**SISTEMI OPERATIVI IIN/IEL/IDT**
**INFORMATICA INDUSTRIALE E SISTEMI OPERATIVI IDI**

## SISTEMI DI ELABORAZIONE P.O.
## prova scritta preliminare del 16.04.2004

Nome: _____ Cognome: _____

Un numero K di giocatori partecipa ad un gioco il cui schema base è il seguente:

- I giocatori condividono un "campo di gioco" in forma di griglia di dimensioni assegnate (NxM), in cui sono disposti in modo casuale un insieme di obiettivi. Ogni cella della griglia ha associato un valore. Ciascuna cella corrispondente ad un obiettivo ha associato un valore intero variabile nell'intervallo [1:MAX]. Le celle del campo non coperte da obiettivi hanno associato un valore 0;

- I giocatori non hanno conoscenza della disposizione degli obiettivi nel campo di gioco, ma possono conoscere solo la dimensione del campo. Obiettivo di un giocatore è di individuare gli obiettivi fornendo in modo casuale le coordinate di un punto del campo. Come effetto della selezione di una cella il giocatore riceve il valore associato, nel campo, alla coordinata fornita. Il valore della cella selezionata è posto a 0;

- Dopo ogni tentativo un giocatore deve aspettare che almeno un altro giocatore esegua una selezione, prima di poter tentare nuovamente;

- Il gioco termina quando tutti gli obiettivi sono stati selezionati. Ogni giocatore si accorge di questo ricevendo il valore –1 come risultato di una selezione sul campo. Ogni giocatore che si accorge del termine del gioco segnala il punteggio totale riportato nei suoi tentativi.

Supponendo che ogni giocatore possa essere rappresentato come un processo separato di esecuzione, si scriva:

- la soluzione, in linguaggio JAVA, che coordina tra loro i giocatori. È necessario scrivere: la classe che arbitra l'avvio dei Thread giocatori leggendo da linea di comando il loro numero e la dimensione del campo di gioco; la classe che modella i giocatori; la classe che rappresenta il campo di gioco ed i metodi per accederlo (per questa classe è possibile ipotizzare l'esistenza di un metodo `popola()` che inizializza il campo di gioco disponendo gli obiettivi con il loro valore) – anno accademico 2003/04;

- la soluzione che coordina tra loro i giocatori utilizzando uno pseudo linguaggio e come strumento di sincronizzazione: (a) un monitor; (b) semafori – anno accademico 2002/03.

Soluzione in linguaggio JAVA

```
public class Arbitro
{
	public static void main(String args[]) {
		// crea il campo di gioco
		Campo campo = new Campo(Integer.parseInt(args[1]),Integer.parseInt(args[2]));
		campo.popola();

		int giocatori = Integer.parseInt(args[0]);

		System.out.println("Sessione di gioco con " + giocatori + " giocatori");

		for (int i=0;i<giocatori;i++) {
			// crea il numero di giocatori richiesto
			Giocatore player = new Giocatore(i,campo);

			player.start();
		}
	}

}



public class Giocatore extends Thread
{
	private Campo campo;  // campo di gioco
	private int number;	// identificativo del giocatore
	private int score;

	public Giocatore(int n,Campo c) {
		number = n;
		campo = c;
		score = 0;	// score accumulato dal giocatore
	}

	public void run() {
		int value;
		int i = (int) (Math.random()*campo.getRow());
		int j = (int) (Math.random()*campo.getColumn());

		System.out.println("Selezione del giocatore " + number + " = (" + i + "," + j + ")");

		boolean stop = false;
		while (!stop) {
			value = campo.seleziona(number,i,j);
			if (value!=-1)
				score += value;
			else
				stop = true;
		}
		System.out.println("Giocatore " + number + " termina con score = " + score);
	}

}
```

```
class Campo
{
	public static final int MAX_SCORE = 5;
	private int last;		// ultimo ad aver giocato
	private int count;		// numero di elementi della matrice occupati
	private int row;		// righe del campo
	private int column;		// colonne del campo
	private int[][] field;		// campo di gioco

	public Campo(int r, int c) {		// costruttore
		count = 0;
		last = -1;
		row = r;
		column = c;
		field = new int[row][column];
	}

	public int getRow() {
		return row;
	}

	public int getColumn() {
		return column;
	}

	// dispone gli obiettivi sul campo di gioco.  È simulato con celle di valore diverso
	public void popola() {
		int score;
		for (int i=0;i<row;i++)
			for (int j=0;j<column;j++) {
				score = (int) (MAX_SCORE * Math.random());
				field[i][j] = score;
				if (score!=0)
					count++;
			}
	}

	// metodo di accesso al campo di gioco
	public synchronized int seleziona(int number,int i,int j) {
		// non puo' giocare due volte di seguito
		while (number==last) {
			try {
				wait();
			}
			catch (InterruptedException e) {
				System.out.println("Thread " + number + "interrotto");
			}
		}
		int value = field[i][j];
		field[i][j] = 0;
		last = number;
		System.out.println("Score per giocatore " + number + " = " + value);
		count --;		// decrementa il numero di elementi ancora presenti

		if (count<=0) {
			System.out.println("Sessione di gioco terminata");
			value = -1;
		}
		notifyAll();
		return value;
	}
}
```

