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 **9 luglio 2015**

Tempo a disposizione: 3 ore

Requisiti. L'applicazione da progettare riguarda il gioco "DroneFight", descritto nel seguito. Ad una partita, caratterizzata da una mappa (una stringa), partecipano vari giocatori (almeno 3). Ogni giocatore, caratterizzato da un nome, partecipa ad una sola partita. Un giocatore possiede almeno 6 *Drone*. I Drone hanno un nome ed un livello di energia massimo (un intero non negativo) e possono appartenere ad al più un giocatore. I Drone sono suddivisi in due categorie *AttackingDrone* e *DefendingDrone*. E' di interesse memorizzare se un AttackingDrone ha combattuto con un DefendingDrone, e quante volte ha vinto l'uno e quante volte ha vinto l'altro.

All'inizio della partita un AttackingDrone è *a riposo* (e appartiene ad un giocatore). Quando a riposo, il suo giocatore può chiedergli di *attaccare* un altro Drone utilizzando un certo livello di energia. In tal caso lui verifica, *(i)* che il Drone d'attaccare sia un DefendingDrone di un giocatore diverso dal suo e *(ii)* che il livello di energia dell'attacco richiesto non sia superiore al suo massimo. In caso negativo ignora la richiesta, altrimenti, in caso affermativo, *attacca* il DefendingDrone richiesto passando in uno stato di *attesa* della risposta da parte del DefendingDrone attaccato. Quando il DefendingDrone attaccato *risponde*, con un dato livello di energia, se questo è minore di quello assegnato all'AttackingDrone per l'attacco dal suo giocatore allora vince altrimenti perde, in ogni caso memorizza queste informazioni sul diagramma delle classi e passa nello stato *a riposo*. Il comportamento del DefendingDrone non è di interesse per la prova.

Siamo interessati alla seguente attività principale. L'attività prende come parametro di input una partita P. Come prima cosa verifica che tutti i giocatori di P posseggano sia AttackingDrone che DefendingDrone. 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 Drone di P e tutto quello che serve per iniziare a giocare (i dettagli non interessano) e poi 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 (segnale di output) l'insieme degli AttackingDrone e dei DefendingDrone di ciascun giocatore. 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 *AttackingDrone*, diagramma delle attività, specifica del diagramma stati e transizioni, e specifica dell'attività principale (NON delle sottoattività), 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 `Drone`, la sottoclasse `AttackingDrone` con la classe `AttackingDroneFired`, e le classi JAVA per rappresentare le *associazioni* di cui `Drone` e `AttackingDrone` hanno responsabilità.
- L'*attività principale* (NON le sue sottoattività).