Rush Hour

Si vuole scrivere un modello e una implementazione semplificata per il puzzle RushHour.
Nel RushHour una griglia 6x6 contiene 6 macchine numerate da 1 a 6 (ogni macchina per semplicità ha dimensione 1x1). Puoi pensare di rappresentare la griglia come array bidimendionale di interi con le celle vuote indicate con 0 oppure rappresentare la posizione delle 6 macchine con una coppia di interi da 0 a 5 per ogni macchina.

	0	1	2	3	4	5
0				6		
1						2
2			1			3
3						4
4		5				
5						

Obiettivo del gioco è portare la macchina numero 1 (chiamata macchina rossa) all'uscita della griglia, che corrisponde alla cella con indici (2,5) (o 3,6 se le coordinate iniziano da 1) indicata dallo sfondo scuro. Quando la macchina rossa raggiunge l'uscita il gioco termina.

La configurazione iniziale del gioco è quella indicata.

Il sistema deve permettere lo spostamento di una macchina in una casella adiacente libera. Deve inoltre essere possibile sapere se la macchina rossa è all'uscita.

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.

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.

Note riguardo Java.

La classe deve avere come minimo un metodo boolean `moveCar` con segnatura:

```
moveCar(int row, int col, int dir)
```

che permetta di muovere una macchina sulla griglia; il metodo ritorna true se la macchina è stata spostata, false altrimenti. I parametri row e col indicano una cella della griglia contenente la macchina che si vuole spostare; dir indica una direzione nel seguente modo:

1. verso l'alto;
2. verso destra;
3. verso il basso;
4. verso sinistra.

Il metodo dove controllare che i valori passati siano corretti. Se una macchina non si può spostare in direzione dir perché ostruita da un'altra macchina o dai limiti della griglia, non deve essere spostata.

C'è anche il metodo **`redCarAtExit`** che dice se la macchina rossa ha raggiunto l'uscita;

In Java ti consiglio di rappresentare la griglia come un array bidimensionale.

1. ES = TESTING

Scrivere l'implementazione e i casi di test con Junit. Scrivi una classe di test `RedCarExitTest` per lo scenario in cui a partire dalla configurazione iniziale la macchina rossa esce.

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).
Metti la directory test con i test separata (oltre src).
Prova con Randoop a generare casi di test 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 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

Prova a verificare qualche contratto. Ricorda: togli tutti i system.out e l'uso delle librerie (ad esempio Math). Ricorda di creare un nuovo progetto e convertirlo in key. Deve apparire l'icona di key e la directory con i proofs. Evita anche di usare i float e double.

4. ES = NUMSV

GRIGLIA 6x6 MA SOLO 3 MACCHINE: 1, 2 e 3.

In questo caso consiglio di rappresentare la board con le posizioni delle macchine e come variabili la macchina che si vuole spostare e la direzione.

Scrivi alcune proprietà sia di liveness che di safety e provale. Spiegale nel documento.

5. ES = FSM (MBT con le FSM)

In questo caso considera una configurazione semplificata con due solo auto e una griglia 3x3.

	0	1	2
0			
1		1	2
2			

Modella l'evoluzione del sistema con una FSM. Implementa la FSM con modelJunit.

Salva il disegno della macchina come immagine.

6. ES = IDM (Input domain modelling)

Applica IDM al metodo di spostamento delle macchine. Individua gli input da passare e spiega le scelte che hai fatto. Scrivi i corrispondenti casi di test Junit.

7. ES = COMB (Combinatorial testing)

Applica il combinatorial testing al sistema semplificato utilizzato per NUSMV in modo di avere tutte le configurazioni possibili. Prova a generare la copertura pairwise delle posizioni. Includi i necessari constraints. Se riesci aggiungi anche una variabile che indica se l'auto rossa è nella posizione di uscita.