

# Laboratorio Metodi di Programmazione

A.A. 2005/2006

## CSROBOTS: Specifiche di Progetto

Stefano Zacchioli  
zack@cs.unibo.it

Versione 0.1  
1 Maggio 2006

### 1 Introduzione

Questo documento descrive il progetto che gli studenti del corso di Laboratorio di Metodi di Programmazione dovranno svolgere per l'Anno Accademico 2005/2006.

Scopo del progetto è la realizzazione del server di gioco e di uno o più giocatori per un gioco multiplayer fra robot chiamato CSROBOTS. Le regole del gioco sono liberamente ispirate (ma non identiche!) al gioco proposto nella quinta edizione dell'ICFP Programming Contest [2].

Questo documento è organizzato come segue. In Sezione 2 sono descritte le regole del gioco multiplayer, in Sezione 3 il protocollo di comunicazione tra giocatori e server di gioco. In Sezione 4 è descritto ciò che dovrete effettivamente implementare come progetto ed in Sezione 5 la logistica corrispondente. In Sezione 6 trovate un po' di consigli ed in Sezione 7 la storia delle revisioni di questo documento.

### 2 Regole del gioco

Un numero arbitrario di robot si sfidano su di un terreno comune irto di insidie. Ogni giocatore controlla un robot. Scopo di ogni giocatore è far raccogliere al proprio robot i pacchetti disponibili sul terreno di gioco e farli recapitare a destinazione. Tutti i robot

competono per gli stessi pacchetti, ed ogni pacchetto ha una diversa destinazione.

#### 2.1 Terreno di gioco

I robot si sfidano su un terreno bidimensionale, rettangolare, composto da un numero finito di *caselle*. Ogni casella appartiene ad uno dei seguenti tipi:

- spazio aperto;
- muro;
- acqua;
- casa base.

Un terreno di gioco è *ben formato* se contiene almeno una casella di tipo casa base.

I robot possono camminare su spazi aperti e case base, non posso oltrepassare muri o esservi locati temporaneamente (i.e. le caselle di tipo muro si dicono *solide*), muoiono se entrano nell'acqua (i.e. le caselle di tipo acqua si dicono *letali*). Non è possibile uscire dal terreno di gioco, è come se questo fosse circondato da muri.

Ogni terreno di gioco è caratterizzato da una dimensione: una coppia di numeri naturali positivi *larghezza*  $\times$  *altezza*. Ogni casella del terreno di gioco è indirizzata da una coppia di coordinate (numeri naturali non negativi)  $\langle \text{riga}, \text{colonna} \rangle$ , dove *riga* = 0, ..., *altezza* - 1 e *colonna* = 0, ..., *larghezza* - 1.

Intuitivamente, la casella  $\langle 0, 0 \rangle$  è a nord-ovest del terreno di gioco (si veda Figura 1).

$\uparrow$ N	$\langle 0, 0 \rangle$	$\langle 0, 1 \rangle$	$\langle 0, 2 \rangle$	$\langle 1, 3 \rangle$
	$\langle 1, 0 \rangle$	$\langle 1, 1 \rangle$	$\langle 1, 2 \rangle$	$\langle 1, 3 \rangle$
	$\langle 2, 0 \rangle$	$\langle 2, 1 \rangle$	$\langle 2, 2 \rangle$	$\langle 2, 3 \rangle$

Figura 1: Esempio di terreno di gioco  $4 \times 3$  e sue coordinate

## 2.2 Pacchetti

Ogni pacchetto è caratterizzato dalle seguenti proprietà:

- un identificativo univoco (un intero non-negativo);
- un peso (un intero positivo);
- una destinazione (una coppia di coordinate analoghe a quelle utilizzate per referenziare la caselle del terreno di gioco).

Tutte le proprietà di un pacchetto sono immutabili durante lo svolgimento del gioco.

In ogni istante di gioco i pacchetti sono o trasportati da un robot, o appoggiati su una casella. All'inizio del gioco tutti i pacchetti sono ripartiti tra le case base disponibili.

I pacchetti scompaiono dal gioco in due occasioni:

1. un pacchetto viene lasciato cadere sulla sua destinazione;
2. un robot muore; in questo caso tutti i pacchetti che trasportava scompaiono dal gioco.

Un pacchetto che viene lasciato cadere su una casella diversa dalla sua destinazione rimane su quella casella fino a quando un altro robot non lo raccoglie. Il semplice atto di muovere un robot su di una casella contenente pacchetti (senza raccoglierli) non ha alcun effetto su di essi.

Durante il gioco ogni robot colleziona punti, partendo da un ammontare iniziale di 0 punti. Un pacchetto che viene lasciato cadere sulla sua destinazione fa guadagnare al robot che lo ha consegnato tanti punti quanto è il suo peso.

## 2.3 Robot

Ogni robot è caratterizzato dalle seguenti proprietà:

- un identificativo univoco (un intero non-negativo, immutabile durante il gioco);
- una forza (un intero positivo, immutabile durante il gioco);
- una quantità di € disponibili per fare offerte (mutabile durante il gioco).

