SAPIENZA Università di Roma
Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Ingegneria Informatica ed Automatica
Corso di Progettazione del Software
Esame del **12 luglio 2019**
Tempo a disposizione: 3 ore

Requisiti. L'applicazione da progettare riguarda una variante del gioco della battaglia navale. Un gioco è caratterizzato da un nome (una stringa), e ha un arbitro e un insieme di giocatori (almeno 2). Dell'arbitro interessa un codice identificativo (un intero). L'arbitro appartiene ad un solo gioco. Ciascun giocatore ha un nome (una stringa) e anche esso appartiene ad un solo gioco. Inoltre ogni giocatore ha un insieme di navi. Le navi hanno un nome (NB: non è di interesse data una nave sapere a quali giocatori appartiene). Le navi sono divise in due categorie: semplici e speciali. Le navi semplici hanno una "posizione annotata". Una posizione annotata è costituita da due interi (le coordinate) e un booleano che indica se la posizione è stata colpita. Le speciali hanno un insieme non vuoto di posizioni annotate (non necessariamente adiacenti).

Siamo interessati al comportamento dei giocatori. Un giocatore è inizialmente a *riposo*. Se riceve il comando di *si gioca* da parte dell'arbitro del gioco annota a *false* tutte le posizioni delle proprie navi e passa *in gioco*. Quando *in gioco* riceve l'evento *colpo* con payload due interi, verifica se essi corrispondono alla posizione di una delle sue navi, in caso affermativo mette a *true* l'annotazione della posizione, altrimenti manda a sua volta un *colpo* scegliendo in due interi secondo una funzione predefinita i cui dettagli non interessano). Se quando *in gioco* il giocatore riceve il comando *fine* da parte dell'arbitro del gioco torna a *riposo*.

Siamo interessati alla seguente attività principale. L'attività prende in input un gioco G e verifica che non ci siano sovrapposizioni tra le posizioni delle navi dei vari giocatori. Se la verifica non va a buon fine, l'attività termina segnalando in output un errore. Altrimenti concorrentemente esegue le seguenti due sottoattività: *(i)* gioco e *(ii)* analisi. La sottoattività di gioco *(i)* avvia il gioco attivando tutti i giocatori di G e l'arbitro mandando opportuni eventi (i dettagli non interessano). Poi si mette in attesa del segnale di fine-gioco che interrompe il gioco stesso. La sottoattività di analisi *(ii)* calcola un report (una stringa) con l'elenco dei nomi delle nami dei vari giocatori di G, Una volta che tali sottoattività sono state completate, l'attività principale calcola il giocatore che ha vinto, cioè colui le cui navi hanno complessivamente meno posizioni colpite, poi manda un segnale di output con il report calcolato dalla sottoattività di analisi con in aggiunta il nome del vincitore.

Domanda 1. Basandosi sui requisiti riportati sopra, effettuare l'analisi producendo lo schema concettuale in UML per l'applicazione, comprensivo di: diagramma delle classi (inclusi eventuali vincoli non esprimibili in UML); diagramma stati e transizioni per la classe *Giocatore*; diagramma delle attività; specifica del diagramma stati e transizioni; segnatura dell'attività principale, sottoattività non atomiche, atomiche e segnali di input/output. Si noti che NON è richiesta la specifica delle attività. Motivare, 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. In particolare 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. In particolare realizzare in JAVA SOLO i seguenti aspetti dello schema concettuale:

- La classe `Giocatore` con la classe `GiocatoreFired`, le classi JAVA per rappresentare le *associazioni* di cui la classe `Giocatore` ha responsabilità.
- L'*attività principale* e le sue eventuali sottoattività NON atomiche.