Discussione di Febbraio e Aprile 2009	**Gioco dell'Oca**
	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/estensioni, l'esecuzione di una partita all'omonimo gioco da tavola.

2 Gioco dell'Oca

2.1 Il Gioco (Versione Originale)

Figura 1: Tabellone.

Si gioca su un tabellone, simile a quello di Fig. 1, sul quale è disegnato un percorso a spirale composto da 63 caselle (talvolta questo numero sale fino a 90), contrassegnate con numeri o altri simboli. I giocatori iniziano con un segnalino nella casella di partenza e, a turno, procedono lungo il percorso di un numero di caselle ottenuto attraverso il lancio di una coppia di dadi. Lo scopo del gioco è raggiungere la casella centrale della spirale.

Alcune caselle di arrivo hanno un effetto speciale. Nella versione tradizionale, le caselle che rappresentano oche (da cui il nome del gioco) consentono di spostarsi subito in avanti di un numero di caselle pari a quelle coperte dal movimento appena effettuato. Queste caselle sono collocate ogni nove caselle a partire dalle caselle 5 e 9 (una conseguenza di questa disposizione è che un lancio iniziale di 9 porta immediatamente il giocatore alla casella 63 e quindi alla vittoria).

Altre caselle con effetti speciali sono:

- casella 6 (“il ponte") si paga la posta e si ripete il movimento come nelle caselle con le oche;
- casella 19 (“casa" o "locanda") si paga la posta e si rimane fermi tre turni;
- caselle 31 (“pozzo") e 52 ("prigione") si rimane fermi fino a quando non arriva nella casella un altro segnalino, che viene a sua volta "imprigionato";
- casella 42 (“labirinto") si paga la posta e si torna alla 39;
- casella 58 (“scheletro") si paga la posta e si torna alla 1.

La casella d'arrivo deve essere raggiunta con un lancio di dadi esatto; altrimenti, giunti in fondo, si retrocede dei punti in eccesso.

2.2 Semplificazioni e Varianti

Ovviamente la realizzazione del gioco dell'oca originale non è possibile, nell'ambito del progetto, in particolare le interazioni tra giocatori. Quello che si dovrà implementare è la simulazione di una variante del gioco tramite l'evoluzione della configurazione del tabellone. Il tutto con le seguenti limitazioni/varianti rispetto al gioco originale:

- il numero delle caselle che compongono il campo non è prefissato e non sono necessariamente disposte a spirale, saranno ovviamente contigue a formare un percorso senza bivi;
- le varie caselle sono caratterizzate da un valore (positivo o negativo) e da uno (eventuale) spostamento; il valore della casella va a sommarsi al punteggio del giocatore che ci si ferma;
- le caselle “oche” sono disposte ovunque sul campo di gioco e hanno la proprietà di far ripetere al giocatore lo spostamento e di raddoppiare il valore della casella di destinazione;
- le caselle “prigione” sono disposte ovunque sul campo di gioco e bruciano 3 turni del malcapitato giocatore che ci si fermerà;
- la casella “indietreggia” farà indietreggiare il giocatore di 3 caselle;
- raggiunto un capo del percorso si cambierà direzione (il giocatore si gira di 180°) di marcia e proseguirà con l'avanzo del lancio del dado.

Il gioco terminerà quando uno dei 4 (sono sempre 4) giocatori raggiungerà la casella traguardo (che può essere ovunque nel percorso) e avrà vinto il giocatore che ha accumulato il punteggio più alto. Raggiunta la fine del percorso si *rimbalza* indietro e si prosegue nella direzione opposta. I giocatori inizieranno la partita dalla casella “inizio” (che può essere ovunque nel percorso) e inizieranno a muoversi verso destra se la casella a destra non è vuota, oppure verso l'alto, oppure verso sinistra, oppure verso il basso.