In ogni istante di gioco un robot è o morto, o locato su di una casella del terreno di gioco. Nel momento in cui muore scompare dal terreno di gioco.

Il gioco si svolge a turni, ad ogni turno ogni giocatore impartisce al proprio robot uno tra i seguenti comandi:

**Move**  $d$  indica al robot di spostarsi di una casella dalla sua attuale posizione nella direzione  $d$ . Le direzioni possibili corrispondono ai quattro punti cardinali: N (nord), S (sud), E (est), W (ovest). Il tentativo di spostarsi su di una cella solida lascia il robot nella sua attuale posizione;

**Pick**  $p_1, \dots, p_n$  ( $n \geq 0$ ) indica al robot di raccogliere da terra i pacchetti i cui identificativi sono  $p_1, \dots, p_n$ . I pacchetti da raccogliere vengono cercati nella cella ove il robot è attualmente locato. I pacchetti vengono raccolti nell'ordine indicato (i.e.  $p_1$  è il primo ad essere raccolto,  $p_n$  l'ultimo);

**Drop**  $p_1, \dots, p_n$  ( $n \geq 0$ ) indica al robot di lasciare cadere a terra i pacchetti i cui identificativi sono  $p_1, \dots, p_n$ . I pacchetti da lasciare cadere vengono cercati tra quelli attualmente trasportati dal robot.

Ad ogni turno ogni robot esegue il comando impartitogli per quel turno. A causa delle spinte (vedi Sezione 2.4) l'ordine di esecuzione dei comandi durante un singolo turno è rilevante. Tale ordine viene stabilito in base alle offerte effettuate all'inizio del turno (vedi Sezione 2.5).

È possibile richiedere di lasciare cadere a terra pacchetti che non si stavano trasportando (solo quelli effettivamente trasportati verranno lasciati cadere) e di raccogliere da terra pacchetti che non sono presenti sulla casella corrente (solo quelli effettivamente presenti verranno raccolti).

Un robot non può trasportare un insieme di pacchetti il cui peso complessivo superi la sua forza. Un robot che tenta di raccogliere da terra un insieme di pacchetti  $p_1, \dots, p_n$  che gli farebbe violare tale limite viene ucciso. In tale caso i pacchetti  $p_1, \dots, p_n$  vengono lasciati a terra.

Il gioco termina quando tutti i robot sono stati uccisi. Al termine del gioco risulta vincitore il giocatore il cui robot ha guadagnato più punti per la consegna di pacchetti.

## 2.4 A cucci e spintoni

I robot in gioco sono ben lungi dall'essere immateriali (centinaia di kg di ferraglia ognuno). Le caselle sono piccole: ognuna di essere non può ospitare più di un robot alla volta. In compenso i robot sono molto forti e offrono poco attrito allo spostamento. L'unione di questi fattori fa sì che un robot che si sposta su di una casella occupata da un altro lo spinga via, potenzialmente spingendone altri, a meno che un muro non lo impedisca.

Diciamo che *il robot  $r_1$  spinge il robot  $r_2$*  (o, analogamente, che *il robot  $r_2$  viene spinto dal robot  $r_1$* ) quando  $r_1$ , effettuando una mossa MOVE  $d$ , cerca di spostarsi sulla casella attualmente occupata da  $r_2$ . Il risultato tipico di questa situazione è che sia  $r_1$  che  $r_2$  si muovano in direzione  $d$ . Se la casella di arrivo per  $r_2$  è letale,  $r_2$  viene ucciso; se è solida nessuno dei due robot si sposta dalla rispettiva posizione attuale. *N.B.* lo status di “essere spinti” è indipendente dall'effettiva riuscita dello spostamento.

Le spinte sono transitive: l'effetto di “ $r_1$  spinge  $r_2$ ” può causare “ $r_2$  spinge  $r_3$ ”. In questo caso sia  $r_2$  che

$r_3$  sono considerati “spinti” e i 3 robot si spostano solo se tutti e 3 possono farlo (ovvero se non c'è una casella solida nella casella di arrivo di  $r_3$ ).

L'essere spinti ha effetti negativi sul sistema operativo dei robot:<sup>1</sup> un robot che viene spinto entra nello stato di *reboot* e vi rimane fino alla fine del turno corrente. Un robot nello stato di *reboot* non esegue il comando che gli è stata impartito. Ciò implica che un robot non eseguirà il comando impartitogli per un dato turno se viene spinto prima di avere avuto la possibilità di eseguirlo.

Quando un robot che trasporta uno o più pacchetti viene spinto, l'urto fa sì che l'ultimo dei pacchetti che ha raccolto da terra cada. Il pacchetto viene lasciato cadere sulla casella dove il robot si trova al momento della spinta (ovvero prima di essere spostato).

Esempi di cause/effetti relative alle spinte sono riportati in [2], nella Sezione “Pushing and Bidding”.

