DAMA

Caso di studio di Schiavone Roberto

DOCUMENTAZIONE

Ingegneria della Conoscenza e Sistemi Esperti

ANNO ACCADEMICO 2014/2015

Documentazione utente

Il programma può essere lanciato dal prompt dei comandi con l'istruzione:

```
> python Dama.py
```

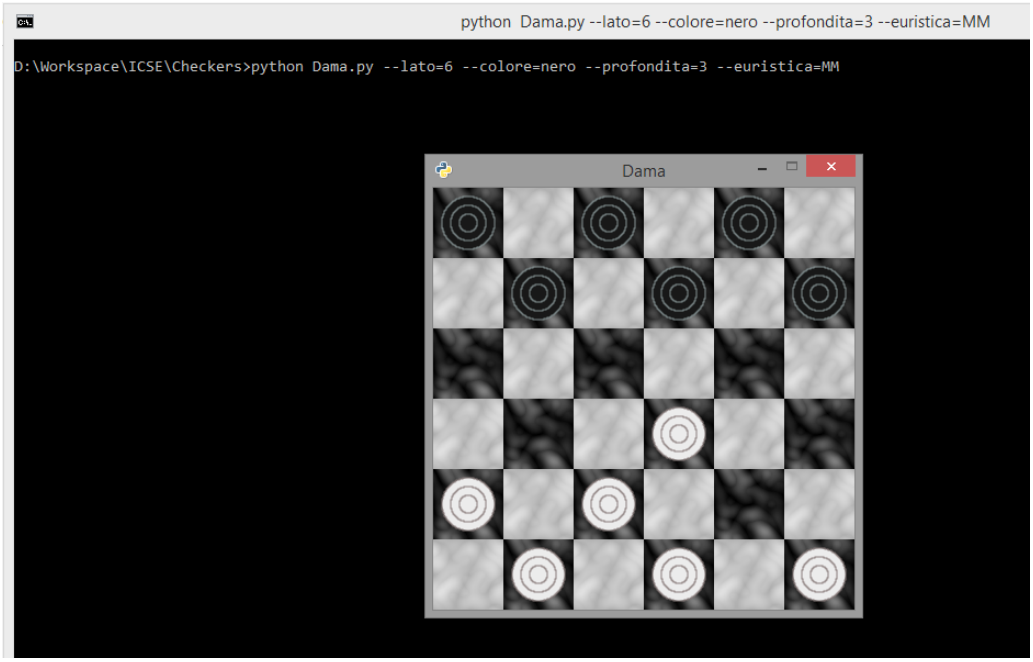
I requisiti minimi da soddisfare sono la presenza della libreria pyglet(un wrapper OpenGL) e driver della propria scheda grafica.

Lanciando il programma da prompt, è possibile passare alcuni parametri per personalizzare l'esperienza di gioco. I parametri disponibili sono i seguenti:

- -h: stampa l'elenco dei parametri possibili;
- --profondita: specifica la profondità dell'euristica;
- --lato: specifica la larghezza e l'altezza della damiera (si consiglia un numero pari per evitare differenze nel numero di pedine bianche e nere);
- --algoritmo: specifica l'algoritmo che utilizzerà l'AI per muovere le proprie pedine, i valori possibili sono "AB" per alpha-beta e "MM" per minimax;
- --colore: specifica il colore con cui giocherà l'utente, i valori sono "bianco" e "nero".

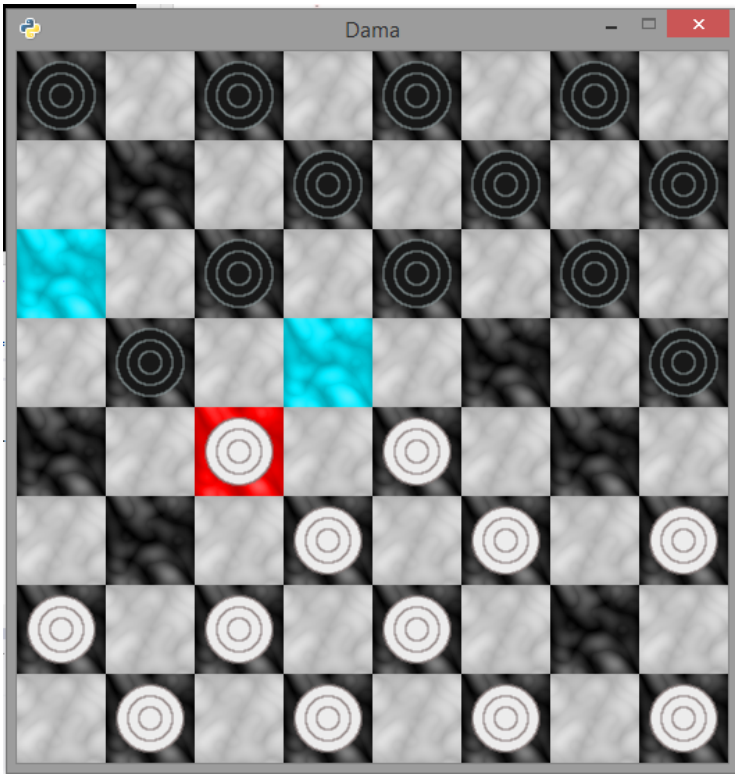
Qualora l'utente decida di non voler passare alcun parametro, il gioco si avvia con i seguenti parametri di default:

