Discussione di Febbraio e Aprile 2007	**SpaceInvaders**
	Laurea Triennale in Comunicazione Digitale Laboratorio di Programmazione

1 Descrizione

Il progetto consiste nell'implementare in Java un'applicazione che simuli, con le dovute semplificazioni, l'esecuzione di una partita al famoso arcade space invaders.

2 Space Invaders

2.1 Il Gioco

Nella versione originale del gioco, il giocatore controlla gli spostamenti di un cannoncino laser che può muoversi esclusivamente da destra a sinistra e viceversa sul fondo dello schermo mentre dall'alto scendono schiere di alieni desiderosi di conquistare la Terra. Compito del giocatore è proprio quello di distruggere le armate nemiche senza farsi distruggere o lasciare conquistare la Terra.

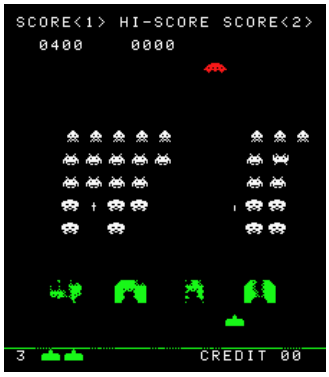

Figura 1: Coin-Op Screenshoot.

L'intera armata di alieni si muove sistematicamente da destra verso sinistra fino a raggiungere il bordo estremo quindi scende di una riga e si muove nella direzione opposta e così via fino a toccare terra. Durante i loro spostamenti, gli alieni sparano, in modo asincrono, dei proiettili diretti verso terra. Il cannoncino può sparare alla volta degli alieni e può proteggersi nascondendosi dietro delle casematte (3 per la precisione). Le casematte se colpite un numero adeguato di volte vengono distrutte, mentre basta un singolo colpo a distruggere un alieno o il cannoncino. Gli spari del cannoncino e quelli degli alieni di elidono quando si scontrano. Si hanno a disposizione tre cannoncini mentre, gli alieni eliminati non vengono rimpiazzati mentre se si distrugge l'intera armata un'altra la sostituisce passando allo stage successivo.

Parallelamente al gioco, una navicella aliena appare di tanto in tanto in cima allo schermo muovendosi da sinistra a destra arrivata al bordo opposto dello schermo sparisce. Ovviamente può essere colpita e rappresenta un bonus.

2.2 Semplificazioni e Varianti

Ovviamente non è possibile, nell'ambito del progetto, implementare interamente il gioco stesso ma dovrete realizzarne una simulazione che permetta il passaggio da una configurazione ad un'altra. Il tutto con le seguenti limitazioni/varianti rispetto al gioco originale.

- i cannoncini non sono controllati dal giocatore ma agiscono secondo uno schema prestabilito (vedere la descrizione della classe corrispondente);
- esistono diverse tipologie di alieni con schemi di gioco prestabilite e differenti (vedere la descrizione più avanti);
- basta un colpo per distruggere le casematte, queste occupano solo una casella del campo di gioco (esattamente come tutti gli altri elementi in gioco) e possono essere in numero arbitrario, possono essere ovunque tranne nella riga più in basso;
- non ci sono limiti sul numero di cannoncini a disposizione nell'arco della partita, distrutto un cannoncino questo riapparirà al centro della riga sul fondo dello schermo; non può esserci più di un cannoncino presente nel campo di gioco;
- l'armata aliena si muove solo da destra a sinistra e viceversa ma non scende o sale mai;
- distrutta un'armata questa non viene rimpiazzata;
- non si tiene conto in nessun modo del punteggio;

Lo scopo a questo punto diventa quello di far muovere le varie entità in gioco e di salvare (a richiesta) la configurazione corrente.

3 Classi da Realizzare

È obbligatorio realizzare in Java il programma descritto nelle sezioni precedenti utilizzando le seguenti classi:

BattleFieldElement

`BattleFieldElement` è una classe astratta usata per descrivere il comportamento, in astratto, dei vari personaggi in gioco.

Attributi

- **`int`** `x` e **`int`** `y`: variabili di istanza che memorizzano la posizione corrente dell'elemento sul campo di gioco.

Notare la peculiare presenza di alcuni attributi nella classe astratta, vale sempre la regola che lo stato degli oggetti deve essere nascosto ma in questo caso deve poter essere usato dalle classi concrete che estenderanno la classe astratta.

