

SAPIENZA Università di Roma

A.A. 2008-2009

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Esercitazioni di Progettazione del Software
(Canale A-L)

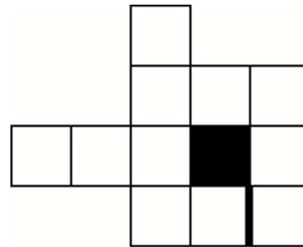
Realizzazione di Applicazioni Basate su Eventi

Fabio Patrizi

Requisiti

Nel videogioco *Gold Seekers*, un giocatore (umano) sfida il computer in una corsa all'oro.

I giocatori si muovono su diverse caselle quadrate. Di ciascuna casella, interessa conoscere tutte quelle ad essa adiacenti, lungo le direzioni nord, sud, est, ovest. Ciascuna casella può essere adiacente ad al più un'altra per direzione, e deve sempre averne almeno una adiacente (ovvero, tutte le caselle devono essere *connesse*). Una configurazione di esempio è mostrata in figura. La casella nera indica assenza di caselle, mentre il bordo spesso indica che le caselle interessate non sono adiacenti.



La presenza di un pozzo ed un sacchetto d'oro, o di due o più sacchetti d'oro nella stessa casella non è ammessa.

Gli elementi di gioco possono essere di tre tipi: sacchetti d'oro, pozzi d'acqua o cercatori d'oro. Ciascun sacchetto è caratterizzato dal proprio peso, ovvero una misura della quantità d'oro che contiene. In ogni istante, ciascun cercatore può aver raccolto nessuno o più sacchetti.

Per raccogliere un sacchetto, il cercatore deve posizionarsi sulla casella che lo ospita. Una volta raccolto, il sacchetto non è più contenuto nella cella.

Requisiti (cont.)

Ciascun cercatore è caratterizzato dal proprio nome (una stringa).

Un cercatore può essere negli stati: *disidratato*, *dissetato* o *non in gioco*. Inizialmente, è nello stato *non in gioco*. Quando il gioco inizia, ha una riserva d'acqua di 5 litri e passa allo stato *dissetato*. Quando la sua riserva d'acqua raggiunge il valore 0, il cercatore entra nello stato *disidratato*. Quando un cercatore è disidratato, va in cerca d'acqua, altrimenti va in cerca d'oro.

In ogni momento, il cercatore può ripristinare la propria riserva d'acqua occupando una casella con pozzo. In tal caso, la riserva raggiunge il valore 10 ed il cercatore torna nello stato *dissetato*. Ad ogni turno, il cercatore consuma 0,5 litri d'acqua. Quando è disidratato, il cercatore non può raccogliere i sacchetti d'oro, sebbene possa muoversi liberamente sulla mappa, occupando anche le celle contenenti i sacchetti d'oro (che non sono, però, raccolti).

Esistono due tipi di cercatori: autonomi e controllati.

I cercatori controllati vengono comandati dall'utente. Prima di effettuare ogni mossa, il cercatore rimane in attesa che il giocatore indichi la direzione (mediante opportuna interfaccia) verso cui muoversi.

I giocatori muovono uno alla volta, seguendo una politica basata su turni. Il raggiungimento di una casella contenente un pozzo o un sacchetto d'oro e l'azione di raccolta o ripristino dell'acqua sono da considerarsi effettuate in un unico turno.

