SAPIENZA Università di Roma
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corsi di Laurea in Ingegneria Informatica ed Automatica ed in Ingegneria dei Sistemi Informatici

Corso di Progettazione del Software

Esame del **11 giugno 2015**

Tempo a disposizione: 3 ore

Requisiti. L'applicazione da progettare riguarda il gioco "CapsuleMonster", descritto nel seguito. Ad una partita, caratterizzata da una mappa (una stringa), partecipano vari giocatori (almeno 2). Ogni giocatore, caratterizzato da un nome, partecipa ad una sola partita. Un giocatore possiede dei CapsuleMonster, di questi massimo 6 fanno parte della sua squadra di combattimento. I CapsuleMonster hanno un nome ed un livello di energia corrente (un intero non negativo) e possono appartenere ad al più un giocatore. Alcuni CapsuleMonster sono Leggendari ed hanno un fattore moltiplicativo della propria energia (un reale compreso tra 1 e 2).

All'inizio della partita un CapsuleMonster è inizialmente *nascosto* e non appartiene a nessun giocatore. Se è *visto* da un giocatore si mette *in fuga*. Se quando è *in fuga* viene *afferrato* dal giocatore allora se ha un livello di energia inferiore a 5 viene *catturato* e diventa proprietà del giocatore. Altrimenti torna nello stato *nascosto* ma con un livello di energia diminuito di una unità. Nello stato *catturato* il giocatore può: *(i) dirgli di entrare in squadra*, e allora entra nella squadra di combattimento del giocatore, ma solo se il giocatore non ha ancora 6 elementi in squadra; *(ii) dirgli di combattere* con un CapsuleMonster avversario, ma solo se in squadra, e allora se ha un livello di energia (eventualmente moltiplicato per il suo fattore moltiplicativo se leggendario) superiore all'avversario aumenta il proprio livello di energia di una unità, altrimenti lo diminuisce di una unità; *(iii) liberarlo* e allora torna libero, cioè non posseduto dal giocatore (e quindi neanche in squadra se lo era) e passa nello stato *nascosto*.

Siamo interessati alla seguente attività principale. L'attività prende in input una partita P ed un insieme S di CapsuleMonster. Come prima cosa verifica che i giocatori di P non posseggano ancora alcun CapsuleMonster. Se il controllo non va a buon fine termina con un segnale di output "KO". Altrimenti concorrentemente attiva due sottoattività: *partita* e *analisi*. La sottoattività di *partita* avvia tutti i CapsuleMonster in S e tutto quello che serve per iniziare a giocare (i dettagli non interessano) e si mette in attesa del segnale di input di fine da parte dell'utente che fa terminare la partita. La sottoattività di *analisi* calcola e stampa l'insieme dei giocatori della partita P. Una volta che tali sottoattività sono state completate, l'attività principale invia un segnale di output "OK" e termina.

Domanda 1. Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo del diagramma delle classi (inclusi vincoli non esprimibili in UML), diagramma stati e transizioni per la classe *CapsuleMonster*, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica dell'attività principale e delle sottoattività NON atomiche, motivando, qualora ce ne fosse bisogno, le scelte di progetto.

Domanda 2. Effettuare il progetto, illustrando i prodotti rilevanti di tale fase e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio definire solo le responsabilità sulle associazioni del diagramma delle classi.

Domanda 3. Effettuare la realizzazione, producendo un programma JAVA e motivando, qualora ce ne fosse bisogno, le scelte di progetto. È obbligatorio realizzare in JAVA solo i seguenti aspetti dello schema concettuale:

- La classe `CapsuleMonster`, con la classe `CapsuleMonsterFired`, e le classi JAVA per rappresentare le *associazioni* di cui `CapsuleMonster` ha responsabilità.
- L'*attività principale* (NON le sue sottoattività).