。

表 4　Getch クラスのデータ構造

変数名・メソッド名	詳細
impl	呼ばれたときに行う処理
call()	OS に合わせた 1 文字入力の処理

5.　主要なクラス・関数の仕様

以下に、主要なクラス・関数の仕様を示す。その時、変数名が()で囲われているものは、省略可能であることを示している。

5.1　Board クラス

5.1.1　コンストラクタ

・引数:(level)…ゲーム開始時のレベル、(height)…ゲームフィールドの高さ、(width)…ゲームフィールドの幅

・機能:ゲームに関する初期設定を行う。

・返り値:なし

5.1.2　board_element メソッド

・引数:pos…ゲームフィールド上の座標

・機能:座標を渡すと、その座標上にいるオブジェクトを返す。

・返り値:座標にいたオブジェクト

5.1.3　 setting メソッド

・引数:なし

・機能:ゲームフィールドの初期状態を生成する。

・返り値:なし

5.1.4　show メソッド

・引数:なし

・機能:ゲームフィールドを表示する。

・返り値:なし

5.1.5　teleport メソッド

・引数:なし

・機能:Object がないマス(移動できないマス)を除いた場所からランダムな座標を選択する

・返り値:Object がないランダムなマス

5.1.6　action メソッド

・引数:command…どの数字が入力されたか

・機能:入力されたコマンドに対応した状態を反映させる

・返り値:何かしらのイベントがあるか否かを True, False で返す

5.2　read_command 関数

・引数:game_master…Game クラスのインスタンス

・機能:Enter キー入力無しで、コマンドを読み取る。その際、コマンドが移動以外のコマンドなら弾く。

・返り値:移動するコマンド(0〜9 のいずれか)

5.3　Player クラス

5.3.1　move メソッド

・引数:x…移動先の x 座標または、y…移動先の y 座標、game_master…Game クラスのインスタンス、(absolute)…x,y が絶対指定か相対指定か

・機能:プレイヤーが動ける座標に移動しようとしているなら移動、動けない座標なら移動しない。また、その結果を返す。

・返り値:プレイヤーは正常に移動できたかを True, False で返す。

6.　検証

このプログラムでは、通常終了されるものを除き、ほとんどの入力に耐えられるように設計を行った。検証した内容は、ゲーム中移動する時にどのような入力に耐えられるかである。以下に試した内容とその結果をまとめた表を示す。

表 5　様々な入力とその応答結果

入力した内容	結果
h 以外のアルファベットを入力(z)	Error. Input is integer.
記号などを入力(@)	Error. Input is integer.
h	ヘルプを表示
座標外の入力(-1,-1 に行くような入力)	Move error. Now pos is (-1, -1)
c-c(KeyboardInterrupt)を入力	通常、終了すべきなので、そのまま終了。
c-d(EOF)を入力	通常、終了すべきなので、そのまま終了

また、ゲームオーバーの処理が正常に行われていること、レベルアップの処理が正常に行われていること、レベル 8 以降に登場するロボットの数は 40 固定になっていたことも確認した。

7.　Robots の計算量とメモリ使用量

ここでは、Robots のメインとなる処理にかかる時間と消費するメモリ量について考える。これ以下に出てくる n, m についてはそれぞれ、ロボットの数、フィールドサイズ(マス目の総数)に対応する。また、m_width ならフィールドの幅、m_height なら高さに対応する。

7.1　プレイヤーの移動

プレイヤーの移動は、テレポートを含む場合と含まない場合で大きく変わってくる。

7.1.1　テレポートをする場合

テレポートをする場合、テレポート先の座標を算出するために、teleport 関数内で O(m)の処理が発生し、move 関数内で O(1)の処理が発生する。よって、テレポートをする場合のオーダーは O(m)であると言える。

メモリ使用量は、teleport 関数内で O(m-n)のメモリを消費し、move 関数内で O(1)のメモリを消費している。

7.1.2　テレポートをしない場合

テレポートをしない場合、テレポート先の座標は算出しないため、move 関数内で O(1)の処理が発生するだけである。そのため、テレポートしない場合のオーダーは O(1)であるといえる。

メモリ消費量は move 関数内の O(1)の消費ですむ。

7.2　ロボットの移動

ロボットの移動は、ゲームフィールドを左上から順に参照していき、ロボットがいるなら動かすという処理をしている。そのため、ロボットの探索で O(m)の処理が発生する。