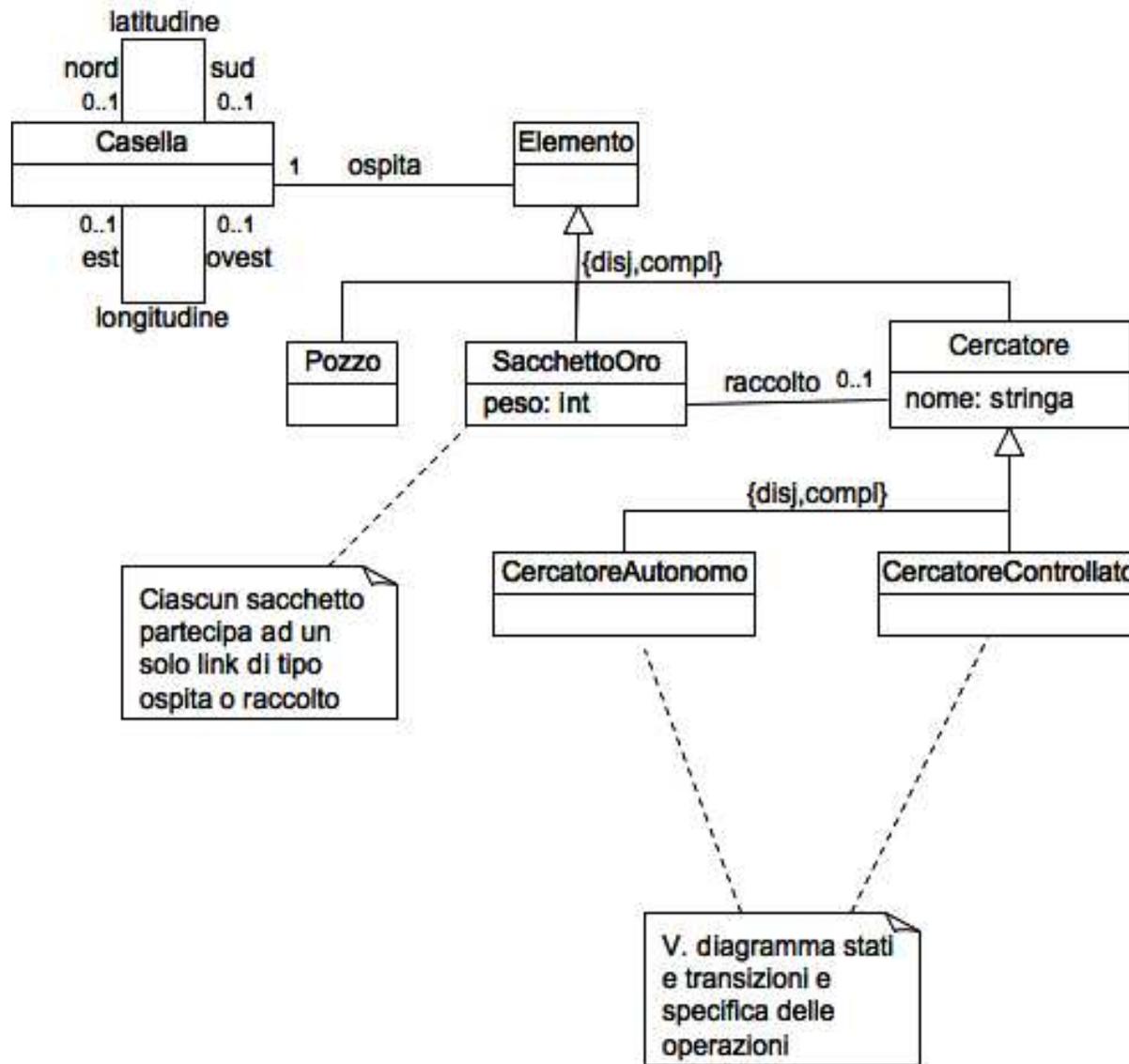
Requisiti (cont.)

Quando sono nello stato *dissetato*, i cercatori autonomi si muovono di un passo verso una delle caselle più vicine contenenti oro. Raggiunta una di esse, raccolgono l'oro. Quando sono nello stato *disidratato*, si muovono di un passo verso la casella più vicina contenente un pozzo e, una volta raggiunta, ripristinano la riserva d'acqua.

Quando non ci sono più sacchetti d'oro presenti, i cercatori tornano nello stato *non in gioco*. Si deve prevedere un'operazione che, dato un insieme di giocatori, restituisca quelli che hanno raccolto la maggior quantità d'oro.