Soluzione in pseudocodice con uso di Monitor

**Monitor**

```
Monitor Campo {
        Int [][] field;
        int last;
        int count;
        condition x;      // variabile di condizione

        // inizializza sarà chiamato una sola volta prima dell'inizio del gioco
        // per popolare il campo di gioco
        inizializza (int r, int c) {
                count = 0;
                last = -1;
                field = new int[r][c];
        }

        seleziona (int number, int i, int j) {
                if (number==last)
                        x.wait();          // si sospende se è l'ultimo ad aver giocato

                int value = field[i][j];
                field[i][j] = 0;
                last = number;
                count --;          // decrementa il numero di celle con score diverso da zero ancora presenti

                if (count<=0)
                        value = -1;       // la sessione di gioco termina

                x.signal();       // risveglia un eventuale processo sospeso
                return value;
        }
}
```

**Processo Giocatore i**

Una istanza m del monitor Campo è condivisa tra i processi giocatore. Le dimensioni del campo sono row e column. La funzione `random()` restituisce un numero compreso in [0:1].

```
int score = 0;
int value;
int id = i;
boolean stop = false;
while (!stop) {
        int x = random()*(row-1);
        int y = random()*(column-1);
        value = m.seleziona(id,x,y);
        if (value!=-1)
                score += value;
        else
                stop = true;
}
// il giocatore termina con il punteggio accumulato in score
```

## Soluzione in pseudocodice con uso di Semafori

Variabili e semafori condivisi tra i processi giocatore:

- mutex(1), semaforo di mutua esclusione per l'accesso alla sezione critica rappresentata dalla matrice che modella il campo di gioco;
- turno(0), semaforo su cui si sospende un processo giocatore se è stato l'ultimo a giocare;
- int last=-1, identificatore dell'ultimo giocatore ad aver giocato;
- int waiting = 0, processo sospeso in attesa che un altro giochi.

Le funzioni `getlast()` e `seleziona()` ritornano, rispettivamente il numero dell'ultimo giocatore e lo score della giocata attuale (–1 nel caso di assenza di obiettivi).

### Processo Giocatore i

```
Int id;    // identificatore del giocatore
int score = 0
boolean stop = false;
while (!stop) {
        int x = random()*(row-1);
        int y = random()*(column-1);

        wait (mutex)
        if (i==getlast()) {
                waiting ++;
                signal (mutex);
                wait (turno);
                wait (mutex);
                waiting --;
        }

        int value = seleziona(id,,x,y);
        signal (mutex);

        if (value!=-1)
                score += value;
        else
                stop = true;

        wait (mutex)
        if (waiting>0)
                signal(turno);    // sveglia un eventuale processo sospeso sul semaforo turno
        signal (mutex);
}
// il giocatore termina con il punteggio accumulato in score
```