ロボットの move 関数では、移動するべき座標を求めるのに O(1)の処理を、衝突の判定をするのに O(1)の処理をしているため、結果的に move 関数内では O(1)の処理をしていると言える。よって、ロボットの移動のオーダーは O(m)であるといえる。

メモリ消費量は、探索部分では、比較部分の消費だけのため O(1)、移動部分では、座標算出のため、O(n)のメモリを消費している。

7.3　ロボットの衝突

ロボットの衝突は 7.2 でも述べたように O(1)の処理を行っている。よって、ロボットの衝突のオーダーは O(1)であるといえる。

メモリ消費量は、予め生成されていた部分のみを使用し、直接メモリに与える影響はないため、O(0)である。

8.　プログラム

今回作成した Robots を次頁から示す。

リスト 1　robots.py のソースコード

```
import random



# タプルの計算をするプログラム
# 初めに入った数から ope に従い計算していき、結果のタプルを返す。
def tuple_calc(*tuples, ope):
    ans_list = [0] * len(tuples[0])
    first_flg = True
    for tp in tuples:
        for i in range(len(tp)):
            if ope == 'add' or first_flg == True: #初めの数から後ろを引いていくための準備
                ans_list[i] += tp[i]
            elif ope == 'sub':
                ans_list[i] -= tp[i]
        first_flg = False

    return tuple(ans_list)



# Enter 入力無しで入力処理を行うクラス。
# Win 版と UNIX 版の 2 種類が用意されていて、import 状況に応じて自動的に分岐される。
class _Getch:
    """Gets a single character from standard input.  Does not echo to the
    screen."""
    def __init__(self):
        try:
            self.impl = _GetchWindows()
        except ImportError:
            self.impl = _GetchUnix()

    #この時、得られるのは b(バイト列)である。
    #そのため、普通の文字として扱うため、.decode()を書いている。
    def __call__(self): return self.impl()
```

```
# UNIX 版
# stdin から 1 文字読み込んで、読み込んだ文字を返す。
class _GetchUnix:
    def __init__(self):
        import tty, sys

    def __call__(self):
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch


# Win 版
# msvcrt 内蔵の関数を呼出して、その値を返すだけ。
class _GetchWindows:
    def __init__(self):
        import msvcrt

    def __call__(self):
        import msvcrt
        return msvcrt.getch().decode()


#  ゲームボードの上に存在するオブジェクトは必ずこのクラスを継承して作る
#  オブジェクトは x,y の座標を持っている。
class Object():
    def __init__(self, x = -1, y = -1):
        self._x, self._y = x, y

    @property
```

```
    def position(self):
        return self._x, self._y

    @position.setter
    def position(self, pos):
        if len(pos) != 2:
            raise "test"
        (x, y) = pos
        self._x, self._y = x, y

    @property
    def x(self):
        return self._x

    @x.setter
    def x(self, x):
        self._x = x

    @property
    def y(self):
        return self._y

    @y.setter
    def y(self, y):
        self._y = y


# ゲームボード上を動き回るクラスはこのクラスを継承する。
class MoveObject(Object):
    def __init__(self, x, y):
        super().__init__(x, y)

    def move(self, x, y):
        self._x = x
        self._y = y
```

```
# MoveObject を継承した Player クラス
#  移動するときに、オブジェクトが存在していたら動けない。
class Player(MoveObject):
    def __init__(self, x, y):
        super().__init__(x, y)

    #  移動したかどうかを TF で返す
    def move(self, x, y, game_master, absolute = False):
        if absolute == False:
            x += self._x
            y += self._y

        if x in range(game_master._width) ₩
        and y in range(game_master._height) ₩
        and type(game_master._before_board[y][x]) != Robot:
            #移動ができる
            game_master._after_board[y][x] = self
            super().move(x, y)
            game_master.player_pos = (y, x)
            return True

        else:
            print('move error. now pos is ' + str((x, y)))
            return False



class Robot(MoveObject):
    def __init__(self, x, y):
        super().__init__(x, y)
        self._sclap = False

    @property
    def sclap(self):
        return self._sclap
```

```
#使う予定はないけど、一応用意しておく
@sclap.setter
def sclap(self, boolean):
    self._sclap = boolean

# 返り値として、GameOver か否かを返す。
def move(self, game_master):
    if self._sclap == False:
        #移動する座標の算出
        pos = []
        for p, e in zip(game_master.player_pos, (self._y, self._x)):
            if p - e > 0:
                pos.append(1)
            elif p - e == 0:
                pos.append(0)
            else:
                pos.append(-1)

        y = self._y + pos[0]
        x = self._x + pos[1]

        if type(game_master.after_board[y][x]) == Robot:
            #collision
            self.kill(game_master)
            game_master.after_board[y][x].kill(game_master)

        elif type(game_master.after_board[y][x]) == Player:
            #gameover
            return True

        game_master.after_board[y][x] = self