Fase di Analisi

Diagramma UML delle Classi



Vincoli Aggiuntivi (Invarianti)

Ogni casella deve sempre averne almeno una adiacente

```
context Casella
  inv ( self.latitudine.nord->size() + self.latitudine.est->size() +
        self.latitudine.sud->size() + self.latitudine.ovest->size()= 1)
```

Nessuna casella è adiacente a se stessa:

```
context Casella
  inv ( self.latitudine.nord <> self and self.latitudine.sud <> self and
        self.latitudine.est <> self and self.latitudine.ovest <> self )
```

(Inoltre, potremmo garantire la consistenza del campo di gioco imponendo ulteriori vincoli come, ad esempio, che le associazioni *longitudine* e *latitudine* siano delle *relazioni d'ordine stretto*).

Non possono coesistere un pozzo ed un sacchetto d'oro o due sacchetti d'oro nella stessa casella

```
context Casella
  inv not (self.ospita->exists
           (os | os.Elemento.oclIsKindOf(SacchettoOro) and
            (self.ospita->exists(os | os.Elemento.oclIsKindOf(Pozzo)) or
             self.ospita->exists(os1 | os1<>os
                                os1.Elemento.oclIsKindOf(SacchettoOro)))
```

)))

Ciascun sacchetto è coinvolto in un solo link *ospita* o *raccolto*

```
context SacchettoOro
```

```
  inv (self.ospita->size()+self.raccolto->size())=1)
```

