Corso di Programmazione e Algoritmi 2
Modulo di Laboratorio
A.A. 2024/2025

Regole per lo svolgimento del progetto:

- Il progetto può essere svolto **singolarmente** o **in gruppi di qualsiasi dimensione**.
  In ogni caso, ciascuno studente dovrà poi **singolarmente** discutere il progetto dimostrando di essere in grado di modificare e/o aggiungere alcune parti. In caso di esito sufficiente di tale discussione, **verrà attribuito il punteggio da 0 a 3 punti** che è additivo rispetto al voto della parte da 6 CFU di teoria.
- Il progetto deve essere consegnato **almeno 7 giorni prima** della prova orale di *Programmazione e Algoritmi 2* relativa all'appello in cui si desidera discutere il progetto. Non è necessario che la restante parte dell'esame (da 6 CFU) sia svolta nel medesimo appello.
- Il progetto deve contenere le classi e i metodi richiesti **rispettandone esattamente il nome, il tipo e l'ordine dei parametri formali, ed il tipo di ritorno**. Si tenga presente che può essere necessario sviluppare anche altre classi (non pubbliche) oltre quelle richieste.
- Si tenga presente che nella specifica **non sono presenti** tutti i campi di istanza che devono essere opportunamente aggiunti da voi nella consegna.
- Le **proprietà in lettura e scrittura** non sono tutti presenti nella specifica. Deve essere vostra cura aggiungerle, dove occorrono, in modo opportuno.
- Dove si rende necessario, vanno implementati anche i metodi **__eq__**.
- Si è naturalmente liberi di sviluppare (e anzi siete incoraggiati a farlo) classi e/o metodi aggiuntivi, laddove lo si ritenga utile o necessario.
- Deve essere consegnato via email all'indirizzo **luca.moscardelli@unich.it** un file compresso contenente tutti i file sorgenti e avente come nome
  [python] progetto 2024_2025.zip
  l'email deve avere come oggetto
  [python] consegna progetto programmazione 2024/2025
  e deve contenere nel testo la lista (Cognome, Nome, Matricola e Email) di tutti gli studenti che hanno svolto il progetto consegnato.
  In caso di consegna in gruppo, la consegna deve essere effettuata da uno dei componenti del gruppo, mettendo in CC nell'email anche l'indirizzo di tutti gli altri componenti.
- Il modulo **campominato.py** deve funzionare in modo autonomo, anche senza il modulo **gui.py**, e deve possedere tutte le indicazioni di tipo in modo da passare senza errori il **type checking** di livello **strict**.
- L'interfaccia grafica del modulo **gui.py** va sviluppata usando la libreria *EzGraphics*. In questo modulo non è richiesto il type checking.

Testo del progetto:

Sviluppare in **Python** due moduli (**campominato.py** e **gui.py**) per la rappresentazione ed esecuzione di partite del famoso videogioco solitario "campo minato". Per la descrizione del gioco si faccia riferimento alla pagina Wikipedia https://it.wikipedia.org/wiki/Campo_minato_(videogioco).
Al link https://g.co/kgs/2jsvCv è invece disponibile una versione giocabile online del gioco.
Si noti che una casella del gioco può avere fino a 8 caselle adiacenti (in direzione N, S, E, W, NE, NW, SE, SW).

Impostazione formale

Il modulo **campominato.py** deve contenere le seguenti **classi**:

- `Tabellone`
  Classe pubblica che rappresenta un tabellone del gioco.
  Contiene i seguenti metodi:
  - Costruttore
    `__init__(self, p:Partita)`
    che costruisce un tabellone, relativo alla partita `p`, avente `p.altezza` righe e `p.larghezza` colonne.

  - Metodo di classe per costruttore "di copia"
    `copia_tabellone (t:Tabellone) -> Tabellone`
    che costruisce un tabellone che è la copia del tabellone `t`. Attenzione: le modifiche eventualmente fatte al nuovo tabellone non devono ripercuotersi su `t` e viceversa.

  - Metodo `segna_casella(self, riga:int, colonna:int) -> None`
    che contrassegna la casella di riga `riga` e colonna `colonna` come contenente una mina, se non era già stata contrassegnata tale; altrimenti toglie il contrassegno dalla casella stessa.

