

Laboratorio 09

Programmazione - CdS Matematica

Michele Donini
20 Gennaio 2015

Esercizio - Forza 4 con I.A.

- Partendo da una struttura già fissata implementare le parti mancanti in modo tale che Forza 4 funzioni correttamente.
- Due giocatori: uno umano (p) e uno artificiale (c).

Esercizio - Forza 4 con I.A.

- Partendo da una struttura già fissata implementare le parti mancanti in modo tale che Forza 4 funzioni correttamente.
- Due giocatori: uno umano (p) e uno artificiale (c).
- Scaricare il file e osservare la struttura dei metodi e il loro funzionamento (alcuni metodi sono già completati).

```
http://www.math.unipd.it/~mdonini/didattica/forza4.py
```

Iniziamo vedendo la definizione del costruttore per la classe Forza4:

```
class Forza4:
    def __init__(self, righe, colonne):
        self.numero_righe = righe
        self.numero_colonne = colonne
        # La scacchiera rappresentata come una matrice:
        self.scacchiera = [[' ']*self.numero_colonne for
                             i in range(self.numero_righe)]
        # Numero di pedine in una certa colonna:
        self.pedine = [0]*self.numero_colonne
```

- Ordine del completamento dei metodi:

- 1 mossa_valida
- 2 mosse_legali
- 3 mossa_player
- 4 mossa_random_cpu
- 5 conta_vert
- 6 conta_oriz
- 7 conta_diagA
- 8 conta_diagB
- 9 mossa_IA_cpu

Soluzione 1

Completamento del metodo `mossa_valida`: ritorna vero se la scelta della colonna `c` risulta essere una mossa **valida**.

Soluzione 1

Completamento del metodo `mossa_valida`: ritorna vero se la scelta della colonna `c` risulta essere una mossa valida.

```
def mossa_valida(self, c):  
    # usa valutazione short-circuit  
    return 0<=c<self.numero_colonne and  
           self.pedine[c]<self.numero_righe
```

Soluzione 2

Completamento del metodo `mosse_legali`: ritorna una lista contenente le colonne valide.

Soluzione 2

Completamento del metodo `mosse_legali`: ritorna una lista contenente le colonne valide.

```
def mosse_legali(self):  
    return [c for c in range(self.numero_colonne)  
            if self.pedine[c]<self.numero_righe]
```

Soluzione 3

Completamento del metodo `mossa_player`: chiede in input una mossa all'utente fin quando non viene inserita una `mossa_valida`. Usa il metodo `esegui_mossa` per renderla effettivamente eseguita dal giocatore 'p'. Ritorna la mossa.

Soluzione 3

Completamento del metodo `mossa_player`: chiede in input una mossa all'utente fin quando non viene inserita una `mossa_valida`. Usa il metodo `esegui_mossa` per renderla effettivamente eseguita dal giocatore 'p'. Ritorna la mossa.

```
def mossa_player(self):  
    print 'Sta a te.',  
    mossa = -1  
    while not self.mossa_valida(mossa):  
        mossa = int(raw_input('Inserire colonna:'))  
    self.esegui_mossa(mossa, 'p')  
    return mossa
```

Soluzione 4

Completamento del metodo `mossa_random_cpu`: sceglie una colonna tra le mosse_legali, la esegue (`esegui_mossa`) per il giocatore 'c'. Ritorna la mossa.

Soluzione 4

Completamento del metodo `mossa_random_cpu`: sceglie una colonna tra le mosse `legali`, la esegue (`esegui_mossa`) per il giocatore `'c'`. Ritorna la mossa.

```
def mossa_random_cpu(self):  
    legali = self.mosse_legali()  
    mossa = random.choice(legali)  
    self.esegui_mossa(mossa, 'c')  
    print 'Io muovo su ', mossa  
    return mossa
```

Soluzione 5

Completamento del metodo `conta_vert`: conta il numero di pedine in sequenza consecutiva in direzione verticale attorno alla pedina inserita nella colonna `c` dal giocatore `g`. Ritorna il numero di pedine in sequenza.

Soluzione 5

Completamento del metodo `conta_vert`: conta il numero di pedine in sequenza consecutiva in direzione verticale attorno alla pedina inserita nella colonna `c` dal giocatore `g`. Ritorna il numero di pedine in sequenza.

```
def conta_vert(self, g, c):  
    # pedine_in_sequenza = numero di pedine g in  
    #   verticale di (self.pedine[c]-1, c)  
    pedine_in_sequenza = 1  
    riga = self.pedine[c]-2  
    colonna = c  
    while riga >= 0 and self.scacchiera[riga][colonna]  
        ] == g:  
        pedine_in_sequenza += 1  
        riga -= 1  
    return pedine_in_sequenza
```

Soluzione 6

Completamento del metodo `conta_oriz`: conta il numero di pedine in sequenza consecutiva in direzione orizzontale attorno alla pedina inserita nella colonna `c` dal giocatore `g`. Ritorna il numero di pedine in sequenza.

Soluzione 6

Completamento del metodo `conta_oriz`: conta il numero di pedine in sequenza consecutiva in direzione orizzontale attorno alla pedina inserita nella colonna `c` dal giocatore `g`. Ritorna il numero di pedine in sequenza.

```
def conta_oriz(self, g, c):  
    # pedine_in_sequenza = numero di pedine g a sx e  
    dx di (self.pedine[c]-1, c)  
    pedine_in_sequenza = 1  
    riga, colonna = self.pedine[c]-1, c-1  
    while colonna >= 0 and self.scacchiera[riga][  
        colonna] == g:  
        pedine_in_sequenza += 1  
        colonna -= 1  
    colonna = c + 1  
    while colonna < self.numero_colonne and self.  
        scacchiera[riga][colonna] == g:  
        pedine_in_sequenza += 1  
        colonna += 1  
    return pedine_in_sequenza
```

Soluzione 7,8,9

- I metodi `conta_diagA` e `conta_diagB` sono simili a `conta_oriz` (cambia il movimento in cui effettuare la ricerca nei due versi).
- Il metodo `mossa_IA_cpu` può essere modificato e ampliato a piacere.

Soluzione 7,8,9

- I metodi `conta_diagA` e `conta_diagB` sono simili a `conta_oriz` (cambia il movimento in cui effettuare la ricerca nei due versi).
- Il metodo `mossa_IA_cpu` può essere modificato e ampliato a piacere.

```
www.math.unipd.it/~mdonini/didattica/forza4\_completo.py
```