Laboratorio 10 – Esercizi

Elementi di Informatica e Programmazione

Lab 10 – Es 1

Progettare il modulo **triangle.py** che contenga la definizione della classe **Triangle**, i cui esemplari rappresentino triangoli geometrici nel piano cartesiano, oltre a codice di collaudo (ad esecuzione condizionata, come al solito).

Per prima cosa definire la classe **Point** che descriva un punto nel piano cartesiano: il costruttore riceve le coordinate **x** e **y** del punto; il metodo di esemplare **x()** restituisce la coordinata **x** e il metodo di esemplare **y()** restituisce la coordinata **y**, mentre il metodo di esemplare **xy** restituisce una tupla contenente, nell'ordine, la coordinata **x** e la coordinata **y**; il metodo di classe **delta(p1, p2)** calcola e restituisce la distanza tra il punto **p1** e il punto **p2**.

La classe **Triangle** ha un costruttore che richiede tre punti (cioè tre esemplari di **Point**) e solleva l'eccezione **ValueError** se tali punti non costituiscono un triangolo (ricordando che tre punti definiscono un triangolo se e solo se ciascuno dei tre segmenti da essi definiti è minore della somma degli altri due). Per decidere quali siano le migliori informazioni di stato per un tale oggetto, è bene analizzare la sua interfaccia pubblica, così definita:

- il metodo **area()** restituisce l'area del triangolo (può essere utile ricordare la formula di Erone)
- il metodo **height()** restituisce l'altezza relativa al lato maggiore
- il metodo **isScalene()** restituisce **True** se e solo se il triangolo è scaleno
- il metodo **isIsosceles()** restituisce **True** se e solo se il triangolo è isoscele (ovviamente un triangolo equilatero è anche isoscele)
- il metodo **isEquilateral()** restituisce **True** se e solo se il triangolo è equilatero (ovviamente un triangolo equilatero è anche isoscele)
- il metodo **isRight()** restituisce **True** se e solo se il triangolo è rettangolo (ovviamente un triangolo rettangolo può anche essere isoscele o scaleno)

Lab 10 – Es 2

Riprendere in esame il Gioco di Nim visto nei precedenti esercizi nim.py e nim2.py e scrivere il programma nim3.py con cui un giocatore umano possa giocare a Nim contro il computer, effettuando una progettazione orientata agli oggetti. Il comportamento del programma deve essere identico al comportamento di nim2.py.

Progettare le **classi** seguenti:

- **NimPile** che rappresenta il mucchio di biglie usato durante una partita. Deve avere:
  - il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
  - il metodo **getTotal** che restituisce il numero di biglie presenti nel mucchio
  - il metodo **take** che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori
- **NimHumanPlayer** che rappresenta il giocatore umano. Deve avere:
  - il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e chiede all'utente di fare una mossa (ripetendo la richiesta finché non viene indicata una mossa valida), dopodiché mette in atto la mossa (invocando il metodo **take** del mucchio)
  - dato che gli esemplari di tale classe non hanno bisogno di variabili di esemplare, non hanno bisogno di un costruttore
- **NimComputerPlayer** che rappresenta il computer che gioca. Deve avere:
  - il costruttore che riceve un valore booleano, **True** se e solo se il computer deve giocare in modo *intelligente*
  - il metodo **isExpert** che restituisce **True** se e solo se il computer sta giocando in modo *intelligente*
  - il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo take del mucchio)
- **NimGame** che rappresenta un'intera partita. Deve avere:
  - il costruttore che non riceve parametri
  - il metodo **play** che gioca un'intera partita e visualizza il vincitore; se il vincitore è il computer, visualizza anche la modalità di gioco (*intelligente* o *stupido*)

Il programma principale è semplicemente costituito da un ciclo "infinito" che, ad ogni iterazione, crea un esemplare di NimGame, ne invoca il metodo play e, poi, chiede all'utente se vuole fare un'altra partita oppure no.

Lab 10 – Es 3 (1/4)

Scrivere il programma **maze.py** che svolga il ruolo di aiutante nell'uscita da un labirinto.