## 2.5 L'asta tosta

L'ordine di esecuzione delle mosse dei robot in un singolo turno è stabilito in base all'*offerta* che ogni giocatore associa ad un comando quando lo impartisce al proprio robot. Ogni offerta è un intero diverso da 0.

I comandi vengono eseguiti in ordine decrescente di offerta. In caso di pari offerta i comandi vengono eseguiti in ordine di anzianità, ovvero: il robot che per primo ha preso parte al gioco (vedi Sezione 3) agisce per primo, poi il secondo, . . .

Offerte positive aumentano la possibilità di agire prima di altri robot, offerte negative aumentano quella di agire dopo altri robot. Entrambi i tipi di offerta hanno un *costo* pari al valore assoluto di quanto è stato offerto. Il costo di una offerta viene detratto dalla quantità di € del robot a cui è stato impartito il comando. Robot che non agiscono a causa di spinte non vengono rimborsati di quanto hanno offerto.

Robot che effettuano offerte il cui costo è troppo alto (più degli € che hanno a disposizione) o troppo basso (0€) vengono uccisi. Ciò implica che un singolo

<sup>1</sup>trattasi di sistema operativo proprietario, di conseguenza nessuno è ancora riuscito a debuggare il problema e a fornire una patch che lo risolva. La ditta produttrice del sistema operativo non ha risposto ai ripetuti bug report.

gioco può durare al massimo un numero di turni pari alla quantità massima di € disponibili per un robot all'inizio del gioco.

### 3 Protocollo di gioco

Il gioco è distribuito, ogni giocatore si collega ad un server che svolge il ruolo di coordinatore del gioco. Server e giocatori possono essere in esecuzione su elaboratori diversi accessibili via Internet. Ogni giocatore comunica con il server e viceversa, i giocatori non comunicano direttamente tra loro.

Il protocollo di comunicazione è testuale, utilizza solo caratteri ASCII ed è basato su righe (i.e. l'unità minima di comunicazione scambiata tra server e giocatori è una riga di testo). Una *riga* è una sequenza di caratteri diversi da '\n' (codice ASCII 10) seguiti da un carattere '\n'. I numeri vengono spediti come sequenze di caratteri (e.g. il numero -127 viene spedito come la sequenza di caratteri: '-','1','2','7').

Il gioco è diviso in due fasi: inizializzazione e sfida.

#### 3.1 Inizializzazione

Il server viene avviato ed attende che i giocatori prendano parte al gioco. Una volta connessi al server i giocatori prendono parte al gioco secondo il protocollo seguente, dove P→S indica una comunicazione dal giocatore (Player) al server e S→P una comunicazione nella direzione opposta:

1. (P→S) spedisce la linea "player\n";
2. (S→P) spedisce il terreno di gioco;
3. (S→P) spedisce la configurazione del giocatore;
4. (S→P) spedisce il primo update.

##### 3.1.1 Spedizione del terreno di gioco

Il terreno di gioco sul quale si svolge la sfida è un parametro di esecuzione del server e viene letto da file (vedi Sezione 4.3).

Il terreno di gioco viene spedito secondo il protocollo:

1. (S→P) dimensioni del terreno di gioco;
2. (S→P) riga 0;
3. (S→P) riga 1;
- ... (S→P) riga *altezza* - 1.

Ad ogni passo del protocollo corrisponde una riga che viene spedita dal server al giocatore (i.e. dopo avere spedito sia le dimensioni che una riga del terreno di gioco viene spedito uno '\n').

Le dimensioni sono spedite come la coppia di numeri *larghezza* e *altezza*, separati da uno spazio (codice ASCII 32). Ogni riga del terreno di gioco è spedita come una sequenza di caratteri — tanti caratteri quanta è la larghezza del terreno di gioco — che indicano il tipo di casella; '.' (punto) indica uno spazio aperto, '#' (cancelletto) indica un muro, '~' (tilde) indica acqua, '@' (chiocciola) indica una casa base.

Un esempio di spedizione di terreno di gioco è riportato in Figura 2.

```
4 3
...~
@...
..##
```

Figura 2: Esempio di spedizione di un terreno di gioco 4 × 3. I terminatori di riga sono riportati implicitamente andando a capo. Il terreno in questione ha acqua in posizione ⟨0, 3⟩, una casa base in posizione ⟨1, 0⟩ e muri alle posizioni ⟨2, 2⟩ e ⟨2, 3⟩.

##### 3.1.2 Spedizione della configurazione dei giocatori

La configurazione di un giocatore consiste in una tripla composta da: identificativo univoco, forza, € disponibili.

Il server genera identificatori univoci crescenti e li assegna ai giocatori in ordine di arrivo; l'identificativo del primo giocatore che prende parte al gioco è 0, il successivo 1, ... Forza ed € disponibili di ogni robot sono parametri di esecuzione del server e vengono letti da file (vedi Sezione 4.3).

I tre valori della configurazione vengono spediti dal server al giocatore su di un'unica riga, separati da spazi, nell'ordine: identificativo, forza, €.