Metodi

- **`public void`** `move(`**`int`** `x, `**`int`** `y)`: sposta la posizione corrente del corrispondente elemento al punto specificato dai valori passatigli. Non si cura del fatto che potrebbe debordare o meno i limiti del campo di gioco.
- **`abstract public int`** `getXOffset()` e
  `abstract public int` `getYOffset()`: servono per stabilire, rispettivamente, di quante caselle l'elemento si può spostare verticalmente e orizzontalmente ed in quale direzione. La direzione sarà espressa dal segno:
  - \+ da sinistra a destra o dall'alto al basso (la casella (0,0) è in alto a sinistra della matrice);
  - \- da destra a sinistra o dal basso all'alto.
- **`public`** `String toString()`: metodo dall'ovvio significato ereditato da `Object` e ridefinito in modo da garantire la rappresentazione di ogni singolo elemento come spiegato descrivendo la configurazione del campo da gioco.

La classe `BattleFieldElement` dovrà essere estesa da diverse altri classi che rappresenteranno i vari elementi del gioco. Nel seguito descriveremo le singole classi senza entrare nel merito dell'implementazione dei metodi ereditati ma solo del comportamento delle sue istanze.

Alien

`Alien` classe astratta derivata da `BattleFieldElement`, rappresenta la classe progenitore di tutte le entità aliene presenti nel gioco.

Attributi

- **`int`** `direction`: indica in che direzione si muoverà l'alieno.

RedSpacecraft

`RedSpacecraft` classe concreta che estende `Alien`, rappresenta la navicella bonus, questa si muove sempre e solo da destra verso sinistra di una casella alla volta sulla riga più in alto dello schermo (sollevare l'eccezione `IllegalPositionException` se posizionata in una riga diversa). Arrivata in

fondo all'estrema sinistra sparisce dal campo di gioco.

OneStepAlien

`OneStepAlien` classe concreta che estende `Alien`, rappresenta gli alieni, tutti gli alieni di questo tipo si muovono come se fossero un'unica entità da destra a sinistra e viceversa (non scendono o salgono mai, partono muovendosi da destra verso sinistra). Quando un alieno raggiunge il bordo l'intera armata cambia direzione.

Shoot

La classe `Shoot` è una classe astratta derivata da `BattleFieldElement` che rappresenta la classe progenitrice di tutti i possibili spari, sia degli alieni (istanze della classe `AlienShoot`) che del cannoncino (istanze della classe `GunShoot`). Due spari che collidono si elidono come pure si elidono lo sparo e l'elemento (alieno, casamatta o cannoncino) che colpisce. Gli spari degli alieni vanno dall'alto verso il basso mentre quelli del cannoncino dal basso verso l'alto. Raggiunti gli estremi del campo di gioco escono dal gioco.

Empty

La classe `Empty` estende `BattleFieldElement` e rappresenta le caselle vuote che ovviamente non si spostano (o meglio non ha senso che si spostino).

Gun

Le istanze della classe `Gun` (che estende `BattleFieldElement`) rappresentano il cannoncino. Non ci può essere più di un cannoncino alla volta presente nella configurazione corrente (sollevare l'eccezione `IllegalElementException` quando si tenta di crearne più d'uno). Quando un cannoncino viene distrutto (colpito da uno sparo o scontrandosi con un alieno) viene rimpiazzato da un altro cannoncino la cui posizione iniziale sarà il centro della linea più in basso del campo di gioco. Ad ogni mossa, il cannoncino procede, una casella alla volta, nella direzione in cui si stava muovendo fino a raggiungere l'estremo del campo di gioco a questo punto inverte la direzione e continua la sua marcia (inizia muovendosi da sinistra a destra). I cannoncini possono essere solo sull'ultima riga altrimenti si deve sollevare l'eccezione `IllegalPositionException`.

Casemate

Le istanze della classe `Casemate` (che estende `BattleFieldElement`) rappresentano le casematte. Sono ferme nella loro posizione e ce ne possono essere un po' ovunque tranne in quella più in basso riservata al cannoncino (nell'ipotesi si deve sollevare l'eccezione `IllegalPositionException`).

