Programmazione Orientata agli Oggetti

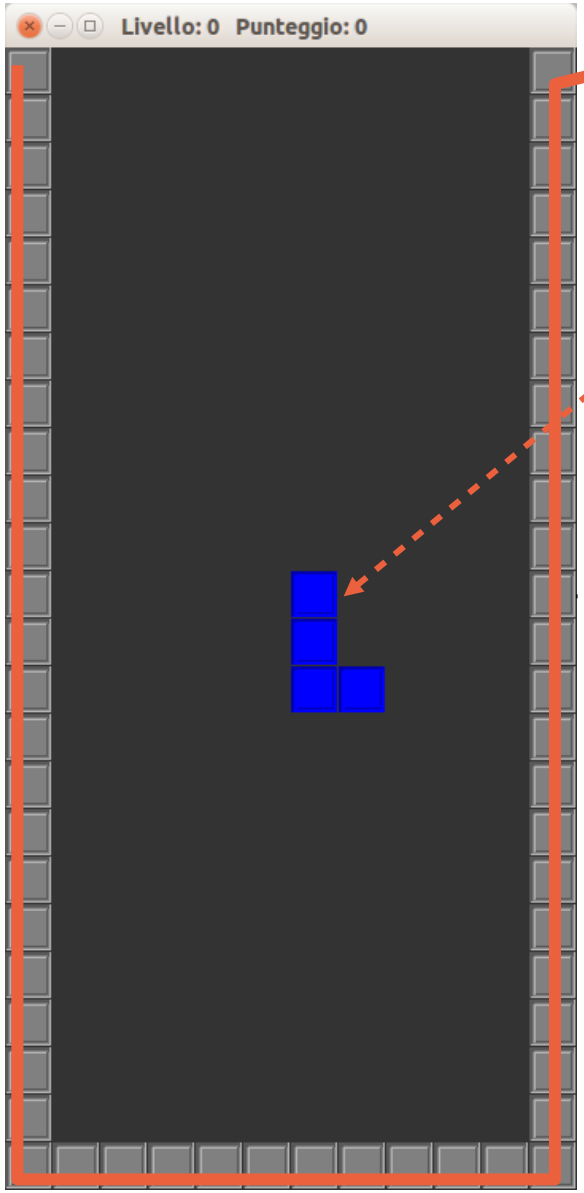
Collezioni Insiemi: Esercitazione Tetris

A cura di Valerio Cetorelli e Federico Piai

Sommario

- Tetris: Le regole del gioco
  - I tetramini
- Diagrammi delle Classi
  - Classi di “Alto Livello”
  - Classe **`Pozzo`** e conoscenti
- Esercizio 1
  - Codificare il criterio di euivalenza in **`Cella`**, **`Posizione`**
  - Introdurre nelle due classi un criterio di ordinamento *naturale*
- Esercizio 2, 3, 4
  - Unit-testing
  - Completare i metodi di interrogazione del pozzo utilizzando
    - **`Set/SortedSet/NavigableSet`**
    - **`TreeSet`**
    - **`Comparable/Comparator`**
- Riflessione finale