  - Metodo `is_segnata(self, riga:int, colonna:int) -> bool`
    che restituisce `True` se la casella di riga `riga` e colonna `colonna` è stata contrassegnata come casella contenente una mina, False altrimenti.

  - Metodo `scopri_casella(self, riga:int, colonna:int) -> None`
    che segna come scoperta la casella di riga `riga` e colonna `colonna`.

  - Metodo `is_coperta (self, riga:int, colonna:int) -> bool`
    che restituisce `True` se la casella di riga `riga` e colonna `colonna` è coperta nel tabellone, `False` altrimenti.

- Metodo `__str__(self) -> str`
  che deve restituire una stringa che rappresenta il tabellone corrente secondo la seguente convenzione:
  - `C` corrisponde a una casella **coperta** che **non contiene mine** e **non è segnata come contenente mine**
  - `D` corrisponde a una casella **coperta** che **non contiene mine** ed **è segnata come contenente mine**
  - `X` corrisponde a una casella **coperta** che **contiene mine** e **non è segnata come contenente mine**
  - `Y` corrisponde a una casella **coperta** che **contiene mine** ed **è segnata come contenente mine**
  - `Z` corrisponde a una casella **scoperta** che **contiene** mine (ce ne può essere solo una, poiché appena si scopre una casella con una mina il gioco termina)
  - Le cifre da `0` a `8` corrispondono a una casella **scoperta non contenente mine**, con la cifra che indica il numero di caselle adiacenti contenenti mine.

  Ad esempio,

```
CCCCCCCCCC
CCCXCCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCXCCCCCCC
CCCCCCCXCC
```

  corrisponde al tabellone relativo alla configurazione iniziale di una partita con larghezza 10, altezza 6 e 7 mine (in cui nessuna casella è segnata);

```
CCCCCCCCCC
CCCXCCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCX2222CCC
CCC1001XCC
```

  corrisponde al tabellone relativo alla configurazione ottenuta a partire da quella del precedente esempio dopo che è stata scoperta la casella di riga 6 e colonna 5, e in cui nessuna casella è segnata.

- `Partita`

  Classe pubblica che rappresenta una partita del gioco.
  Contiene i seguenti metodi/proprietà:
  - Costruttore
    `__init__(self, larghezza:int, altezza:int, n_mine:int)`
    che costruisce una partita del gioco contraddistinta da un tabellone avente `altezza` righe e `larghezza` colonne, nel quale sono posizionate in modo casuale `n_mine` mine. Lo stato della partita è impostato su **partita in corso**;
  - Proprietà in lettura `larghezza` (di tipo int)
  - Proprietà in lettura `altezza` (di tipo int)
  - Proprietà in lettura `n_mine` (di tipo int)
  - Proprietà in lettura `stato_corrente` (di tipo int)
    che vale `0` se la partita è in corso, `1` se la partita è terminata con successo e `2` se la partita è terminata senza successo.
  - Metodo `segna_casella(self, riga:int, colonna:int) -> None`
    che, se lo stato della partita è **partita in corso** e la casella è coperta, contrassegna la casella di riga `riga` e colonna `colonna` come contenente una mina, se non era già stata contrassegnata tale; altrimenti toglie il contrassegno dalla casella stessa.
    Se lo stato della partita non è **partita in corso**, oppure la casella è scoperta, viene lanciata un'eccezione opportuna.
  - Metodo `get_casella_segnata(self, riga:int, colonna:int) -> bool`
    che restituisce `True` se, nel tabellone della configurazione corrente, la casella di riga `riga` e colonna `colonna` è stata contrassegnata dal giocatore come casella contenente una mina, `False` altrimenti.
  - Metodo `scopriCasella(self, riga:int, colonna:int) -> None`
    che, se lo stato della partita è **partita in corso** e la casella non è segnata, scopre la casella di riga `riga` e colonna `colonna`, e di conseguenza
    - se è stata scoperta una casella che non contiene una mina: se tale casella è **sicura** (cioè non ha nessuna casella adiacente contenente mine), vengono scoperte tutte le caselle adiacenti, iterando il procedimento fino a che tutte le caselle sicure hanno tutte le caselle a loro adiacenti scoperte. La partita viene portata in stato di **terminazione con successo** se dopo aver scoperto la casella di riga `riga` e colonna `colonna` (ed eventualmente le caselle adiacenti alle caselle sicure) rimangono da scoprire solo caselle che contengono mine.
    - se è stata scoperta una casella contenente una mina porta la partita in stato di **terminazione senza successo**.