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

3 Struttura del Campo di Gioco

Il campo di gioco avrà struttura bidimensionale $n \times m$ e verrà descritto per righe dal basso verso l'alto e sinistra verso destra. Ogni riga è descritta da una sequenza di

coppie «“n^{o} caselle contigue dello stesso tipo” “tipo caselle”» separate dal simbolo `"."` e terminata dal simbolo `"$"`. I tipi di caselle ammesse sono: `"G"` casella oca, `"J"` casella prigione, `"T"` casella finale, `"S"` casella d'inizio, `"B"` casella indietreggia, `"E"` casella vuota, oppure un intero, dove l'intero rappresenta il valore (da moltiplicare per mille) della casella. Le dimensioni del campo da gioco sono inserite ad inizio configurazione separate da uno spazio tra loro e da `|` dal resto della configurazione. Tenere presente che in una configurazione corretta il percorso sarà leggibile da sinistra a destra e dal basso verso l'alto; diverse interpretazioni/configurazioni sono da considerarsi errate.

Il seguente è un esempio di input per il campo di gioco (per semplicità l'input va a capo, nel file non sarà così).

```
11␣14|3␣E.1␣1.10␣E$3␣E.1␣J.10␣E$3␣E.1␣J.2␣2.1␣-1.1␣G.1␣-3.1␣-2.1␣2.1␣4.2␣E$
11␣E.1␣J.2␣E$6␣E.1␣J.2␣4.1␣3.1␣E.1␣G.2␣E$5␣E.2␣G.2␣E.1␣-1.1␣-2.1␣T.2␣E$
5␣E.1␣3.8␣E$1␣E.1␣7.1␣-1.1␣1.1␣B.1␣3.8␣E$1␣E.1␣-14.12␣E$1␣E.1␣G.1␣J.11␣E$
2␣E.1␣B.1␣3.1␣-1.1␣S.1␣G.7␣E$
```

Nota. Non ci sono garanzie che l'input contenga il giusto quantitativo di caselle, che ci siano una casella finale ed una di inizio e che le caselle siano contigue, sarà cura del vostro programma verificarlo.

4 Classi da Realizzare

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

Piece

`Piece` è una classe usata per descrivere la casella generica che compone il campo di gioco.

Attributi

- **`int`** `value`: valore assegnato alla casella;
- **`int`** `offset`: eventuale spostamento che subisce il giocatore che atterra su questa casella.

Metodi

- **`int`** `getValue()`: restituisce il valore associato alla casella;
- **`int`** `getOffset()`: restituisce di quanto si deve muovere ed in che direzione (se negativo nella direzione opposta a quella in cui stava andando) il giocatore;
- `String toString()`: 'nuff said!!! realizzatela in modo che sia utile.

Dalla classe `Piece` si deriveranno le classi `EmptyPiece`, `GoosePiece`, `JailPiece`, `BackPiece`, `StartPiece` e `TargetPiece` che rappresentano, rispettivamente, le caselle speciali spazio vuoto, “oche”, “prigione”, “indietreggia” “inizio” e “traguardo”. Non sono ulteriormente descritte.

Nota. Potete ovviamente derivare altri tipi di caselle che ritenete opportuni; la scelta ovviamente verrà premiata o punita in base alla sua validità.

Board

Descrive il campo di gioco in termini della sua configurazione corrente.

Attributi

- `board`: variabile di istanza che memorizza elementi nel campo di gioco (cioè le istanze di `Piece` o suoi derivati).
- `String filename`: variabile di istanza contenente il nome del file da cui si legge la configurazione. Implementare anche i corrispondenti setter and getter: `String getFilename()` e **`void`** `setFilename(String)`.

Metodi e Costruttori

- **`public`** `Board(String filename)`:
  costruttore che legge la configurazione del campo di gioco, `filename` è il nome del file su disco contenente una serie di campi da gioco, uno per riga, l'ultima riga letta diventerà la configurazione corrente del campo di gioco.
- **`public`** `String toString()`:
  metodo che crea e restituisce una stringa rappresentante la configurazione corrente del campo di gioco, sfruttando le convenzioni descritte precedentemente;
- **`public void`** `setBoard(String)`:
  metodo che permette di inizializzare il campo di gioco alla configurazione passata come parametro, la configurazione è codificata nella stringa secondo le stesse convenzioni descritte precedentemente;
- **`public`** `String getBoard()`:
  metodo che restituisce 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 restituisce;
- **`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 rimpiazzandone uno eventualmente già esistente;
- **`public`** `Piece getPiece(Player p)`:
  restituisce il pezzo del campo di gioco relativo alla posizione corrente del giocatore specificato come parametro. Solleverà un'eccezione quando la posizione richiesta non esiste;
- **`public void`** `setPiece(Player p, Piece n)`:

sostituisce il pezzo del campo di gioco alla posizione corrente del giocatore con quello passato come parametro;

Player

Classe rappresentante il singolo giocatore.