        super().move(x, y)

    else:
        #先に移動してきたロボットの処理
```

```
            if type(game_master.after_board[self._y][self._x]) == Robot:
                game_master.after_board[self._y][self._x].kill(game_master)

            #スクラップだから動かない
            game_master.after_board[self._y][self._x] = self

    #まだスクラップになっていないなら、ロボ残数を減らし、スクラップ状態にする。
    #del とかを使って書いたほうが良かったかも
    def kill(self, game_master):
        if self._sclap == False:
            self._sclap = True
            game_master.robot_left -= 1
            game_master.score += 1


# ゲームの中心となる情報をすべて持っているクラス。
# 基本的に、この中からゲームを操作する。
class Game():
    def __init__(self, level = 1, height = 10, width = 10):
        self._level = level
        self._height = height
        self._width = width
        self._robot_left = level * 5
        self._board = [[None for i in range(height)] for j in range(width)]
        self._before_board = list(self._board)
        self._after_board = list(self._board)
        self._score = 0
        self._MAX_ROBOT = 40

    @property
    def level(self):
        return self._level

    @level.setter
    def level(self, level):
        self._level = level
```

```
        #残りロボ数も更新
        self._robot_left = level * 5
        if self._robot_left > self._MAX_ROBOT:
            self._robot_left = self._MAX_ROBOT

    @property
    def score(self):
        return self._score

    @score.setter
    def score(self, num):
        self._score = num

    @property
    def height(self):
        return self._height

    @height.setter
    def height(self, height):
        self._height = height

    @property
    def width(self):
        return self._width

    @width.setter
    def width(self, width):
        self._width = width

    @property
    def robot_left(self):
        return self._robot_left

    @robot_left.setter
    def robot_left(self, robot_left):
        self._robot_left = robot_left
```

```
@property
def board(self):
    return self._board

@property
def before_board(self):
    return self._before_board

@property
def after_board(self):
    return self._after_board

@property
def player_pos(self):
    return self._player_pos

@player_pos.setter
def player_pos(self, tp):
    self._player_pos = tp

#引数で(y, x)の順で座標を渡すと、その座標に何がいるかを返す関数
def board_element(self, pos):
    (y, x) = pos
    return self._board[y][x]

#ゲームの初期状態を生成するクラス。
#レベルアップごとに呼び出される。
def setting(self):
    self._board = [[None for i in range(self._height)] for j in range(self._width)]
    temp_list = []
    center_x = self._width // 2
    center_y = self._height // 2
    center = (center_y, center_x)
    for y in range(self._height):
        for x in range(self._width):
```

```
                if (y, x) != center:
                    temp_list.append((y, x))
        #it = itertools.product(range(self._height), range(self._width))
        #temp_list = [e for e in it if e != center]

        random.shuffle(temp_list)
        for y, x in temp_list[:self._robot_left]:
            self._board[y][x] = Robot(y = y, x = x)

        self._board[center_y][center_x] = Player(y = center_y, x = center_x)
        self._player_pos = (center_y, center_x)

    def show(self):
        #ゲームボード表示
        bug_flag = False
        print('-' * (self._width + 2))
        for y in range(self._height):
            print('|', end = '')
            for x in range(self._width):
                if self._board[y][x] is None:
                    print(' ', end = '')
                elif type(self._board[y][x]) == Robot:
                    if self._board[y][x].sclap == True:
                        print('*', end = '')
                    else:
                        print('+', end = '')
                elif type(self._board[y][x]) == Player:
                    print('@', end = '')
                else:
                    #本来ありえない表示
                    print('#', end = '')
                    bug_flag = True

            print('|')

        print('-' * (self._width + 2))
```

```
        if bug_flag == True:
            #あり得ない表示が出た時のメッセージ
            print('予期せぬエラーが発生しています。')
            print('直ちに開発者に連絡してください。')

        #ゲームステータス表示
        print('lv:'+str(self._level)+', score:'+str(self._score))