    Al termine, viene opportunamente memorizzato il tabellone così ottenuto in modo che possa essere in seguito restituito dalla proprietà `evoluzione`.
    Se lo stato della partita non è **partita in corso**, oppure la casella è segnata, viene lanciata una eccezione opportuna.

- Metodo `get_mine_adiacenti(self, riga:int, colonna:int) -> int`
  che, se la casella di riga `riga` e colonna `colonna` non contiene mine, restituisce il numero di caselle ad essa adiacenti contenenti mine (0 se la casella è sicura). Se la casella di riga `riga` e colonna `colonna` contiene mine, viene restituito -1.
- Metodo `contiene_mina(self, riga:int, colonna:int) -> bool`
  che restituisce `True` se la casella di riga `riga` e colonna `colonna` contiene una mina, `False` altrimenti.
- Metodo `is_coperta (self, riga:int, colonna:int) -> bool`
  che restituisce `True` se la casella di riga `riga` e colonna `colonna` è coperta nel tabellone della configurazione corrente, `False` altrimenti.
- Proprietà in lettura `evoluzione` (di tipo list[Tabellone])
  che restituisce una lista contenente il tabellone iniziale e i tabelloni ottenuti dopo ogni mossa (che consiste nello scoprire una casella) del giocatore.
- Metodo `__str__(self) -> str`
  che deve restituire una stringa che contiene, in sequenza, le rappresentazioni di tutti i tabelloni ottenuti a seguito del fatto che il giocatore scopre delle caselle, preceduta dall'indicazione della mossa (che consiste nello scoprire una casella) che ha condotto a tale tabellone:

```
Tabellone iniziale:
CCCCCCCCCC
CCCXCCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCXCCCCCCC
CCCCCCCXCC
Tabellone a seguito della mossa di riga 6 e colonna 5:
CCCCCCCCCC
CCCXCCCCCC
CCCCCCCCXC
CCCCXXCCXC
CCX2222CCC
CCC1001XCC
```

Il modulo **gui.py** deve contenere la seguente **classe**:

- `InterfacciaGrafica`

  Classe pubblica che sfrutta la libreria `EzGraphics`

  La richiesta minimale è che venga disegnato il tabellone in modo che alle diverse tipologia di casella (disegnate come quadrati) siano associati diversi colori:

tipologia casella	**Colore nella grafica**
Casella coperta non segnata	Grigio chiaro
Casella coperta segnata	Rosso
Casella scoperta (senza mine) sicura	Bianco
Casella scoperta (senza mine) con **n**>0 mine adiacenti	Bianco con **n** visualizzato
Casella scoperta con mina	Nero

  L'interfaccia con l'utente avviene attraverso mouse e tastiera:

  - il **click con il tasto sinistro** su una casella provoca l'azione di scoprire la casella che non deve essere segnata (se la casella è segnata o è già scoperta non avviene nulla).

  - il **click con il tasto destro** segna la casella (che deve essere coperta) come contenente mine (se non è già segnata) o rimuove tale segno (se è già segnata). Se la casella è scoperta, non avviene nulla.

  - La pressione del tasto **Esc** della tastiera provoca la visualizzazione di tutte le mine e inibisce il proseguimento della partita.

  Contiene (tra gli altri) il seguente metodo:

  - Costruttore

    `__init__(self, p:Partita)`

    che costruisce e visualizza una interfaccia grafica per la partita `p`, visualizzando come punto di partenza il tabellone della configurazione corrente della partita e permettendo l'interazione con l'utente a partire da tale configurazione. Può quindi essere passata come parametro sia una partita appena iniziata, in cui il tabellone ha tutte le caselle coperte, sia una partita in cui sono già state scoperte e/o segnate alcune caselle.