Gabriele Palazzolo (Mat:883434) Michela Rivautella (Mat:881496)

Relazione Progetto Sistemi Operativi a.a. 2019/20

Di seguito vengono descritte le modalità di compilazione, esecuzione e le strategie intraprese.

1. *Come compilare*

Per questo progetto sono stati implementati due metodi per la compilazione.

Utility make:

attraverso questo programma è possibile compilare solo i moduli per i quali non è già presente il file oggetto. Quindi nel file **makefile** vengono specificate delle regole di dipendenza e di interpretazione:

```
CFLAGS = -std=c89 -pedantic

output: master.c player.c pawn.c common.o semaphore.o
  gcc common.o semaphore.o master.c $(CFLAGS) -o master
  gcc common.o semaphore.o player.c $(CFLAGS) -o player
  gcc common.o semaphore.o pawn.c $(CFLAGS) -o pawn

common.o: common.c common.h
  gcc common.c -c

semaphore.o: semaphore.c semaphore.h
  gcc semaphore.c -c
```

Frammento di codice di makefile

In questo modo vengono assemblate le librerie e i file sorgente, creati i file eseguibili e rinominati.

La compilazione viene effettuata secondo lo standard, più restrittivo, **-std=c89 -pedantic**.

Per compilare digitare sulla shell il comando **make**.

Bash scripting:

questa modalità non permette di effettuare una compilazione di determinati moduli, tuttavia risulta comoda e intuitiva al fine di compilare ed eseguire il codice. La compilazione di tutti i file sorgente avviene in automatico all'interno del file **esegui.sh**, attraverso le righe di codice:

```
#compila common.c
gcc -std=c89 -pedantic common.c -c
#compila semaphore.c
gcc -std=c89 -pedantic semaphore.c -c
#compila pedine
gcc common.c semaphore.c pawn.c -std=c89 -pedantic -o pawn
#compila giocatori
gcc common.c semaphore.c player.c -std=c89 -pedantic -o player
#compila master
gcc common.c semaphore.c master.c -std=c89 -pedantic -o master
```

Frammento di codice di esegui.sh

In questo modo vengono assemblate le librerie e i file sorgente, creati i file eseguibili e rinominati.

La compilazione viene effettuata secondo lo standard, più restrittivo, **-std=c89 -pedantic**.

2. *Come eseguire*

Per questo progetto sono stati implementati due metodi per l'esecuzione.

Utility make:

l'esecuzione effettiva del codice avviene in seguito alla scelta dei parametri e quindi con l'invocazione del file eseguibile **master** e il caricamento del **pathname** del file. Si noti che attraverso questo programma un cambiamento dei parametri di gioco non prevede la ricompilazione dei file sorgente.

Passi da seguire per avviare il gioco:

I. Scrivere sul terminale **./master Parameters/NomeFile.ini** dove **NomeFile.ini** indica quale dei file leggere tra **ParametersEasy.ini**, **ParametersHard.ini**, **ParametersCustom.ini**, i quali contengono valori diversi a seconda della "complessità del gioco" che si vuole configurare:

1. **ParametersEasy.ini**: contiene parametri semplici, in generale i parametri hanno valori bassi (2 giocatori con 10 pedine ciascuno);
2. **ParametersHard.ini**: contiene parametri complessi, in generale i parametri hanno valori semi-elevati (4 giocatori con 400 pedine ciascuno);
3. **ParametersCustom.ini**: contiene parametri intermedi, viene usato questo file per effettuare test personalizzati (di default: 5 giocatori con 10 pedine ciascuno).