### 3.1.3 Spedizione degli update

Un *update* è un insieme di eventi che descrive cosa è successo rispetto alla precedente situazione di gioco. Viene spedito dal server ai giocatori sia prima dell'inizio del gioco (*primo update*), che al termine di ogni turno di gioco.

L'update viene spedito su di un'unica riga ed è composto da una lista di *robot update* separati da ';' (punto e virgola). L'inizio e la fine della lista sono delimitati rispettivamente dai caratteri '[' e ']'. Ogni robot update descrive un evento accaduto ad un singolo robot.

Un singolo robot update è codificato dal carattere '#' (cancelletto), l'identificativo del robot, un carattere ',', (virgola), la codifica dell'evento avvenuto. Gli eventi possibili e le loro codifiche sono i seguenti:

<i>evento</i>	<i>stringa</i>
il robot si è mosso in dir. N	"n"
il robot si è mosso in dir. S	"s"
il robot si è mosso in dir. E	"e"
il robot si è mosso in dir. W	"w"
il robot ha raccolto da terra il pacchetto identificato da $p_i$	"p $p_i$ "
il robot ha lasciato cadere a terra il pacchetto identificato da $p_i$	"d $p_i$ "
il robot è apparso alla casella avente coordinate $\langle r, c \rangle$	"r $r$ c $c$ "
il robot è stato ucciso	"k"

Il primo update è costituito da soli eventi di tipo "apparizione".

*Esempio.* Alcuni update possibili:

"#2,e;#1,n"	robot 2 si è mosso verso E, robot 1 si è mosso verso N
"#1,w;#2,p 0;#2,p 1"	robot 2 ha raccolto i pacchetti 0 e 1, robot 1 si è mosso verso W
"#2,e;#1,w;#1,e]"	robot 2 si è mosso verso E, robot 1 si è mosso prima verso W e poi verso E (i.e. robot 1 è stato spinto)
"#1,r 1 c 2;#2,e]"	robot 2 si è mosso verso E, robot 1 è apparso alla casella $\langle 1, 2 \rangle$

## 3.2 Sfida

Ad ogni turno il seguente protocollo viene ripetuto:

1. (S→P) pacchetti disponibili;
2. (P→S) comando;
3. (S→P) update.

All'inizio di ogni turno il server informa ogni giocatore dei pacchetti disponibili alla loro attuale posizione. Il giocatore risponde impartendo un comando al proprio robot e indicando l'offerta corrispondente. Non appena tutti i giocatori hanno impartito comandi ed offerte il server calcola la situazione risultante e spedisce ai giocatori un update (i.e. "cosa è successo durante il turno"). *N.B.* la comunicazione dei pacchetti disponibili è personalizzata, ogni giocatore riceve solo la lista dei pacchetti disponibili alla sua posizione; la comunicazione degli update è invece globale, tutti i giocatori ricevono lo stesso update.

### 3.2.1 Spedizione dei pacchetti disponibili

Tutti i pacchetti disponibili vengono spediti al giocatore su di un'unica riga. Per ogni pacchetto disponibile il server spedisce 4 interi separati da ',', (virgola): l'identificativo del pacchetto, la riga destinazione del pacchetto, la colonna destinazione del pacchetto, il peso del pacchetto. Le sequenze di interi di pacchetti diversi vengono separate da ';' (punto

e virgola). L'inizio e la fine della stringa dei pacchetti disponibili sono delimitati rispettivamente dai caratteri '[' e ']'.  
Esempio. Se sulla casa base di Figura 2 sono locati il pacchetto 13 di peso 5 che deve essere consegnato alla cella  $\langle 0, 1 \rangle$  ed il pacchetto 17 di peso 7 che deve essere consegnato alla cella  $\langle 2, 0 \rangle$ , un giocatore il cui robot è locato sulla casa base riceverà dal server la riga: "[13,0,1,5;17,2,0,7]\n". Se nessun pacchetto è disponibile il player riceverà dal server la riga "[]\n".

*Esempio.* Se sulla casa base di Figura 2 sono locati il pacchetto 13 di peso 5 che deve essere consegnato alla cella  $\langle 0, 1 \rangle$  ed il pacchetto 17 di peso 7 che deve essere consegnato alla cella  $\langle 2, 0 \rangle$ , un giocatore il cui robot è locato sulla casa base riceverà dal server la riga: "[13,0,1,5;17,2,0,7]\n". Se nessun pacchetto è disponibile il player riceverà dal server la riga "[]\n".

### 3.2.2 Spedizione dei comandi

Impartire un comando ad un robot consiste nello specificare il comando ed una offerta per influenzare quando questo verrà eseguito. Il formato di spedizione consiste in un offerta (un numero intero, permettendo il carattere '-' (meno) in caso di numero negativo), uno spazio, ed il comando stesso codificato come segue:

**Move**  $d$  la stringa "move", seguita da uno spazio, seguita dalla codifica della direzione: la stringa "n" per N, la stringa "s" per S, la stringa "e" per E, la stringa "w" per W;

**Pick**  $p_1, \dots, p_n$  la stringa "pick", seguita da uno spazio, seguita dalla lista degli identificatori  $p_1, \dots, p_n$  separati da spazi;

**Drop**  $p_1, \dots, p_n$  la stringa "drop", seguita da uno spazio, seguita dalla lista degli identificatori  $p_1, \dots, p_n$  separati da spazi.