- profondita=4;
- lato=6;
- euristica=AB;
- colore=bianco.



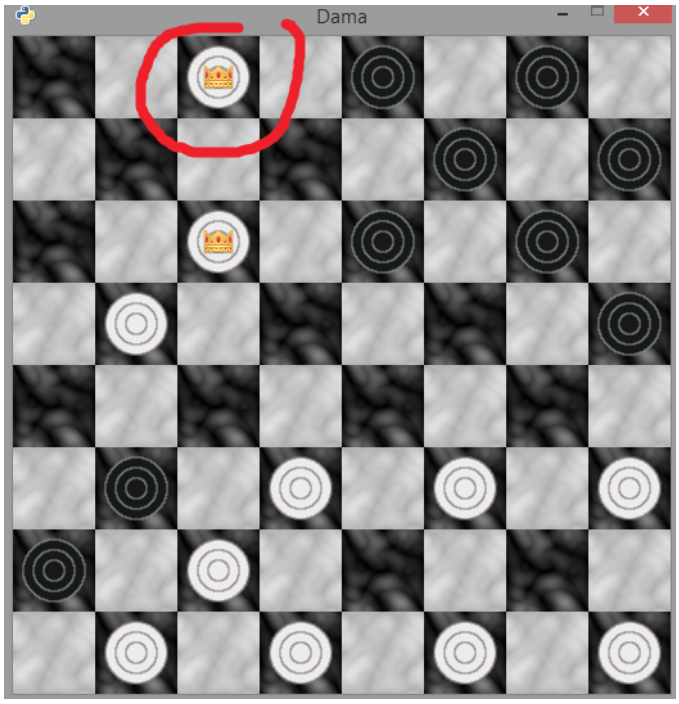
Esempio di avvio personalizzato

Ogni volta che l'utente seleziona una propria pedina da muovere, il gioco evidenzia le possibili mosse consentite:



Il gioco ha il supporto completo per ogni mossa disponibile, una volta che

l'utente o l'AI abbiano raggiunto l'estremo opposto della damiera, la pedina viene promossa a dama:



A questo punto è possibile muoversi in qualunque direzione:

Il gioco implementa una variante della dama tradizionale in cui ad una pedina è concesso di mangiare una dama avversaria.

L'unica condizione di vittoria è eliminare tutte le pedine avversarie; quando le pedine rimaste in gioco sono solo 4, parte un contatore di 30 turni. Quando il contatore si esaurisce, se nessun giocatore è riuscito a vincere in quei 30 turni, il gioco finisce in parità.

Documentazione tecnica

Il gioco implementa due algoritmi diversi per il calcolo della mossa migliore: alpha-beta e minimax.

*Pseudocodice dell'algoritmo **alpha-beta** utilizzato dal gioco.*
Input: **nodo** = mossa, **profondita** = intero maggiore o uguale di zero, **alpha** e **beta** numeri rispettivamente molto grande e molto piccolo, **turno** = flag per gestire la situazione di massimizzazione o minimizzazione del punteggio

```
01. if profondita := 0 then
02.     mossa_ai := nodo;
03.     return valuta_mossa(nodo, turno);
04.
05. mosse_consentite := calcola_mosse_consentite();
06.
07. if turno := turno_corrente then
08.     risultato := -infinito;
09.
10.     for mossa in mosse_consentite
11.         risultato := max(risultato, alpha_beta(mossa, profondita-1,
                                                   alpha, beta, false));
12.         alpha := max(alpha, risultato);

13.         if beta <= alpha then
14.             break;
15.     endfor

16. else
17.     risultato := infinito;

18.     for mossa in mosse_consentite:
19.         risultato := min(risultato, self.alpha_beta(mossa, profondita-1,
                                                   alpha, beta, true));
20.         beta := min(beta, risultato);

21.         if beta <= alpha then
22.             break;
23.     endfor

24. return risultato;
```

*Pseudocodice dell'algoritmo **minimax** utilizzato dal gioco.*
Input: **nodo** = mossa, **profondita** = intero maggiore o uguale di zero, **massimizza** = flag per gestire la situazione di massimizzazione o minimizzazione del punteggio