II. Premere **Invio** per avviare il gioco.

Bash scripting:

l'esecuzione effettiva del codice avviene in seguito alla scelta dei parametri e quindi con l'invocazione del file eseguibile **master** e il caricamento del **pathname** del file:

```
#esegui
echo "Scegli il file da usare per i parametri: "
echo "1 - 'ParametersEasy.ini' "
echo "2 - 'ParametersHard.ini' "
echo "3 - 'ParametersCustom.ini' "
echo "Inserisci numero: "
read i
if [ $i == 1 ]
then
  ./master Parameters/ParametersEasy.ini
fi
if [ $i == 2 ]
then
  ./master Parameters/ParametersHard.ini
fi
if [ $i == 3 ]
then
  ./master Parameters/ParametersCustom.ini
fi
```

Frammento di codice di esegui.sh

Passi da seguire per avviare il gioco:

III. prima di effettuare l'esecuzione, se necessario, dare i permessi di lettura ed esecuzione dei file .ini con il comando chmod +rx esegui.sh.

IV. Scrivere sul terminale ./esegui.sh per avviare la lettura dei parametri e successivamente l'inizio del gioco.

V. Inserire il numero corrispondente per indicare quale dei file leggere tra ParametersEasy.ini, ParametersHard.ini, ParametersCustom.ini, i quali contengono valori diversi a seconda della "complessità del gioco" che si vuole configurare:

1. ParametersEasy.ini: contiene parametri semplici, in generale i parametri hanno valori bassi (2 giocatori con 10 pedine ciascuno);
2. ParametersHard.ini: contiene parametri complessi, in generale i parametri hanno valori semi-elevati (4 giocatori con 400 pedine ciascuno);
3. ParametersCustom.ini: contiene parametri intermedi, viene usato questo file per effettuare test personalizzati (di default: 5 giocatori con 10 pedine ciascuno).

VI. Premere Invio per avviare il gioco.

3. *Tempi e risultati ottenuti*

La seguente tabella riassume i risultati ottenuti effettuando una media di tali valori su 10 test (esecuzioni):

	Parametri "Easy"	*Parametri "Hard"*
Numero round effettuati	7 ca.	727
Punteggio vincitore	48 ca.	36520 ca.
Mosse usate vincitore	85 ca.	13170 ca.
Tempo di gioco	6.1 sec	2 min 42 sec

Macchina utilizzata per effettuare le statistiche:

Modello: Notebook, Acer Aspire E15

Processore: Intel Core i3-5005U (2 GHz)

Memoria: 4GB DDR3 L Memory

4. *Organizzazione e notazioni adottate*

Organizzazione:

- per effettuare la compilazione e l'esecuzione si è scelto di realizzare un file bash (esegui.sh), in quanto si è ritenuto che l'uso di tale linguaggio di scripting possa essere adatto all'automatizzazione di questo tipo di programmi. Tuttavia, al fine di evitare la compilazione ridondante di alcuni file sorgente si è scelto di utilizzare l'utility make e quindi di realizzare il file makefile.
- Per impostare i parametri di configurazione si è scelto di utilizzare dei file .ini, perché è un formato di file testuale utilizzato da numerosi programmi per la memorizzazione delle opzioni di funzionamento dei programmi stessi e perché si è adottata la sintassi classica di tale estensione per i commenti (uso del carattere #).

- I tre file per la configurazione dei parametri di gioco sono stati raccolti in una cartella Parameters per comodità.

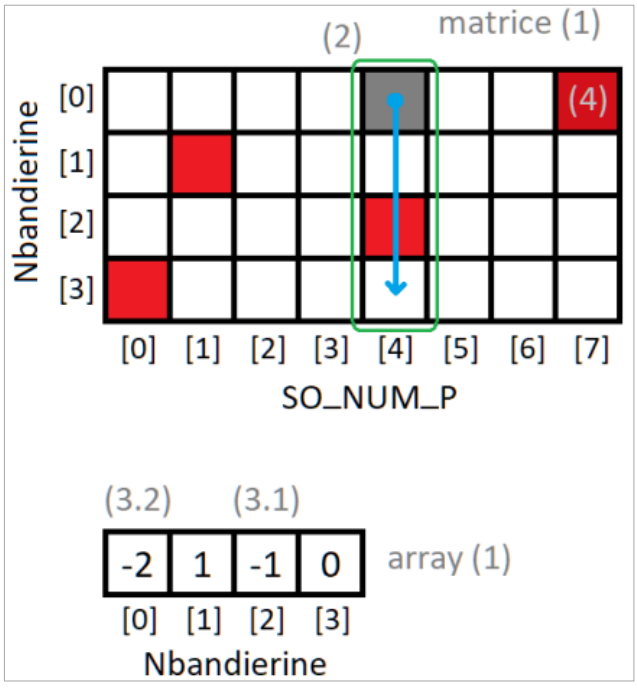
Notazioni:

- i nomi dei metodi presenti nel codice assumono uno stile a *gobbaDiCammello* e sono in lingua italiana, mentre tutti i metodi dedicati alla gestione delle risorse IPC sono in lingua inglese, in quanto questi sono di carattere generale (definiti nelle librerie).
- I metodi scegliDirezioneESposta() e sposta() presenti nel file pawn.c possono apparire simili, ma il primo contiene la parola *sposta* solo perché invoca il secondo metodo che effettua effettivamente lo spostamento di una pedina. Complessivamente, vuole quindi indicare che, il fulcro dedicato ai movimenti della pedina è descritto in questi due blocchi di codice. I due metodi risultano fortemente dipendenti, in quanto il primo effettua un *reserve* sulla cella della scacchiera (MC) su cui spostarsi, mentre il secondo effettua una *release* sulla cella della scacchiera (MC) attualmente occupata.

5. Strategie scelte

Algoritmi nel file player.c:

- inizializzaDestinazionePedina(): per la realizzazione di questo algoritmo si fa uso di una struttura dati monodimensionale e una bidimensionale, la prima (di dimensione Nbandierine) per tenere traccia della pedina più efficiente e la sua posizione corrisponde alla bandierina desiderata. La seconda, invece è di dimensione Nbandierine*SO_NUM_P e si occupa di tenerne in memoria le distanze pedina-bandierina. (1)



Per ogni bandierina si verifica che la pedina considerata sia effettivamente più vicina rispetto alle altre bandierine non ancora assegnate (movimento in verticale). (2)

Se la bandierina considerata è la più vicina alla pedina allora nell'array viene salvato il valore -1 in corrispondenza della bandierina stessa, per indicare l'assegnamento. (3.1)

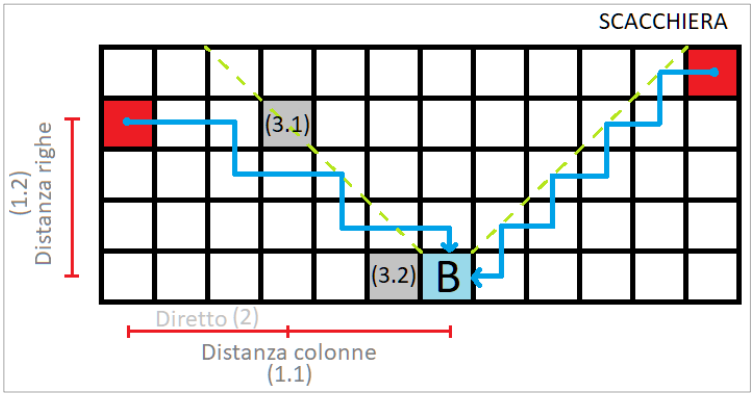
Invece, se la pedina è più vicina a un'altra bandierina allora nell'array viene salvato il valore -1 in corrispondenza di quest'altra bandierina (come 3.1), mentre in corrispondenza della prima bandierina viene settato il valore -2 per indicare che nessuna pedina è ancora stata scelta. Ne segue che la pedina non verrà più considerata nelle successive iterazioni di assegnamento. (3.2)

Al fine di attribuire a ogni bandierina una pedina, si ricalcola la distanza minore tra le pedine rimanenti. (4)

Algoritmi nel file pawn.c:

- controllaDestinazione(): con questo metodo ogni pedina verifica se non è stata assegnata a una bandierina dal suo giocatore oppure le mosse disponibili non sono sufficienti per catturarla.

