Corso di Programmazione e Algoritmi 2
Modulo di Laboratorio
A.A. 2025/2026

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 scritta 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 opportuno sviluppare anche altre classi (non pubbliche) oltre quelle richieste. Si è infatti liberi di sviluppare (e anzi siete incoraggiati a farlo) classi e/o metodi aggiuntivi, laddove lo si ritenga utile.
- 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 tutte presenti nella specifica. Deve essere vostra cura aggiungerle, dove occorrono, in modo opportuno.
- Dove si rende necessario, vanno implementati anche i metodi **__eq__** ed **__hash__**.
- 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 2025_2026.zip
  l'email deve avere come oggetto
  [python] consegna progetto programmazione 2025/2026
  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 **gioco_del_15.py** deve funzionare in modo autonomo, anche senza il modulo **gui.py** (ed anche in combinazione con il modulo gui.py di un altro progetto ben fatto!), e deve possedere tutte le annotazioni 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* *https://www.ezgraphics.org/*. In questo modulo non è richiesto il type checking.

Testo del progetto:

Sviluppare in **Python** due moduli (**gioco_del_15.py** e **gui.py**) per la rappresentazione ed esecuzione di partite del famoso "gioco del 15". Per la descrizione del gioco si faccia riferimento alla pagina Wikipedia https://en.wikipedia.org/wiki/15_puzzle. Al link https://tictactoefree.com/it/gioco-del-quindici/ è invece disponibile una versione giocabile online del gioco.

Impostazione formale

Il modulo **gioco_del_15.py** deve contenere le seguenti **classi**:

- `_Table`

  Classe che rappresenta un tabellone del gioco.
  Si assume che la riga **1** sia la prima riga dall'alto, e che la colonna **1** sia la prima colonna a sinistra.
  Se un tabellone ha dimensioni *larghezza* e *altezza*, contiene i numeri da 1 a (*larghezza * altezza -1*).
  La classe _Tabellone contiene i seguenti metodi:

  - Costruttore
    `__init__(self, g:Game)`
    che costruisce un tabellone, relativo alla partita `g`, avente `g.altezza` righe e `g.larghezza` colonne. Tutte le celle del tabellone sono vuote al momento dell'inizializzazione.

  - Metodo di classe per costruttore "di copia"
    `copy_table (cls, t:_Table) -> _Table`
    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 `set_box(self, row:int, column:int, value:int)-> None`
    che assegna alla casella di riga `row` e colonna `column` il valore `value` (si tenga presente che il valore 0 corrisponde alla casella vuota). Se la casella in questione non appartiene al tabellone (o se `row` e/o `column` sono <= 0), o se `value` non è ammissibile, viene sollevata un'opportuna eccezione.

  - Metodo `get_box (self, row:int, column:int) -> int`
    che restituisce il valore associato alla casella di riga `row` e colonna `column` (si tenga presente che il valore 0 corrisponde alla casella vuota). Se la casella in questione non appartiene al tabellone (o se `row` e/o `column` sono <= 0), viene sollevata un'opportuna eccezione.

  - Metodo `is_solvable(self) -> bool`
    che restituisce `True` se il tabellone corrente contiene tutti i numeri da 1 a n-1 (dove n è il risultato del prodotto tra il numero di righe e il numero di colonne) ed è possibile arrivare dal tabellone corrente alla **configurazione finale di vittoria** (in cui i numeri sono ordinati da 1 a n-1 a partire dalla prima riga in alto e, per ogni riga dall'alto verso il basso, scorrendo le righe da sinistra verso destra, con la casella vuota ultima casella del tabellone, ovvero la casella più a destra della riga più in basso – si veda il metodo `__str__` per un esempio di configurazione finale di vittoria). Si veda l'**Appendice** per capire come decidere circa la risolvibilità di un gioco.

- Metodo `__str__(self) -> str`
  che deve restituire una stringa che rappresenta il tabellone corrente, contenente gli opportuni numeri da 1 a n-1 (dove n è il risultato del prodotto tra il numero di righe e il numero di colonne), separati da uno spazio e debitamente incolonnati. Ad esempio,