La classe **Maze** (da progettare) deve rappresentare il labirinto, che è una matrice rettangolare di posizioni, ciascuna delle quali può essere un *corridoio* (di transito) oppure un *muro* invalicabile. Nel labirinto ci si può spostare soltanto da un corridoio a un altro e gli spostamenti possono avvenire soltanto in una delle quattro direzioni cardinali (N=nord, E=est, S=sud, W=ovest), cioè lungo una riga o una colonna della matrice e non in diagonale. Dato un punto di partenza all'interno del labirinto, l'obiettivo è ovviamente quello di arrivare in un corridoio che si trovi sul margine esterno del labirinto. Ciascuna posizione ha 4 posizioni adiacenti (nelle 4 direzioni cardinali), con l'eccezione delle posizioni che si trovano sui margini esterni del labirinto.

Esempio di visualizzazione di un labirinto (usiamo uno spazio per un corridoio e un asterisco per un muro):

```
* *******
*   * * *
* ***** *
* * *   *
* * *** *
*   *   *
***** * *
*   * * *
******* *
```

Lab 9 – Es 3 (2/4)

Ciascuna posizione è individuata da una coppia di indici, di riga e di colonna, numerati a partire da 0 nell'angolo in alto a sinistra e non negativi.

Le informazioni di stato di un oggetto di tipo **Maze** sono memorizzate in un dizionario in cui le chiavi sono tuple che contengono una coppia di indici che individua la posizione di un corridoio (le posizioni che contengono un muro non vengono memorizzate esplicitamente) e i cui valori sono liste di tuple definite allo stesso modo e rappresentanti i corridoi adiacenti (quindi la dimensione di una di queste liste può variare tra 0 e 4).

La classe deve avere i metodi seguenti:

- il costruttore **__init__** deve ricevere il nome di un file che contiene la descrizione di un labirinto nel formato qui visualizzato (asterischi e spazi), per poi leggerlo e creare le informazioni di stato descritte
- il metodo **__repr__** deve restituire una stringa contenente la visualizzazione del labirinto così come illustrato (senza visualizzarla)
- il metodo **help**, quando invocato, deve visualizzare il labirinto con l'aggiunta di informazioni sufficienti per uscire dal labirinto a partire da un corridoio qualsiasi, se da quel punto è possibile farlo, implementando l'algoritmo descritto nel seguito

Il programma deve svolgere alcuni semplici collaudi della classe **Maze**.

Lab 9 – Es 3 (3/4)

Una possibile rappresentazione delle informazioni sufficienti per uscire dal labirinto, trovandosi in un qualsiasi corridoio, è che nel corridoio stesso ci sia scritto quale direzione prendere, cioè verso quale altro corridoio spostarsi. Questa informazione può semplicemente essere una delle 4 lettere che indicano uno dei punti cardinali (oppure un punto interrogativo se da quel punto non è possibile uscire). Ad esempio, il labirinto precedente diventa:

```
*N*******
*NWW*?*S*
*N*****S*
*N*S*EES*
*N*S***S*
*NWW*EES*
*****N*S*
*???*N*S*
*******S*
```

Come possiamo generare queste informazioni all'interno del metodo **help**? Oltre al dizionario che costituisce le informazioni di stato del labirinto, usiamo un altro dizionario che abbia le stesse chiavi (cioè tuple che rappresentano i corridoi) ma, come valori, una delle lettere che possono caratterizzare un corridoio (cioè ?, N, E, S, W). All'inizio, in tutti i corridoi viene scritto ?, dopodiché nei corridoi che si trovano su un margine esterno del labirinto viene scritta la lettera che indica la direzione da seguire per uscire dal labirinto.

Lab 9 – Es 3 (4/4)

Quindi, si inizia un ciclo:

```
done = False
while not done :
  done = True
  per ogni corridoio c nel dizionario delle direzioni
    se c contiene ?
      cerco c nel dizionario delle adiacenze
        ottenendo la lista delle posizioni adiacenti a c
      per ogni posizione p adiacente a c
        cerco p nel dizionario delle direzioni
        se p ha una direzione d diversa da ?
          nel dizionario delle direzioni, il valore associato a c diventa d
          done = False
          break
```

Il metodo help deve visualizzare il labirinto dopo ogni iterazione del ciclo.