- `confrontaDestinazione()`: questo metodo permette di confrontare le destinazioni delle pedine di un giocatore al fine di ridurre il più possibile le mosse da usare. Quindi, se la pedina non è stata assegnata a una bandierina o non ha mosse sufficienti, cerca la bandierina più vicina. Se ha abbastanza mosse, confronta la sua distanza con quella della pedina che attualmente sta puntando alla bandierina. Nel caso in cui tale distanza è minore, questa pedina avrà come destinazione la bandierina, mentre per l'altra viene resettata la destinazione, che potrà dunque cercare una nuova meta.
- `scegliDirezioneESposta()`: questo metodo viene implementato per assegnare una modalità di spostamento alle pedine. Si è scelto quindi di effettuare un movimento diagonale ottenuto secondo spostamenti in orizzontale e in verticale. Si è notato infatti che questo tipo di scorrimento permette di evitare, con maggiore facilità, le altre pedine con conseguente minor numero di blocchi.



La pedina calcola la distanza in orizzontale (1.1) e in verticale (1.2) rispetto alla bandierina a cui punta. Quindi viene calcolata la differenza per scegliere la direzione più conveniente. (2)

Se la differenza trovata è maggiore di 0, allora la pedina tenterà di spostarsi in orizzontale (sinistra/destra a seconda della collocazione della bandierina rispetto alla pedina stessa). Tuttavia, se l'opzione non è possibile, verrà effettuato un controllo per gli spostamenti in verticale. (3.1)

Se la differenza trovata è minore o uguale a 0, allora la pedina tenterà di spostarsi in verticale (alto/basso a seconda della collocazione della bandierina rispetto alla pedina stessa). Tuttavia, se l'opzione non è possibile, verrà effettuato un controllo per gli spostamenti in orizzontale. (3.2)

Nel caso in cui la pedina è bloccata in tutte e quattro le direzioni, effettuerà questi controlli fino a quando non sarà disponibile un'opzione.

- `sposta(move direzione)`: questo metodo permette di effettuare l'effettivo spostamento della pedina. Quindi una volta individuata la direzione da intraprendere verrà aggiornata la scacchiera (MC) e il numero di mosse totali usate dal giocatore (MC), in modalità di mutua esclusione. Nel caso in cui la pedina abbia conquistato una bandierina potrà cercare, nuovamente, un'altra destinazione secondo i metodi precedentemente descritti.

6. *Osservazioni*

Definizione:

si nota che implementare una `#define` di un puntatore a funzione, in questo caso per il valore assoluto, comporta un rallentamento dell'esecuzione degli algoritmi scelti. Viene quindi definita una funzione ad esso dedicata all'interno di una libreria (`common.c`).

Casi limite:

sono stati effettuati alcuni test definibili come *casi limite*, per verificare la correttezza semantica del gioco e fare delle constatazioni:

- SO_NUM_G = 0: se non ci sono giocatori non ha alcun senso avviare il gioco, quindi una volta terminata la lettura dei parametri, se questo parametro non è valido, viene segnalato un errore e terminati tutti i processi attivi.
- SO_NUM_G = 1: se vi è solo un giocatore non ha alcun senso avviare il gioco, in quanto non vi sarebbe alcuna gara, quindi una volta terminata la lettura dei parametri, se questo parametro non è valido, viene segnalato un errore e terminati tutti i processi attivi.
- SO_NUM_P = 1: se il numero di pedine per giocatore è solo una si nota che vengono effettuati pochi round.
- SO_NUM_P = n: dove n<=(SO_BASE*SO_ALTEZZA - SO_FLAG_MAX) / SO_NUM_G. Si nota che viene effettuato un numero alto di round, in quanto le pedine che dovranno muoversi effettueranno pochi movimenti (1-2), mentre altre saranno bloccate in tutte e quattro le direzioni. Si può, inoltre, osservare che il master impiegherà, mediamente, più tempo per piazzare le bandierine, in quanto dovrà generare diverse posizioni prima di trovarne una libera all'interno della scacchiera.

Nota: se si caricano parametri non validi vengono effettuati i controlli sui valori prima di avviare il gioco, ovvero immediatamente dopo la loro lettura.