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15
```

  corrisponde al tabellone relativo alla **configurazione finale di vittoria** di una partita con un tabellone avente 4 righe e 4 colonne.

- `Game`

Classe pubblica che rappresenta una partita del gioco.
Anche qui si assume che la riga **1** sia la prima riga dall'alto, e che la colonna **1** sia la prima colonna a sinistra.
Contiene i seguenti metodi/proprietà:

- Costruttore
  `__init__(self, width:int, heigth:int)`
  che costruisce una partita del gioco contraddistinta da un tabellone avente `heigth` righe e `width` colonne, nel quale sono posizionati in modo casuale i numeri interi da 1 a `width * heigth - 1`, in modo che il gioco sia **risolvibile** (si veda l'**Appendice**).
  Lo stato della partita è impostato su **partita in corso**.
- Proprietà in lettura `width` (di tipo `int`)
- Proprietà in lettura `heigth` (di tipo `int`)
- Proprietà in lettura `completed` (di tipo `bool`)
  che vale `False` se la partita è in corso, `True` se la partita è terminata con successo.
- Proprietà in lettura `table` (di tipo `list[list[int]]`) che restituisce una lista di liste di interi, in cui ogni elemento di posizione `[row][column]` contiene l'elemento che si trova correntemente nella casella di riga `row` e colonna `column` (0 denota la casella vuota).
- Metodo `move(self, row:int, column:int) -> None`
  che, se le seguenti condizioni sono verificate:
  - lo stato della partita è **partita in corso**,
  - la casella di riga `row` e colonna `column` è una casella non vuota del tabellone
  - sulla riga `row` o sulla colonna `column` è presente la casella vuota

  effettua una mossa che consiste nello spostare tutte le caselle non vuote comprese dalla casella di riga `row` e colonna `column` (inclusa) alla casella vuota (esclusa) di una posizione verso la casella vuota, in modo che al termine dello spostamento la casella vuota si venga a trovare in posizione di riga `row` e colonna `column`.
  Al termine, viene opportunamente memorizzato il tabellone così ottenuto in modo che possa essere in seguito sfruttato dal metodo `__str__` che stampa tutti i tabelloni della partita.
  Se almeno una delle condizioni non è verificata, viene lanciata un'opportuna eccezione.

- Metodo `undo(self) -> None`
  che annulla l'ultima mossa (non ancora annullata) effettuata.
- Metodo `redo(self) -> None`
  che ripristina l'ultima mossa annullata (non ancora ripristinata).
  SI tenga conto del fatto che appena si esegue una nuova mossa, tutte le mosse annullate e non ancora ripristinate non sono più ripristinabili.
- 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 ha effettuato delle mosse, preceduta dall'indicazione della mossa che ha condotto a tale tabellone. Ad esempio, in un semplice gioco che è stato risolto in un'unica mossa:

```
Tabellone iniziale:
 1  2  3  4
 5  6  7
 9 10 11  8
13 14 15 12
Tabellone a seguito della mossa di riga 4 e colonna 4:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15
```

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

- `GUI`
  Classe pubblica che sfrutta la libreria `EzGraphics`
  La richiesta minimale è che venga disegnato il tabellone in modo che le caselle siano disegnate come quadrati contenenti il numero corrispondente.

  L'interfaccia con l'utente avviene attraverso mouse e tastiera:
  - il **click con il tasto sinistro** su una casella provoca l'azione di invocare una mossa su tale casella (se la casella non rispetta le condizioni per poter effettuare una mossa, non avviene nulla).
  - La pressione del tasto **CTRL+Z** della tastiera provoca l'annullamento dell'ultima mossa effettuata (se tale cosa è possibile).
  - La pressione del tasto **CTRL+Y** della tastiera provoca il ripristino dell'ultima mossa annullata e non ancora ripristinata (se tale cosa è possibile)

  Contiene (tra gli altri) il seguente metodo:

  - Costruttore
    `__init__(self, g:Game)`
    che costruisce e visualizza una interfaccia grafica per la partita `g`, visualizzando come punto di partenza il tabellone della configurazione corrente della partita e permettendo l'interazione con l'utente a partire da tale configurazione.

Appendice - Risolvibilità del gioco

Data una configurazione **c**, il suo **numero di inversioni *inv(c)*** è dato dal numero di coppie **(a,b)** con **a>b** tali che **a <u>compare prima di</u> b** nella lettura del tabellone della configurazione riga per riga (da sinistra a destra, dall'alto in basso). Inoltre **l'indicatore** di riga della casella vuota è uguale a ***ind(c)*=n-i+1**, dove **i** è l'indice di riga della casella vuota (*ind(c) = 1* se la casella vuota è sull'ultima riga, *ind(c) = 2* se è sulla penultima e così via).

Si dimostra che nel gioco del 15 "esteso" con **n** righe e **m** colonne, la configurazione finale di vittoria ***c**** è raggiungibile da una configurazione **c** <u>se e solo se</u> la **parità** della configurazione di vittoria ***c**** è la stessa **parità** della configurazione **c**. La parità di una configurazione **c** è un valore booleano (0 o 1) così calcolato:

- Se **m** è pari, la parità è data da ***(inv(c)+ ind(c)) mod 2***
  Si noti che la parità della configurazione di vittoria **c*** è **1** in quanto ***inv(c*)=0*** e ***ind(c*)=1***
- Se **m** è dispari, la parità è data da ***inv(c) mod 2***
  Si noti che la parità della configurazione di vittoria **c*** è **0** in quanto ***inv(c*)=0***

Dunque, per determinare se la configurazione di vittoria è raggiungibile da una configurazione **c**, la parità di **c** deve <u>essere</u> **1** se m è pari, e **0** se m è dispari.