*Esempio.* Questi alcuni esempi di comandi come possono essere spediti da un giocatore al server:

```
1 move n
200 pick 17 89
29 move w
-8 move s
57 drop 11
1 drop
```

## 4 Le vostre fatiche

Il vostro compito per assolvere la parte di progetto del corso di "Laboratorio di Metodi di Programmazione" consiste nell'implementare:

1. il server di coordinazione che implementi le regole (Sezione 2) ed il protocollo di gioco (Sezione 3);
2. uno o più giocatori che partecipino al gioco rispettando il protocollo (Sezione 3).

Il server non è dotato di particolare intelligenza: le regole sono totalmente deterministiche e non è necessario implementare alcun tipo di euristica.

Avete assoluta libertà nell'implementazione del giocatore. La complessità del giocatore può variare dal *giocatore stupido* (vedi Sezione 4.1) ad uno qualsiasi dei giocatori che hanno partecipato all'ICFP Programming Contest del 2002 [2]. Sono descritti alla pagina <http://icfpcontest.cse.ogi.edu/links.html>.

Sia server che giocatori devono essere eseguibili da riga di comando e devono accettare alcuni parametri di configurazione. Il server deve inoltre essere in grado di caricare da file parametri quali il terreno di gioco sul quale si sfidano i robot in un dato gioco e la posizione di pacchetti e robot sullo stesso. I dettagli che riguardano i parametri di configurazione di server e giocatori sono riportati nelle Sezioni 4.2 e 4.3.

Al termine del gioco il server deve stampare a video la classifica (in ordine decrescente) indicante per ogni robot partecipante l'ammontare di punti ottenuti.

L'implementazione deve essere realizzata utilizzando il linguaggio di programmazione Java [1].

### 4.1 Il giocatore stupido

Il giocatore stupido implementa il seguente algoritmo, offrendo 1 ad ogni turno per ogni possibile comando:

1. vai alla prima casa base in una visita topologica del terreno di gioco ( $\langle 0, 0 \rangle$ ,  $\langle 0, 1 \rangle$ ,  $\langle 0, 2 \rangle$ , ...,  $\langle 1, 0 \rangle$ ,  $\langle 1, 1 \rangle$ , ...);

2. prendi tanti pacchetti quanti possibile da quella casa base (nell'ordine in cui il server li ha comunicati, fino a non superare il limite di peso consentito);
3. se hai raccolto almeno un pacchetto portalo a destinazione, ripeti questo punto finché hai pacchetti;
4. ritorna al punto (1.) passando alla casa base successiva;
5. se ho terminato le case base inizio una visita topologica del terreno di gioco a caccia di pacchetti abbandonati dagli avversari.

Ogni percorso scelto per raggiungere una data destinazione è calcolato dal giocatore stupido in modo da evitare celle letali (è stupido, non masochista).

Notate che il giocatore stupido non sfrutta minimamente le informazioni fornitegli dagli update del server (ignora gli avversari che possono disturbare le sue traiettorie, non tiene traccia di dove questi hanno lasciato cadere pacchetti, ecc) ... in fondo è stupido!

## 4.2 Parametri dei giocatori

L'unica informazione di cui i giocatori necessitano per partecipare ad un gioco è la locazione su Internet del server che coordina il gioco. Tale informazione viene fornita mediante due parametri che devono essere specificabili a riga di comando:

<i>Parametro</i>	<i>Descrizione</i>
<code>-address</code>	indirizzo IP del server
<code>-port</code>	porta TCP alla quale il server accetta connessioni di giocatori

*Esempio.* Una possibile riga di comando per invocare un giocatore indicandogli di collegarsi al server in ascolto sull'elaboratore corrente alla porta 7919 è la seguente:

```
java it.unibo.cs.csrobots.DumbPlayer \
  -address 127.0.0.1 -port 7919
```

## 4.3 Parametri del server

Il server di gioco deve accettare a riga di comando i parametri duali rispetto ai giocatori più un parametro addizionale che descriva le condizioni nelle quali si svolgerà il gioco:

<i>Parametro</i>	<i>Descrizione</i>
<code>-address</code>	indirizzo IP del server
<code>-port</code>	porta TCP alla quale il server accetta connessioni di giocatori
<code>-board</code>	nome base del file dal quale caricare il terreno di gioco

Il parametro `-board` specifica un file il cui contenuto descrive il terreno di gioco. Il formato di tale file è simile al formato utilizzato per spedire il terreno di gioco ai giocatori (vedi Sezione 3.1.1), con l'eccezione che le dimensioni del terreno di gioco non sono salvate su file (dato che possono essere calcolate). Un possibile file contenente il terreno di gioco è quindi analogo a quanto riportato in Figura 2, fatta salva l'omissione della prima riga.