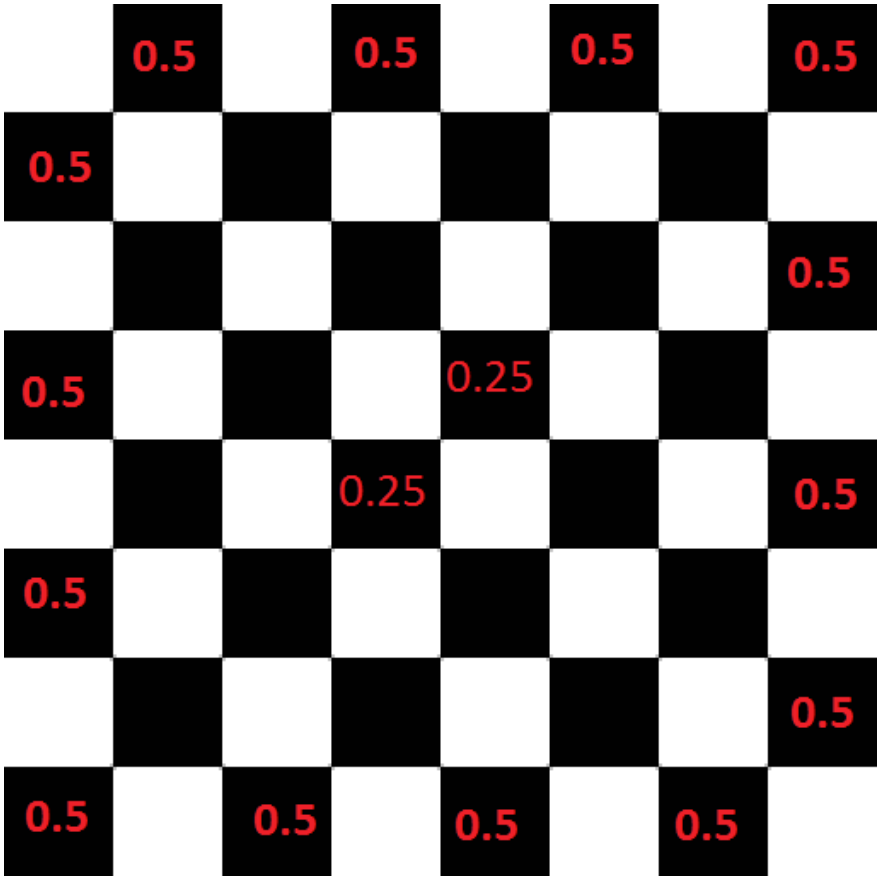
```
01. if profondita := 0 then
02.     mossa_ai := nodo;
03.      return valuta_mossa(nodo, turno);

04. mosse_consentite = calcola_mosse_consentite();

05. if massimizza := true then
06.     risultato := -infinito;

07.     for mossa in mosse_consentite
08.         valore := minimax(mossa, profondita-1, false);

09.         risultato := max(risultato, valore);
10.     endfor
11.     return risultato;

12. else
13.     risultato := infinito;

14.     for mossa in mosse_consentite
15.         valore := minimax(mossa, profondita-1, true);

16.         risultato := min(risultato, valore);
17.     endfor
18.     return risultato;
```

La funzione euristica calcola il punteggio nel seguente modo:

- per ogni pedina avversaria, sottrae 1 al punteggio finale;
- per ogni dama avversaria, sottrae 2 al punteggio finale;
- per ogni pedina propria, aggiunge 1 al punteggio finale;
- per ogni dama propria, aggiunge 3 al punteggio finale.

La funzione euristica implementa inoltre un rudimentale algoritmo di pesatura, in modo da dar maggiore importanza alle pedine che si trovano ai lati della damiera, che quindi non possono essere mangiate, e alle pedine che controllano il centro, che ha un'importanza strategica per poter penetrare la difesa avversaria.

Lo schema di pesatura è il seguente:



Pseudocodice della funzione euristica utilizzata dal gioco.
Input: **damiera** = matrice contenente tutte le posizioni attuali delle pedine, **turno** = giocatore corrente in base al quale calcolare il punteggio

```
risultato := 0;

for pedina in damiera
    if pedina_propria(pedina) then

        risultato := risultato + 1;

    else
        if dama_propria(pedina) then

            risultato := risultato + 3;

    else
        if pedina_avversaria(pedina) then

            risultato := risultato – 1;

    else
        if dama_avversaria(pedina) then

            risultato := risultato – 2;

    if pedina_propria(pedina) or dama_propria(pedina) then
        if pedina.x = bordo or pedina.y = bordo then
            risultato := risultato + 0.5;
        else
            if pedina.x = centro or pedina.y = centro then
                risultato := risultato + 0.25;

return risultato;
```