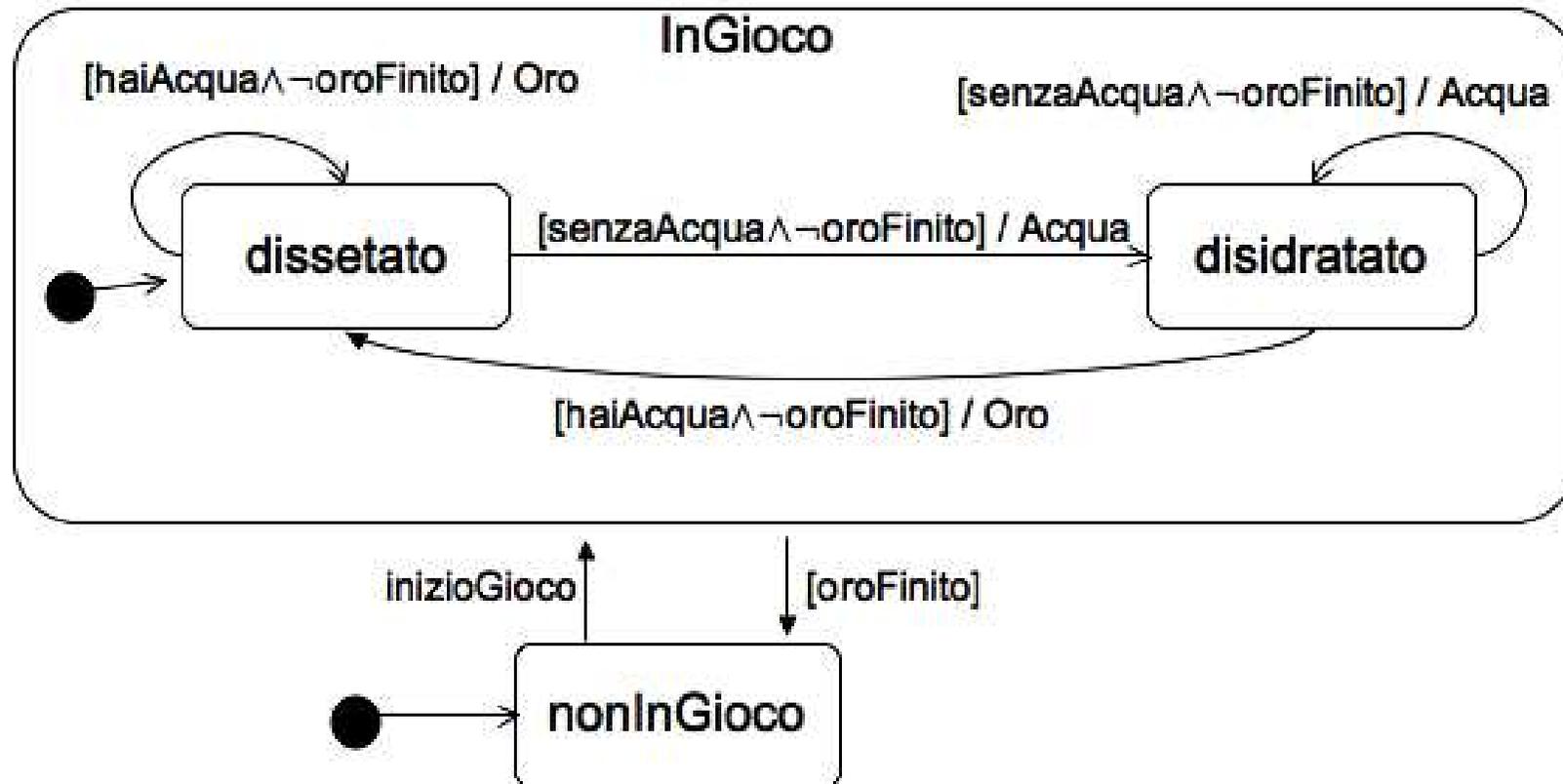
Specifica Operazioni Utente

InizioSpecificaOperazioniUtente Controlli

```
context Utente::massimoOro(c:Set(Cercatore)):Set(Cercatore)
  pre: --
  post:
    result = c.select(cerc | not c->exists(cerc1|
      cerc.raccolto.SacchettoOro.peso->sum() < cerc1.raccolto.SacchettoOro.peso->sum()))
```

FineSpecifica

Diagramma UML degli Stati e delle Transizioni



Specifica Stati Classe Cercatore

InizioSpecificaStatiClasse Cercatore

Stato: {nonInGioco, dissetato, disidratato}

Variabili di stato ausiliarie:

acqua: reale

Stato Iniziale:

stato = nonInGioco

acqua = --

FineSpecifica

Le transizioni sono specificate nelle classi CercatoreAutonomo e CercatoreControllato, poiché da esse dipende l'effettiva implementazione dell'azione.

Specifica Transizioni Classe CercatoreAutonomo

InizioSpecificaTransizioniClasse CercatoreAutonomo

Transizione: inGioco->nonInGioco

Evento:--

Condizione:SacchettoOro.allInstances().ospita->notEmpty()

Azione:

pre:--

post:--

Transizione: nonInGioco->inGioco

Evento:inizioGioco

Condizione:--

Azione:

pre:evento.dest=self

post:acqua=5

Transizione: dissetato->dissetato

Evento:--

Condizione:acqua>0

Azione:

pre:--

post:

```
/*(Informale) Il cercatore si sposta di una casella verso l'oro piu' vicino:  
viene eliminato il link corrente ospita tra il cercatore e la casella su cui  
e' posizionato e ne viene creato uno tra la nuova casella occupata  
ed il cercatore. */
```

```

and
acqua = acqua@pre-0,5
and
let linkOro=self.ospita.Casella.ospita@pre.allInstances()->select(
                                Elemento.oclIsKindOf(SacchettoOro))
in(
  linkOro->notEmpty() implies
  /*il sacchetto viene raccolto*/
  /*ospita non contiene piu' il link al sacchetto d'oro e per il resto rimane immutata*/
  ospita->allInstances()=ospita@pre.allInstances()->excluding(linkOro) and
  /*raccolto contiene un (nuovo) link tra il cercatore ed il sacchetto d'oro*/
  raccolto.allInstances()->exists(ra | ra.Cercatore = self and ra.SacchettoOro=linkOro.SacchettoOro) and
  /*il resto di raccolto rimane immutato*/
  let
    nuovoLinkRaccolto=raccolto.allInstances()->select(Cercatore=self and
                                                    SacchettoOro=linkOro.SacchettoOro)
  in
    raccolto.allInstances()->excluding(nuovoLinkRaccolto)=raccolto@pre.allInstances()
)
and
/*Se si sta passando sopra una casella con pozzo, la riserva d'acqua viene ripristinata*/
(self.ospita.Casella.ospita.Elemento.allInstances()->exists(oclIsKindOf(pozzo)) implies acqua=10)