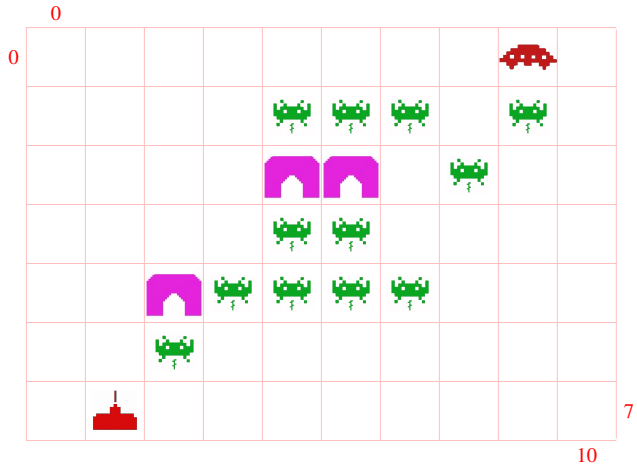

Figura 2: **Esempio di configurazione.**

BattleField

`BattleField` è la classe che descrive il campo di gioco (lo schermo del videogioco per intendersi) durante il gioco stesso. Sostanzialmente può essere schematizzato come una matrice di $n \times m$ elementi, con n e m non definiti a priori. Ogni elemento o sarà vuoto o conterrà un alieno, un cannoncino, uno sparo o una casamatta. In ogni istante un'istanza della classe `BattleField` fornirà uno snapshot della situazione (*configurazione*) corrente del campo di gioco.
La configurazione è codificata in una stringa come segue:

- una coppia di interi (separati da una barretta verticale — |), compresi tra 1 e 100, rappresentano le dimensioni del campo di gioco (prima il numero di righe, poi il numero di colonne);
- ogni riga è rappresentata da una sequenza di coppie *nc* dove $n \in (1, \cdots, 9]$ rappresenta il numero di caselle contigue contenenti lo stesso tipo di elementi mentre $c \in \{$`'R'`, `'A'`, `'G'`, `'C'`, `'s'`, `'S'`, `'␣'`$\}$ rappresenta l'elemento in questione; notare che avere $n \in (1, \cdots, 9]$ non limita a 9 la lunghezza delle righe in quanto si possono avere due o più coppie contigue dello stesso elemento, ad es. `9␣7␣`;
- i caratteri usati nella configurazione hanno il seguente significato:
  - `'R'` — navicella rossa (il bonus nel gioco originale);
  - `'A'` — alieno;
  - `'G'` — cannoncino;
  - `'C'` — casamatta;
  - `'s'` — sparo del cannoncino;

  - `'S'` — sparo degli alieni;
  - `'␣'` — casella vuota.

- $ rappresenta la fine di una riga;

La Figura **??** mostra una possibile configurazione per un campo di gioco di 7×10 caselle. In particolare sono presenti 18 elementi: 1 navicella, 1 cannoncino, 3 casematte e 13 alieni. La corrispondente stringa sarà:

`7|10|8␣1R1␣$4␣3A1␣1A1␣$4␣2C1␣1A2␣$4␣2A4␣$2␣1C4A3␣$2␣1A7␣$1␣1G8␣$`

Notare che nell'esempio non ci sono spari in gioco e gli alieni sono tutti trattati allo stesso modo (ad eccezione della navicella rossa).

Attributi

- `BattleFieldElement[][] battlefield`: variabile d'istanza che memorizza la configurazione corrente del campo di gioco. Ogni elemento della matrice conterrà un elemento diverso da `NULL`.
- **`int`** `rows` e **`int`** `columns`: numero totale delle righe e delle colonne che compongono il campo di gioco.
- `String filename`: variabile di istanza contenente il nome del file su cui si salvano le configurazioni e che viene usato per il ripristino della configurazione. Implementare anche i corrispondenti metodi `publici String getFilename()` e **`void`** `setFilename(String)` dall'ovvio significato.

Metodi e Costruttori

- **`public`** `BattleField(String filename)`: costruttore che configura il campo di gioco, `filename` è il nome del file su disco contenente una serie di configurazioni, una per riga, l'ultima configurazione letta diventerà la configurazione corrente.
- **`void`** `setBattleFieldElement(`**`int`**`,` **`int`**`, BattleFieldElement)` e `BattleFieldElement getBattleFieldElement(`**`int`**`,` **`int`**`)`: metodi accessori per la gestione del singolo elemento del campo di gioco;
- **`public`** `String toString()`: metodo che crea e ritorna una stringa rappresentante la configurazione corrente della campo di gioco, sfruttando le convenzioni descritte precedentemente;
- **`public void`** `setBattleField(String)`: metodo che permette di inizializzare il campo di gioco alla configurazione passata come parametro, la configurazione e codificata nella stringa secondo le stesse convenzioni descritte precedentemente;
- **`public`** `String getBattleField()`: metodo che ritorna la configurazione corrente codificata secondo le convenzioni descritte precedentemente;

