Testing e verifica del software – 15 febbraio 2017

Gioco PariODispari

Ci sono due giocatori (1-l'utente e 2-il pc, per comodità) che possono tirare entrambi contemporaneamente un numero da 1 a 5. Se la somma è pari vince il giocatore 1, se dispari il giocatore 2 (il PC). Ogni giocatore inizialmente ha 5€. Se un giocatore perde viene scalato 1€ e viene aggiunto all'avversario. Il gioco continua finché uno dei due giocatori arriva a 0€, poi si ferma la partita.

Compito

In un file word annota tutto quello che fai. Aggiungi anche gli screenshot per i punti principali (ad esempio copertura ...). Per effettuare gli screenshots usa lo strumento di cattura di windows. Anche la documentazione sarà valutata per ogni singolo progetto (circa del 15%).

Sviluppa un progetto in eclipse per ognuna delle domande. Ogni progetto dovrà avere come nome il tuo COGNOME_**ES** (con ES il tipo dell'esercizio – vedi titoli degli esercizi). Puoi anche andare in parallelo (ad esempio modello NUSMV e codice Java). Metti i progetti e il documento word in una directory con nome COGNOME_NOME (che puoi usare come workspace). Alla fine crea un file zip e consegna quello.

Commenta per bene tutto il codice che scrivi.

Java

In Java scrivi una classe **PariDispari** che implementa il gioco descritto. Ha almeno due metodi:

```
public boolean tira(int utente, int pc)
```

che calcola il nuovo budget dei due giocatori dopo che hanno tirato i numeri `utente` e `pc`, e restituisce true se e solo se i budget cambiano; se i valori non sono validi non fare nulla.

```
public boolean finito()
```

Restituisce true se e solo se il gioco è finito.

1. ES = TESTING

Scrivere l'implementazione e i casi di test con Junit. Scrivi una classe di test PariDispariPCVince con un caso di test che mostri uno scenario in cui il PC vince la partita.

Esegui il controllo della copertura (istruzioni, branch, decisioni, MCDC). Se il caso sopra non è sufficiente ad avere una copertura soddisfacente, aggiungi i casi di test necessari (in una classe separata per ogni copertura). Se ritieni che qualche decisione non sia copribile giustificalo.
Metti la directory test con i test separata (oltre src).
Prova con Randoop a generare casi di test (con pochi casi di test, tipo 10) e prova a valutarne la copertura.

2. ES = JML

Scrivi i contratti JML. Cerca di scrivere sia le precondizioni, che le postcondizioni dei metodi.

Cerca di spostare nelle precondizioni alcuni controlli che facevi all'inizio dei metodi nel precedente esercizio. Cerca di scrivere anche invarianti.
Prova i contratti JML con una classe main in cui chiami i diversi metodi.

Prova anche a modificare il codice e controlla che i contratti siano violati. Documenta bene le violazioni e le loro cause in commenti e nel file di documento.

3. ES = KEY Verifica dei programmi

CONSIGLIO. Inizia da una classe con contratti **vuoti** (true) e **aggiungi** mano a mano i contratti però sempre dimostrandone la loro correttezza – cioè le prove devono essere sempre chiuse **vere**. La valutazione dipenderà dal massimo contratto che riuscite a dimostrare vero (contratti non provati non verranno valutati).

Se la tua classe contiene dei metodi delle librerie (tipo system.out.println) eliminali. Anche Random va eliminata se l'hai usata.

4. ES = NUMSV

Specificare il sistema di gioco in NuSMV e le sue proprietà usando LTL e/o CTL.

Tra le proprietà desiderate (alcune potrebbero essere false) c'è:

- P1: il PC non può mai vincere
- P2: prima o poi qualcuno vince (se ti sembra è po' ambiguo, cerca di specificare meglio e spiegare).

Provare se sono vere oppure no. Se sono false presentare un contro esempio (se viene prodotto).

Aggiungi anche **altre** proprietà che ti sembrano importanti, cerca di provarle e commentale per bene (valutazione).

Per ogni proprietà aggiungi una descrizione in Italiano di quello che vuol dire.

5. ES = FSM (MBT con le FSM)

Scrivi una FSM usando ModelJunit. Prova a fare disegnare il diagramma e valuta la copertura (sia con online testing che offline testing). Non è necessario rappresentare ogni combinazione di numeri con una azione diversa.

Salva il disegno della macchina come immagine.

Prova ad introdurre degli errori nell'implementazione e guarda se li trovi con l'online testing.

6. ES = COMB (Combinatorial testing)

Rappresenta in combinatorial testing una singola giocata (che corrisponde alla singola chiamata del metodo `tira )` con i due numeri tirati dall'utente e i due budget prima e dopo la chiamata e se il gioco è finito (in totale 7 parametri). Introduci anche gli opportuni vincoli.

Applica il combinatorial testing.

Prova a generare la copertura pairwise.