https://it.wikipedia.org/wiki/Tetris

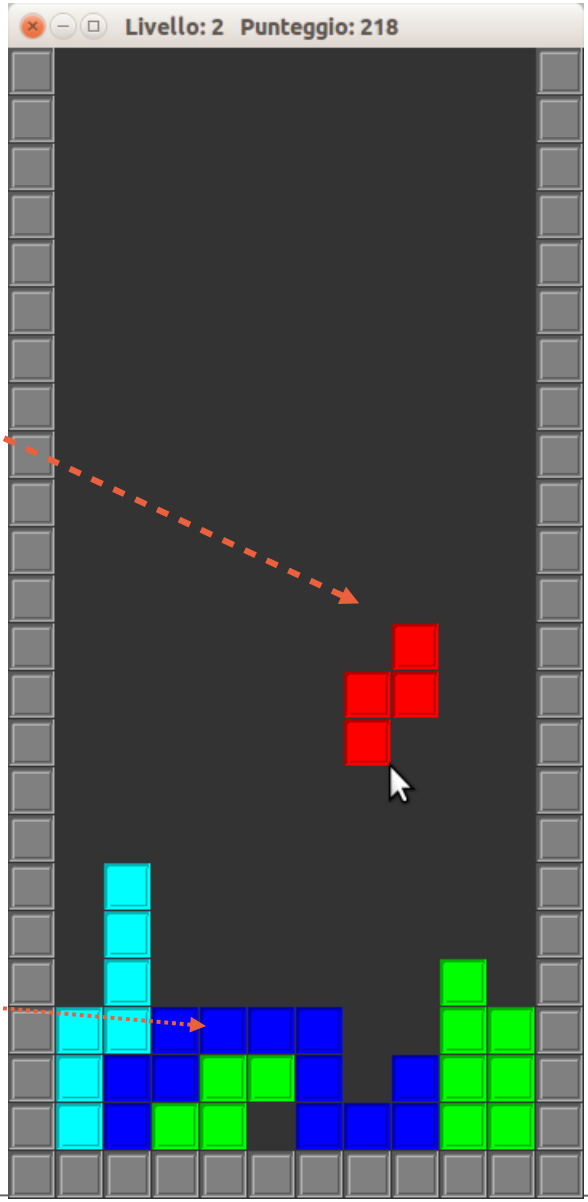


Bordo del Pozzo

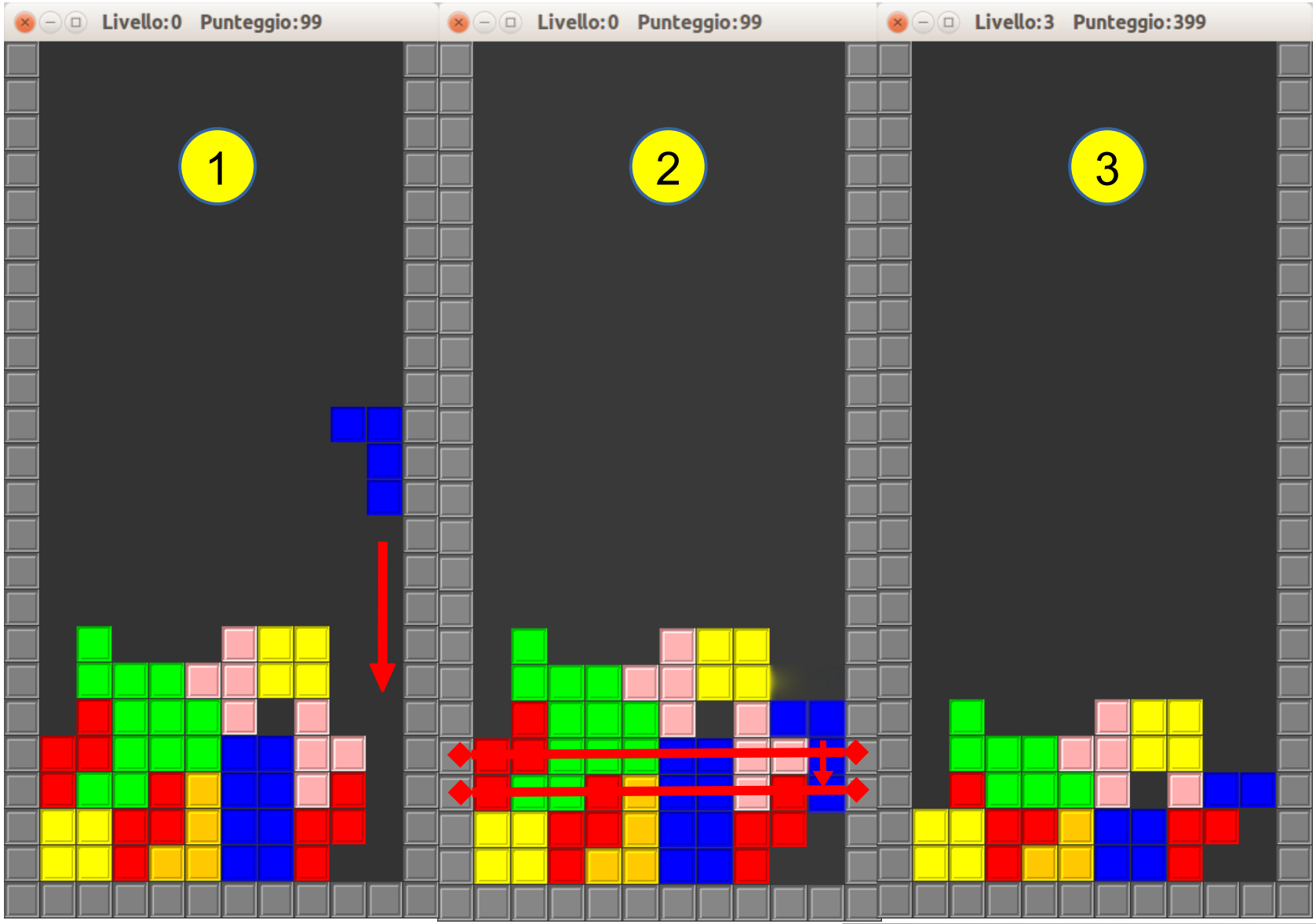
Tetramino corrente: 4 *blocchi*

Blocchi accumulati sul fondo del pozzo

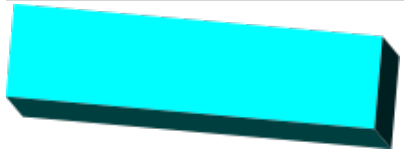
Pozzo



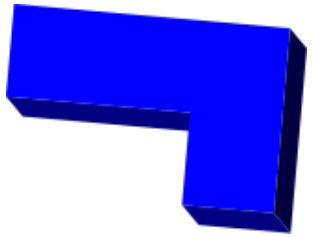
Completamento di Linee



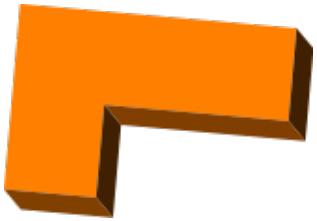
I 7 Tetramini



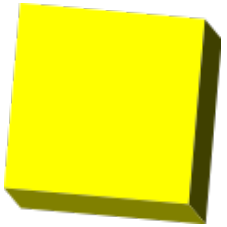
- **I** (anche detto "barra", "dritto" o "lungo"): quattro quadratini allineati



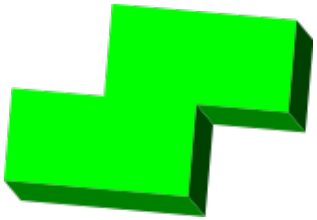
- **J** (anche detto "L rovesciata"): una riga di tre quadratini più un quadratino aggiunto sotto a destra



- **L**: una riga di tre quadratini più un quadratino aggiunto sotto a sinistra. Questo tetramino non è altro che il precedente riflesso, ma non si può passare dall'uno all'altro solo con rotazioni in due dimensioni, per cui esso è chirale

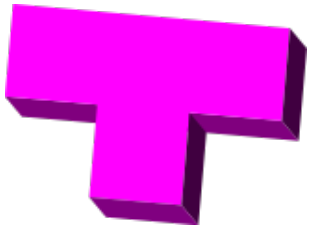


- **O** (anche detto "quadrato"): quattro quadratini in un quadrato



- **S** (anche detto "N" o "serpente"): due domino sovrapposti, con il superiore spostato a destra

- **Z** (anche detto "N rovesciata"): due domino sovrapposti, con il superiore spostato a sinistra. Come nel caso dei pezzi J e L, questi due tetramini sono chirali in due dimensioni e tra di loro speculari