- **public void** `write()`: metodo che appende la configurazione corrente del campo di gioco al file corrispondente;
- **public void** `reload()`: metodo che rilegge dal file indicato dall'attributo `filename` la configurazione del campo di gioco;
- **public** `Object clone()`: metodo che crea una copia dell'oggetto e la ritorna;
- **public void** `backup(String)`: metodo che effettua una copia di backup della configurazione del campo di gioco, salvandola in un file il cui nome è specificato nella stringa passata come argomento, il file verrà creato ex-novo;
- **void** `move()`: metodo che fa avanzare la configurazione di un passo; partendo dall'angolo in alto a sinistra e procedendo da sinistra a destra per righe successive; lo spostamento deve essere fatto *in situ* e non su una copia della matrice sfruttando i vari metodi forniti dai vari `BattleFieldElement`, sarà il metodo `move()` a gestire i casi particolari (come le collisioni);

Attenzione che i vari metodi potrebbero sollevare delle eccezioni queste non sono state specificate nel testo ma dovranno essere gestite *cum grano salis*.

Eccezioni.

Le classi `IllegalElementException`, `IllegalPositionException` e `IllegalMovementException`, da definire in modo opportuno, rappresentano le eccezioni da lanciare (con le modalità indicate) quando si verificano le condizioni di errore descritte nei punti precedenti.

Tutte le altre eccezioni previste dall'uso di metodi Java devono filtrare ed essere gestite nel metodo `main()` anche se non espressamente indicato dalla segnatura dei metodi introdotti in questo documento.

A parte quanto espressamente richiesto, è lasciata piena libertà sull'implementazione delle singole classi e sull'eventuale introduzione di altre classi, a patto di seguire le regole del paradigma ad oggetti ed i principi di buona programmazione.

Non è richiesto e non verrà valutato l'utilizzo di particolari modalità grafiche di visualizzazione: è sufficiente una qualunque modalità di visualizzazione basata sull'uso dei caratteri.

È invece **espressamente richiesto** di non utilizzare package non standard di Java (si possono quindi utilizzare `java.util`, `java.io` e così via), con l'unica eccezione del package `prog.io` incluso nel libro di testo per gestire l'input da tastiera e l'output a video e di rispettare l'interfaccia fornita.

4 Esempio di Programma di Test ed Esecuzione

La classe `TestSpaceInvaders` riportata di seguito è un esempio di possibile `main()` contro il quale verrà provata il vostro programma. **Non verranno presi in considerazione progetti le cui classi non permetteranno la compilazione corretta di questa classe e la cui esecuzione non dia il risultato atteso e riportato di seguito.**

```
public class TestSpaceInvaders {
  public static void main(String[] args) throws Exception {
    BattleField bf = new BattleField("es-in.txt");
    System.out.println("at the beginning :- "+bf);
    bf.move();
    System.out.println("1st step :- "+ bf);
    bf.move();
    System.out.println("2nd step :- "+bf);
    bf.move();
    System.out.println("3rd step :- "+bf);
    bf.setBattleFieldElement(5,3,new GunShoot(5,3));
    bf.setBattleFieldElement(2,1,new AlienShoot(2,1));
    System.out.println("new entries :- "+bf);
    bf.move();
    System.out.println("4th step :- "+bf);
    bf.move();
    System.out.println("5th step :- "+bf);
    bf.move();
    bf.setFilename("es-out.txt");
    bf.write();
  }
}
```

```
[16:04]cazzola@ulik:~/fap/febbraio 2007/soluzione>cat es-in.txt
5|5|1␣1B3␣$5␣$2␣1C2␣$5␣$2␣1A2␣$
5|5|1␣1R3␣$5␣$1C1␣1C2␣$5␣$2␣1A2␣$
7|10|8␣1R1␣$4␣3A1␣1A1␣$4␣2C1␣1A2␣$4␣2A4␣$2␣1C4A3␣$2␣1A7␣$1␣1G8␣$
7|10|1␣1R6␣1R1␣$4␣3A1␣1A1␣$4␣2C1␣1A2␣$4␣2A4␣$2␣1C4A3␣$2␣1A7␣$1␣1G8␣$
[16:04]cazzola@ulik:~/fap/febbraio 2007/soluzione>java TestSpaceInvaders
at the beginning :- 7|10|1␣1R6␣1R1␣$4␣3A1␣1A1␣$4␣2C1␣1A2␣$4␣2A4␣$2␣1C4A3␣$2␣1A7␣$1␣1G8␣$
1st step :- 7|10|1R6␣1R2␣$3␣3A1␣1A2␣$4␣2C1A3␣$3␣2A5␣$3␣3A4␣$1␣1A8␣$2␣1G7␣$
2nd step :- 7|10|6␣1R3␣$2␣3A1␣1A3␣$4␣1C5␣$2␣2A6␣$2␣3A5␣$1A9␣$3␣1G6␣$
3rd step :- 7|10|5␣1R4␣$3␣3A1␣1A2␣$4␣1C5␣$3␣2A5␣$3␣3A4␣$1␣1A8␣$4␣1G5␣$
new entries :- 7|10|5␣1R4␣$3␣3A1␣1A2␣$1␣1S2␣1C5␣$3␣2A5␣$3␣3A4␣$1␣1A1␣1s6␣$4␣1G5␣$
4th step :- 7|10|4␣1R5␣$4␣3A1␣1A1␣$4␣1C5␣$1␣1S2␣2A4␣$3␣1s3A3␣$2␣1A7␣$5␣1G4␣$
5th stepv:- 7|10|3␣1R6␣$5␣3A1␣1A$4␣1C5␣$3␣1s1␣2A3␣$1␣1S3␣3A2␣$3␣1A6␣$6␣1G3␣$
[16:04]cazzola@ulik:~/fap/febbraio 2007/soluzione>cat es-out.txt
7|10|2␣1R7␣$4␣3A1␣1A1␣$3␣1s1C5␣$4␣2A4␣$4␣3A3␣$1␣1S1A7␣$7␣1G2␣$
[16:04]cazzola@ulik:~/fap/febbraio 2007/soluzione>
```