Attributi

- **int** `player`: variabile di istanza che identifica il giocatore e può assumere valori da 1 a 4.
- `?? position`: variabile di istanza che conserva la posizione corrente del giocatore all'interno del campo di gioco; il tipo non è specificato perché la scelta è lasciata a voi.
- **int**`[] tosses`: variabile di istanza che contiene i lanci di dati; ogni entry dell'array rappresenta il risultato che si ottiene dal dado lanciandolo.
- **int** `bonus`: punteggio accumulato dal giocatore, inizialmente posto a zero.
- `String filename`: variabile di istanza del file da cui si leggono i lanci a disposizione del giocatore; il nome del file è strettamente legato al nome del file del campo da gioco, ad esempio se il campo da gioco è memorizzato nel file `"file.txt"`, i file dei 4 giocatori saranno rispettivamente `"file1.txt"`, `"file2.txt"`, `"file3.txt"` e `"file4.txt"`.

Implementare anche i corrispondenti metodi getter e setter.

Il file di ogni giocatore conterrà i lanci dei dadi codificati su una riga sola come segue:

- `m` $\mathtt{n}_1\mathtt{n}_2\cdots\mathtt{n}_m$ con $\mathtt{m}\in\mathbb{N}$ e $\mathtt{n}_i\in\{1,2,\cdots,6\}\ \forall i\in\{1,2,\cdots,m\}$ (non ci sono spazi tra gli $\mathtt{n}_i$, vi è invece uno spazio tra `m` e $\mathtt{n}_1$).

Metodi e Costruttori

- **public** `Player(String filename)`:
  il costruttore riceverà **lo stesso** nome di file che riceve il costruttore di `Board` e calcolerà automaticamente il numero del giocatore e quindi il nome del file da usare; questo costruttore non può essere invocato più di quattro volte;
- **public int** `nextDiceToss()`:
  questo metodo restituisce il risultato del lancio del dado per il turno corrente; ovviamente il giocatore tiene traccia internamente del turno.
- **public void** `move(`**int** `toss)`:
  questo metodo sposta la pedina del giocatore di `toss` caselle;
- **public** `String toString()`:
  questo metodo stamperà **esattamente** la stringa:

  ```
  The Player «id» is on «position»; it has «bonus».
  ```

  dove ovviamente `«id»`, `«position»` e `«bonus»` devono essere rimpiazzati da identità, posizione e punteggio del giocatore.

GameOfTheGoose

La classe `GameOfTheGoose` permette di gestire una partita al gioco dell'oca.

Attributi.
- `Board board`: rappresenta il campo di gioco;
- `Player[] players`: i giocatori in gara;
- `String filename`: il nome del file contenente la configurazione del campo di gioco;
- **`int`** `moves`: contiene il numero di lanci di dado che compongono la partita.

Costruttore e Metodi.
- **`public`** `GameOfTheGoose(String fn)`:
  il costruttore riceve esclusivamente il nome del file cui può attingere tutte le informazioni;
- **`public void`** `makeBoard()`:
  costruisce il campo di gioco;
- **`public void`** `makePlayers()`:
  costruisce i giocatori, leggendo le relative informazioni dai file corrispondenti;
- **`public void`** `move(int player)`:
  muove il giocatore indicato da `player`, calcolando il bonus ed eventuali spostamenti bonus o di penalità;

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

Eccezioni.

Tutte le 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) e di rispettare l'interfaccia fornita. **Nota.** la libreria `prog` non fa parte delle librerie standard pertanto non ne è concesso l'utilizzo.

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 - Laboratorio di Programmazione negli appelli di Febbraio o Aprile 2009, e deve essere consegnato **entro la mezzanotte tra venerdì 20 e sabato 21 febbraio 2009** 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, **entro lunedì 23 febbraio**, una copia cartacea della stampa del codice sorgente prodotto e della relazione 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 ogni singolo componente del gruppo intende discutere il progetto a febbraio o ad aprile.

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 P121 | Tel. +39.02.503.16300
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