Il terreno di gioco non è sufficiente per descrivere uno scenario di gioco, la configurazione dei robot che possono partecipare e la descrizione dei pacchetti in gioco sono parimenti necessarie. Il server deve quindi leggere tali informazioni da un file *nome.items* dove *nome* è l'argomento di `-board`. Il formato di tale file si può evincere dal seguente esempio:

```
robot 0 200 1000 @ (1,0) []
robot 1 200 1000 @ (2,0) []
package 0 10 (1,0) @ (1,9)
package 1 20 (1,0) @ (1,9)
package 2 30 (1,0) @ (1,9)
package 3 40 (1,0) @ (1,9)
# package 4 50 (1,0) @ (2,9)
```

Le righe che iniziano con '#' sono ignorate. Le righe che iniziano con `robot` descrivono un robot riportando nell'ordine: il suo identificativo, la sua forza, la sua quantità iniziale di €, la sua posizione iniziale.<sup>2</sup> Le righe che iniziano con `package` descrivono un

<sup>2</sup>la stringa finale `[]` è usata nel log (vedi Sezione 6.1) per rappresentare i pacchetti che il robot sta trasportando, non è però rilevante ai fini della configurazione iniziale di gioco e può essere ignorata

pacchetto riportando nell'ordine: il suo identificativo, il suo peso, la sua destinazione, la sua posizione iniziale.

*Esempio.* Una possibile riga di comando per invocare il server di gioco indicandogli di restare in ascolto per le connessioni dei giocatori alla porta 7919 dell'elaboratore corrente, caricando la mappa dal file `maps/pesissima` (e gli oggetti corrispondenti da `maps/pesissima.items`) è la seguente:

```
java it.unibo.cs.csrobots.GameServer \  
-board maps/pesissima \  
-address 127.0.0.1 -port 7919
```

## 4.4 Gestione della rete

La parte dell'implementazione di giocatori e server che riguarda l'accesso alla rete non vi compete. Per tanto vi viene fornito un insieme di classi pronte all'uso (il *netkit*), in particolare:

**Stub** uno *stub* rappresenta il collegamento (mediante una connessione TCP) ad un peer disponibile in rete. Uno *stub* offre due metodi per avere accesso ad un `BufferedReader`<sup>3</sup> e ad un `PrintWriter`<sup>4</sup> connessi al peer. Leggere dal primo equivale a ricevere testo dal peer, scrivere sul secondo ad inviarlo al peer;

**ServerConnection** questa classe è pensata per essere utilizzata dai giocatori per connettersi ad un server di gioco dati il suo indirizzo IP e la sua porta (gli stessi che questo ha ricevuto come parametri). Da una istanza di questa classe è possibile ottenere lo *stub* per dialogare con il server;

**PlayersGreeter** questa classe è pensata per essere utilizzata dal server per attendere le connessioni dei giocatori. Viene istanziata specificando indirizzo IP e porta sui quali rimanere in ascolto. Offre un metodo che quando invocato attende per la connessione di un giocatore e ritorna uno *stub* per comunicare con lui.

<sup>3</sup><http://java.sun.com/j2se/1.5.0/docs/api/java/io/BufferedReader.html>

<sup>4</sup><http://java.sun.com/j2se/1.5.0/docs/api/java/io/PrintWriter.html>

Tutte le classi del *netkit* sono locate nel namespace `it.unibo.cs.csrobots`.

Esempi di uso di queste classi e delle classi correlate sono riportati nei sorgenti `PlayerTest.java` e `ServerTest.java`.

## 5 Logistica

### 5.1 Gruppi

Il progetto deve essere svolto in gruppi.

Ogni gruppo deve essere composto da 3 persone. In casi eccezionali, richiedenti mia previa autorizzazione, si accettano deroghe a questa regola in difetto (gruppi composti da meno di 3 persone), ma non in eccesso (gruppi composti da più di 3 persone). Data la difficoltà del progetto è comunque fortemente consigliato che i gruppi siano composti da 3 persone.

Tutti i gruppi devono registrarsi *entro il 15 Maggio 2006* nella apposita pagina del Wiki (vedi Sezione 5.4).

### 5.2 Consegna

Il progetto dovrà essere consegnato entro e non oltre le 23:59:59 del 30 Giugno 2006 (fa fede la data degli elaboratori dei laboratori didattici). La consegna dovrà essere effettuata inviando una mail all'indirizzo `zack@cs.unibo.it` avente come oggetto [labprog0506] **consegna progetto**.