    #ロボットがいない場所からランダムに 1 地点を選び、座標を(y, x)の順で返す関数
    def teleport(self):
        pos_list = []
        for y in range(self._height):
            for x in range(self._width):
                if isinstance(self._board[y][x], Object) == False:
                    pos_list.append((y, x))

        pos = random.choice(pos_list)
        return tuple_calc(pos, self._player_pos, ope='sub')

    #なにかしらのイベントが有る場合、True で返す
    def action(self, command):
        self._before_board = list(self._board) #中身のコピー
        self._after_board = [[None for i in range(self._height)] ₩
        for j in range(self._width)]

        #各オブジェクトの移動
        #プレイヤー
        #(y, x)の順で入れる
        if command == 0:
            tp = self.teleport()
        else:
            tp = (None, None)
        cmd_to_move = [tp,(1,-1),(1,0),(1,1),(0,-1),(0,0),(0,1),(-1,-1),(-1,0),(-1,1)]
        (move_y, move_x) = cmd_to_move[command]
        if self.board_element(self._player_pos).move(move_x, move_y, game_master):
```

```
                #プレイヤーが正常に移動した場合、敵の移動
                for enemy_pos_y in range(self._height):
                    for enemy_pos_x in range(self._width):
                        if type(self.board_element((enemy_pos_y, enemy_pos_x))) == Robot:
                            gameover = self.board_element((enemy_pos_y,
enemy_pos_x)).move(game_master)
                            if gameover == True:
                                #game over!
                                return True
            else:
                #行動せずに終了 = before 状態のまま
                self._after_board = list(self._before_board)

            if self._robot_left <= 0:
                #level up!
                return True

            #最後に after 状態を現在の状態にして終了
            self._board = list(self._after_board)
            #正常終了は False
            return False


#どの方向に動くかを読み取る関数
def read_command(game_master):

    print('press "h" key to open help')
    while True:
        try:
            getch = _Getch()
            x = getch()
            command = int(x)

        except ValueError:
            #h キーならヘルプを表示
            if x == 'h':
```

```
                print('₩nhow to operate')
                print('7 8 9')
                print('4 @ 6')
                print('1 2 3')
                print('₩n0 ... random teleport')
                print('5 ... stand-by')

            #ctr-c、EOF が入力されたら、強制終了
            elif x == '₩x03':
                # ctr-c
                raise KeyboardInterrupt
            elif x == '₩x04':
                # EOF
                raise EOFError


            #数字を読み込んだ直後の末端記号が入ったときは読み飛ばす
            #それ以外の文字は入力範囲外のものだから、エラーメッセージを表示
            elif x != '₩x00':
                print('error. input is integer')

        else:
            #コマンドは 0～9、つまり range(10)
            if command in range(9+1):
                break
            else:
                print('error. range is 1-9')

    return command


#ゲーム開始から終了までを司る関数
def main_game(game_master):
    game_master.setting()
    while True:
        game_master.show()
```

```
            command = read_command(game_master)
            event = game_master.action(command)
            if event == True:
                if game_master.robot_left <= 0:
                    #level up
                    print('CLEAR!  LEVEL' + str(game_master.level) + ' => ' +
str(game_master.level + 1))
                    game_master.score += game_master.level * 10
                    game_master.level += 1
                    game_master.setting()

                else:
                    #game over
                    print('GAME OVER')
                    return None


#ゲームに必要な前準備を書いておく
#今回みたいに 1 行だったら関数にする必要ないが、拡張したときにわかりやすくなるように
def init_game():
    game_master = Game()
    return game_master


#ゲーム終了後の処理を行う関数
#主に、続けてプレイするかを聞いている。
def postprocess_game():
    print('continue? y/n')
    while True:
        getch = _Getch()
        command = getch()
        if command in {'y', 'n'}:
            break

    return command
```

```
if __name__ == '__main__':

    game_master = init_game()
    while True:
        main_game(game_master)
        cont = postprocess_game()

        if cont == 'n':
            break
```