```

Transizione: dissetato->disidratato

Evento:--

Condizione:acqua=0

Azione:

pre:--

post:

```
/*Informale: il cercatore si sposta di una casella verso il pozzo piu' vicino*/  
and  
/*Se e' stato raggiunto un pozzo, la riserva viene ripristinata*/  
(self.ospita.Casella.ospita.Elemento.allInstances()->exists(oclIsKindOf(pozzo)) implies acqua=10)
```

Transizione: disidratato->disidratato

Evento:--

Condizione:acqua=0

Azione:

pre:--

post:

```
/*v. transizione disidratato->disidratato/
```

Transizione: disidratato->dissetato

Evento:--

Condizione:acqua>0

Azione:

pre:--

post:

```
/* v. transizione dissetato->dissetato */
```

FineSpecifica

Specifica Transizioni Classe CercatoreControllato

InizioSpecificaTransizioniClasse CercatoreAutonomo

Le transizioni nonInGioco->inGioco e inGioco->nonInGioco sono analoghe a quelle della classe CercatoreAutonomo

Transizione: dissetato->dissetato

Evento:--

Condizione:acqua>0

Azione:

pre:--

post:

/*Informale:

1. viene richiesto all'utente, mediante interfaccia grafica, la direzione verso cui muoversi.
2. se nella direzione indicata non vi sono caselle adiacenti, allora torna al punto 1.
3. Altrimenti, il cercatore si sposta nella casella indicata (v. sopra)

*/

and

acqua = acqua@pre-0,5

and

let linkOro=self.ospita.Casella.ospita@pre.allInstances()->select(
Elemento.oclIsKindOf(SacchettoOro))

in(
linkOro->notEmpty() implies
/*il sacchetto viene raccolto*/
/*ospita non contiene piu' il link al sacchetto d'oro e per il resto rimane immutata*/
ospita->allInstances()=ospita@pre.allInstances()->excluding(linkOro) and
/*raccolto contiene un (nuovo) link tra il cercatore ed il sacchetto d'oro*/
raccolto.allIntances()->exists(ra | ra.Cercatore = self and ra.SacchettoOro=linkOro.SacchettoOro) and

```

/*il resto di raccolto rimane immutato*/
let
    nuovoLinkRaccolto=raccolto.allInstances()->select(Cercatore=self and
                                                    SacchettoOro=linkOro.SacchettoOro)
in
    raccolto.allInstances()->excluding(nuovoLinkRaccolto)=raccolto@pre.allInstances()
)
and
/*Se si sta passando sopra una casella con pozzo, la riserva d'acqua viene ripristinata*/
(self.ospita.Casella.ospita.Elemento.allInstances()->exists(oclIsKindOf(pozzo)) implies acqua=10)

```

Transizione: dissetato->disidratato

Evento:--

Condizione:acqua=0

Azione:

pre:--

post:

```

/*Informale: analogo ai punti 1-3 della transizione dissetato->dissetato*/
and
/*Se e' stato raggiunto un pozzo, la riserva viene ripristinata*/
(self.ospita.Casella.ospita.Elemento.allInstances()->exists(oclIsKindOf(pozzo)) implies acqua=10)

```

Transizione: disidratato->disidratato

Evento:--

Condizione:acqua=0

Azione:

pre:--

post:

```

/*v. transizione dissetato->disidratato/

```

Transizione: disidratato->dissetato

Evento:--

Condizione:acqua>0

Azione:

pre:--

post:

/* v. transizione dissetato->dissetato */

FIneSpecifica

OSSERVAZIONE: il calcolo dei percorsi minimi da una cella ad un'altra, necessario, in particolare, per realizzare il diagramma degli stati e delle transizioni della classe CercatoreAutonomo, è una procedura articolata, in sostanza corrispondente ad una visita del grafo di caselle che rappresenta la mappa. Per questo corso, non è richiesto scendere ad un livello di dettaglio così profondo (che richiederebbe, tra l'altro, strumenti OCL avanzati). In tali casi rimane, tuttavia, fondamentale fornire una specifica chiara e non ambigua delle azioni che, nella fase di realizzazione, diventeranno codice.

Fase di progetto

Responsabilità sulle associazioni

La seguente tabella delle responsabilità si evince da:

- 1. Requisiti
- 2. Specifica delle operazioni
- 3. Vincoli di molteplicità nel diagramma delle classi.

Associazione	Classe	ha resp.
<i>latitudine</i>	<i>nord</i> <i>sud</i>	$\bar{S}I^{1,2,3}$ $\bar{S}I^{1,2,3}$
<i>longitudine</i>	<i>est</i> <i>ovest</i>	$\bar{S}I^{1,2,3}$ $\bar{S}I^{1,2,3}$
<i>ospita</i>	<i>Casella</i> <i>Elemento</i>	$\bar{S}I^2$ $\bar{S}I^{2,3}$
<i>raccolto</i>	<i>SacchettoOro</i> <i>Cercatore</i>	$\bar{S}I^3$ $\bar{S}I^2$

Strutture di dati

Abbiamo la necessità di rappresentare collezioni omogenee di oggetti, a causa:

- dei vincoli di molteplicità 0..* delle associazioni,
- delle variabili necessarie per vari algoritmi.

Per fare ciò, utilizzeremo le classi del collection framework di Java: Set, HashSet.

Corrispondenza fra tipi UML e Java

Riassumiamo le nostre scelte nella seguente tabella di corrispondenza dei tipi UML.

Tipo UML	Rappresentazione in Java
stringa	String
booleano	boolean
int	int
reale	double
Insieme	HashSet

Tabelle di gestione delle proprietà di classi UML

Riassumiamo le nostre scelte differenti da quelle di default mediante la *tabella delle proprietà immutabili* e la *tabella delle assunzioni sulla nascita*.

Classe UML	Proprietà immutabile
<i>SacchettoOro</i>	<i>peso</i>
<i>Cercatore</i>	<i>nome</i>

Classe UML	Proprietà	
	nota alla nascita	non nota alla nascita
<i>Elemento</i>	-	ospita

Altre considerazioni

Sequenza di nascita degli oggetti: Non dobbiamo assumere una particolare sequenza di nascita degli oggetti.

Valori alla nascita: Non sembra ragionevole assumere che per qualche proprietà esistano valori di default validi per tutti gli oggetti.

Rappresentazione degli stati in Java

Scegliamo di rappresentare gli stati della classe `Cercatore` tramite enumerazione. Quindi `Cercatore` conterrà, in aggiunta al campo dati `nome`, i seguenti elementi:

- una definizione di tipo enumerato:

```
enum Stato{dissetato,disidratato,nonInGioco}
```

- il campo dati privato `Stato statoCorrente`
- una variabile di stato ausiliaria `int acqua`, come suggerito dalla fase di analisi

Rappresentazione degli eventi in Java

Rappresentiamo gli eventi tramite la classe `Evento`, (pre)definita durante il corso

Abbiamo bisogno di estendere `Evento` per modellare l'evento `InizioGioco`

Per gestire gli eventi, la classe `Cercatore` dovrà implementare l'interfaccia `Listener`, (pre)definita durante il corso

Lo scambio di eventi è gestito dalla classe `Environment` (pre)definita durante il corso

API delle classi Java progettate

Omesse per brevità (si faccia riferimento al codice Java).

Fase di Realizzazione

Per la soluzione completa, fare riferimento al codice pubblicato.