Alla mail dovrà essere allegato un archivio `.tar.gz` contenente un'unica directory avente al suo interno:

- un *file README* che descriva chi sono i membri del gruppo che ha svolto il progetto, peculiarità del progetto (e.g. la descrizione del funzionamento dei giocatori che si è scelto di implementare), le righe di comando necessarie ad invocare i giocatori ed il server di gioco implementati, la riga di comando (singolare) necessaria a compilare i sorgenti consegnati;
- il *diagramma delle classi* che compongono il progetto. Il diagramma deve essere in un qualche formato grafico *finale* (e.g. `pdf`, `png`, `ps` vanno bene; `corel draw`, `open office`, `powerpoint` non vanno bene);

- i sorgenti Java del progetto risultanti dalla fase di implementazione (una collezione di file `.java`); *N.B.* non devono essere consegnati i file `.class`;
- altri file necessari ad automatizzare la compilazione (e.g. `Makefile` o file di specifiche di `ANT`); *N.B.* non devono essere consegnati file specifici di eventuali ambienti di sviluppo utilizzati (e.g. Eclipse o Netbeans).

I progetti che non rispettano le modalità di consegna saranno da un minimo di penalizzati ad un massimo di considerati non svolti.

### 5.3 Valutazione

I progetti verranno valutati in base ai seguenti criteri (non in ordine di importanza):

1. *rispetto delle specifiche*: regole del gioco (Sezione 2), protocollo di gioco (Sezione 3), parametri di configurazione (Sezione 4);
2. *design*: verranno valutate le vostre scelte di design ad oggetti, ovvero come deciderete di rappresentare in una gerarchia di classi le entità che compongono il problema presentato da queste specifiche;
3. *bonus*: verrà premiato chi implementerà più di quanto richiesto, in particolare sviluppando più di un giocatore e/o giocatori dotati di euristiche più o meno intelligenti;
4. Se a stato avanzato di progetto un numero sufficiente di gruppi sarà a buon punto non escludo l'organizzazione di un *torneo* tra i giocatori da voi sviluppati nel quale verranno messi in palio alcuni punti per i gruppi autore dei vincitori della competizione.

Un aiuto a massimizzare il vostro successo per il criterio (1) vi viene offerto dalle *implementazioni di riferimento* che verranno rese disponibili sia del server di gioco che di un giocatore. Potrete in questo modo testare il vostro server prima di avere sviluppato un giocatore e confrontare il suo funzionamento

con quello del server di riferimento. *N.B.* le implementazioni di riferimento, come qualsiasi software, possono avere *bug* e non rispettare le specifiche. Segnalatemi quelli che ritenete siano bug, ne discuteremo assieme e li correggerò. In caso di incoerenza tra le implementazioni di riferimento e le specifiche, fanno fede le specifiche. I sorgenti delle implementazioni di riferimento non saranno resi disponibili prima della data ultima di consegna del progetto.

Per quanto riguarda il criterio (2) verrà valutato il vostro diagramma delle classi, il riutilizzo di codice (ovvero quanto siete stati bravi nel *non* riscrivere parti di codice che svolgono compiti molto simili), l'incapsulamento (i metodi che lavorano su certe informazioni sono nella stessa classe che racchiude quelle informazioni?), ed il data-hiding (avete reso pubblici solo le informazioni strettamente necessarie di una classe?).

In fase di discussione del progetto vi verranno inoltre posti dei problemi di *estensibilità* del vostro design. *Esempio* potrebbe esservi chiesto quali parti del vostro codice devono essere modificate se le regole del gioco vengono cambiate in modo da supportare un nuovo tipo di casella "tele-trasporto" che quando un robot vi entra lo sposta su un'altra cella dello stesso tipo scelta a caso tra quelle disponibili sul terreno di gioco.

Gli aspetti di estensibilità che dovete considerare nel design sono i seguenti:

- possibilità di aggiungere nuovi tipi di caselle al terreno di gioco;
- possibilità di aggiungere nuovi tipi di comandi impartibili ai robot;
- possibilità di limitare la quantità di informazione disponibile ai robot (e.g. non ricevere update, o ricevere update solo dei robot "vicini");
- possibilità di diversificare i tipi di robot che partecipano al gioco (e.g. robot più veloci di altri);
- possibilità di modificare il formato dei dati comunicati nel protocollo di gioco e/o il protocollo stesso.

Per quanto riguarda (3) e (4): via libera alla vostra fantasia!

## 5.4 Risorse

**Newsgroup** Lo strumento principale di comunicazione tra voi e con me sarà il newsgroup `unibo.cs.informatica.labprogrammazione` che io consulterò regolarmente. Vi invito a consultarlo spesso e ad esortare i vostri colleghi che non lo consultano a fare altrettanto. Non risponderò privatamente a domande inerenti al progetto, quando queste possano essere postate sul newsgroup.

**Wiki** Lo strumento principale per raccogliere informazioni più “persistenti” è un Wiki [3] basato sul Wiki engine MoinMoin [4]. Il Wiki è accessibile all’indirizzo: `http://mowgli.cs.unibo.it/~zacchiro/cgi-bin/moin.cgi/ProgettoLab10506`

**Laboratorio didattico** Gli elaboratori preposti allo sviluppo del progetto sono quelli dei laboratori didattici con sistema operativo GNU/Linux. Tutti i test vengono effettuati su tali macchine e con la loro versione degli strumenti di sviluppo. La risoluzione di problemi tecnici specifici di altre piattaforme (sistemi operativi, virtual machine, compilatori, ...) spetta interamente a voi.