Dove ␣ evidenzia la presenza di uno spazio nelle configurazioni.

5 Modalità di Consegna

Il progetto deve essere svolto a gruppi di al massimo tre persone che intendono sostenere l'intero esame di Fondamenti di Architettura e Programmazione - Laborato-

rio di Programmazione nell'appello di Febbraio 2007, e deve essere consegnato entro mezzanotte di lunedì 12 febbraio 2007 tramite il sito di sottoposizione:

`http://homes.dsi.unimi.it/~malchiod/LP/sottoposizione`

Per sottomettere un progetto dovrete registrarvi e creare un gruppo a cui affiliare voi ed i vostri compagni di gruppo. La sottoposizione è fatta a nome del gruppo e vale per ognuno dei componenti del gruppo stesso. Nota, sottoposizioni successive cancellano le sottoposizioni già fatte.

Dovranno essere consegnati tutti i sorgenti Java che permettano al programma di essere eseguito correttamente, compressi in un archivio di tipo ZIP che estragga i file nella directory in cui si trova l'archivio stesso. Nell'archivio dovrà anche essere accluso un breve documento in formato txt o rtf in cui:

- verrà descritto il modo in cui interfacciarsi con il programma;
- saranno illustrate le principali scelte implementative e le strategie utilizzate per svolgere il progetto

Le sottoposizioni che non seguono queste specifiche ed i programmi che non compilano correttamente non verranno valutati.

È inoltre richiesto di consegnare una copia cartacea della stampa del codice sorgente prodotto in portineria del DSI/DICo indicando chiaramente nome, cognome e numero di matricola di tutti i componenti del gruppo, nonché il turno e il docente di riferimento. Nella copia cartacea indicare anche se il gruppo **il gruppo** intende discutere il progetto a febbraio o ad aprile (sconsigliato).

6 Valutazione

Durante la prova orale con i singoli studenti saranno discusse le modalità implementative adottate e la padronanza di alcuni dei concetti necessari per preparare il progetto e/o spiegati a lezione. La valutazione del progetto sarà fatta in base alla:

- conformità dell'implementazione scelta per risolvere il problema con il paradigma di programmazione a oggetti;
- conformità del codice presentato alle regole di buona programmazione;
- adeguatezza della documentazione consegnata;
- assenza di errori nel programma;

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Via Comelico 39/41 20135 Milano
Stanza S233 — Tel. +39.010.353.6637
e-mail: `cazzola@dico.unimi.it`

Dario Malchiodi
Dipartimento di Scienze dell'Informazione
Via Comelico 39/41 20135 Milano
Stanza S238 — Tel. +39.02.503.16338
e-mail: `malchiodi@dsi.unimi.it`