**Directory condivisa** Alcuni file condivisi saranno resi disponibili sugli elaboratori del laboratorio didattico nella directory del corso (`/home/students/COURSES/labprogrammazione/`) in modo da potere essere direttamente accessibili ed eseguibili da tutti.

**Discussione comunitaria** In data 31 Maggio 2006 verrà effettuata una lezione di discussione comunitaria. La lezione non avrà un suo programma predefinito, sarà piuttosto un’occasione per confrontarsi di persona tra voi e con me. Di conseguenza presentatevi carichi di dubbi e di domande da porre che non siano state esaurientemente affrontate sul newsgroup.

## 6 Consigli conclusivi

Il progetto di quest’anno non è semplice, vi consiglio quindi di:

- *collaborare* tra gruppi, sia di persona che discutendo sul newsgroup. *N.B.* collaborare  $\neq$  copiare;
- *partecipare* alla lezione di discussione;
- *non reinventare* la ruota. La libreria standard di Java [5] è uno strumento molto potente in quanto contiene molte classi che possono esservi utili nell’implementazione del progetto. Ogni volta che vi si pone un problema implementativo quindi, prima di cominciare ad implementare una soluzione a testa bassa, controllate che non esistano già una o più classi che risolvano problemi simili nella libreria standard di Java.

### 6.1 Debugging del server

Per il debugging della comunicazione con il server di gioco vi consiglio l’uso dell’utility `netcat` [6]. Tale utility permette, tra le sue altre funzionalità, di collegarsi ad un applicativo in ascolto ad un indirizzo IP su di una data porta e di dialogare con lui usando la console. Utilizzando `netcat` potete collegarvi al server “fingendo” di essere il client, verificando così cosa accade durante la comunicazione. `netcat` è installato su tutti gli elaboratori del laboratorio ed è invocabile come comando `nc`.

*Esempio* Se il vostro server è in ascolto sulla macchina locale alla porta 7919 potete collegarvi a lui invocando `nc 127.0.0.1 7919`. Ciò che scrivete verrà inviato al server (in modalità *line-buffered*, ovvero ogni volta che premete invio) e ciò che lui vi risponde vi verrà mostrato sulla console. Di seguito trovate una sessione di esempio nella quale ciò che l’utente invia al server è mostrato in corsivo.

```
$ nc localhost 7919 # eseguo netcat
player
10 6
~~~~~
.....@
```

```

.....@
.....@
.....@
~~~~~
1 200 1000
#0,r 1 c 0;#1,r 2 c 0
[]
10 move n
#1,n;#0,k;#0,n
[]

```

Per il debugging dello stato interno del server, e quindi della corretta implementazione delle regole di gioco, vi consiglio invece di implementare un meccanismo di *logging* che, turno per turno, stampa cosa stia accadendo allo stato percepito dal server.

L'implementazione di riferimento accetta l'opzione `-logfile` per specificare un file sul quale viene riportato quanto accade, questo un estratto da un possibile file di log ottenuto utilizzando tale opzione:

```

----- turn 0 -----
robot 0 200 1000 @ (1,0) []
package 0 10 (1,0) @ (1,9)
package 1 20 (1,0) @ (1,9)
package 2 30 (1,0) @ (1,9)
package 3 40 (1,0) @ (1,9)
robot 1 200 1000 @ (2,0) []
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
0. . . . . . . . . @
1. . . . . . . . . @
. . . . . . . . . @
: ~ ~ ~ ~ ~ ~ ~ ~ ~
----- turn 1 -----
robot 0 200 999 @ (1,1) []
package 0 10 (1,0) @ (1,9)
package 1 20 (1,0) @ (1,9)
package 2 30 (1,0) @ (1,9)
package 3 40 (1,0) @ (1,9)
robot 1 200 999 @ (3,0) []
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
. 0. . . . . . . . . @
. . . . . . . . . @
1. . . . . . . . . @
: . . . . . . . . . @
~ ~ ~ ~ ~ ~ ~ ~ ~ ~

```

## 7 Changelog

**Versione 0.1** Prima release delle specifiche.

In bocca al lupo!

—*il lupo*

## Riferimenti bibliografici

- [1] *Java Technology*,  
<http://java.sun.com/>
- [2] *The Fifth ICFP Programming Contest*,  
<http://icfpcontest.cse.ogi.edu/>
- [3] *Wiki*, WIKIPEDIA The Free Encyclopedia,  
<http://en.wikipedia.org/wiki/Wiki>
- [4] *The MoinMoin Wiki Engine*,  
<http://moinmoin.wikiwikiweb.de/>
- [5] Java™ 2 Platform Standard Edition 5.0 API Specification,  
<http://java.sun.com/j2se/1.5.0/docs/api/index.html>
- [6] netcat,  
<http://www.vulnwatch.org/netcat/>