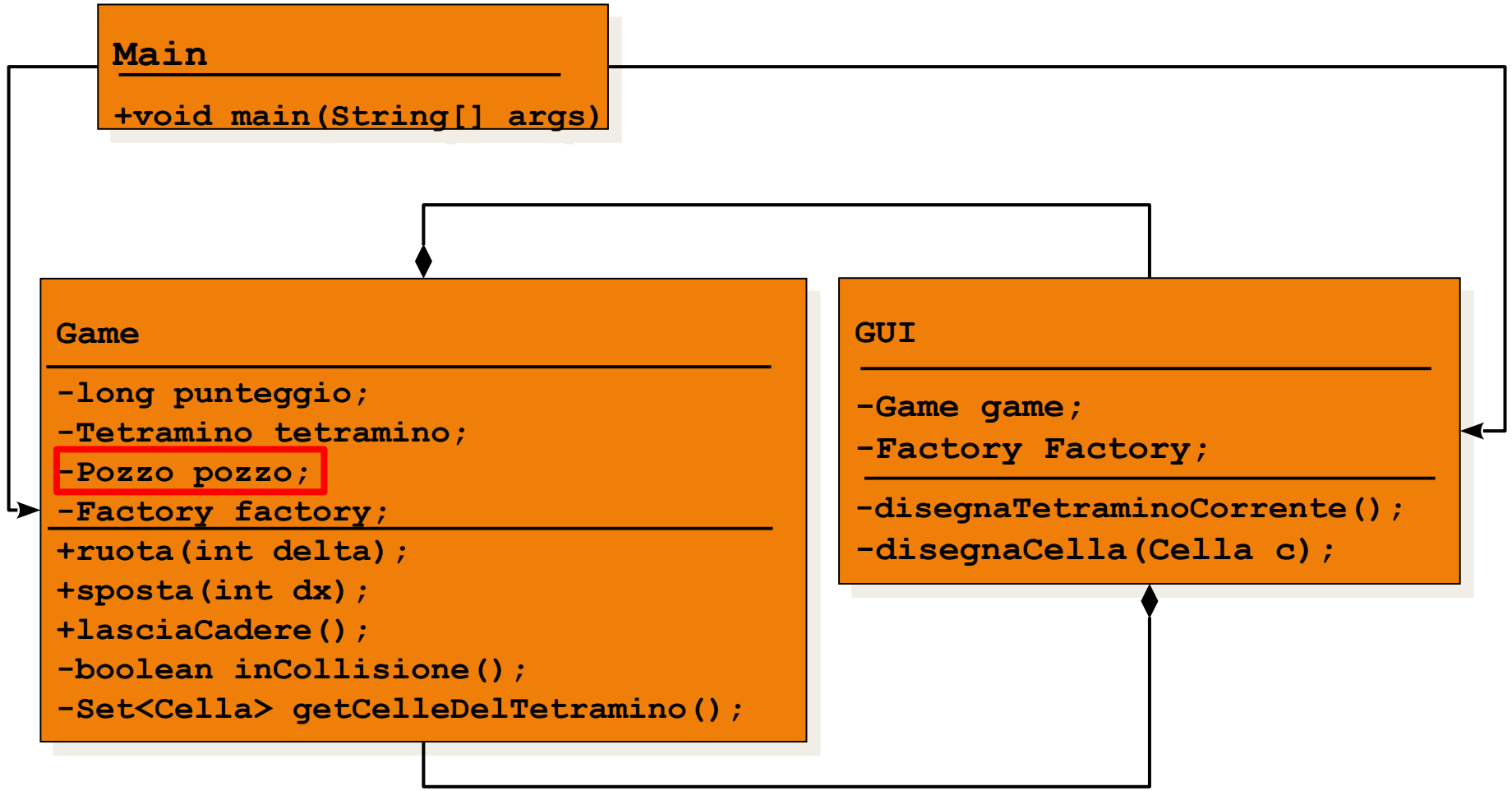


- **T**: tre quadratini in fila più uno aggiunto sotto, al centro

Fonte: https://it.wikipedia.org/wiki/Tetramino

Tetris: Diagramma delle Classi

- Classi di “alto livello” del package **`tetris`**



Il Pozzo

- Una collezione di celle, delle quali una parte (quelle di colore grigio scuro) formano il bordo
- I tetramini sono compositi di 4 blocchi
- Il tetramino possiede una posizione
- Ciascun blocco possiede una posizione relativa al tetramino di appartenenza
- Una volta depositati, i blocchi diventano <u>celle</u> del pozzo

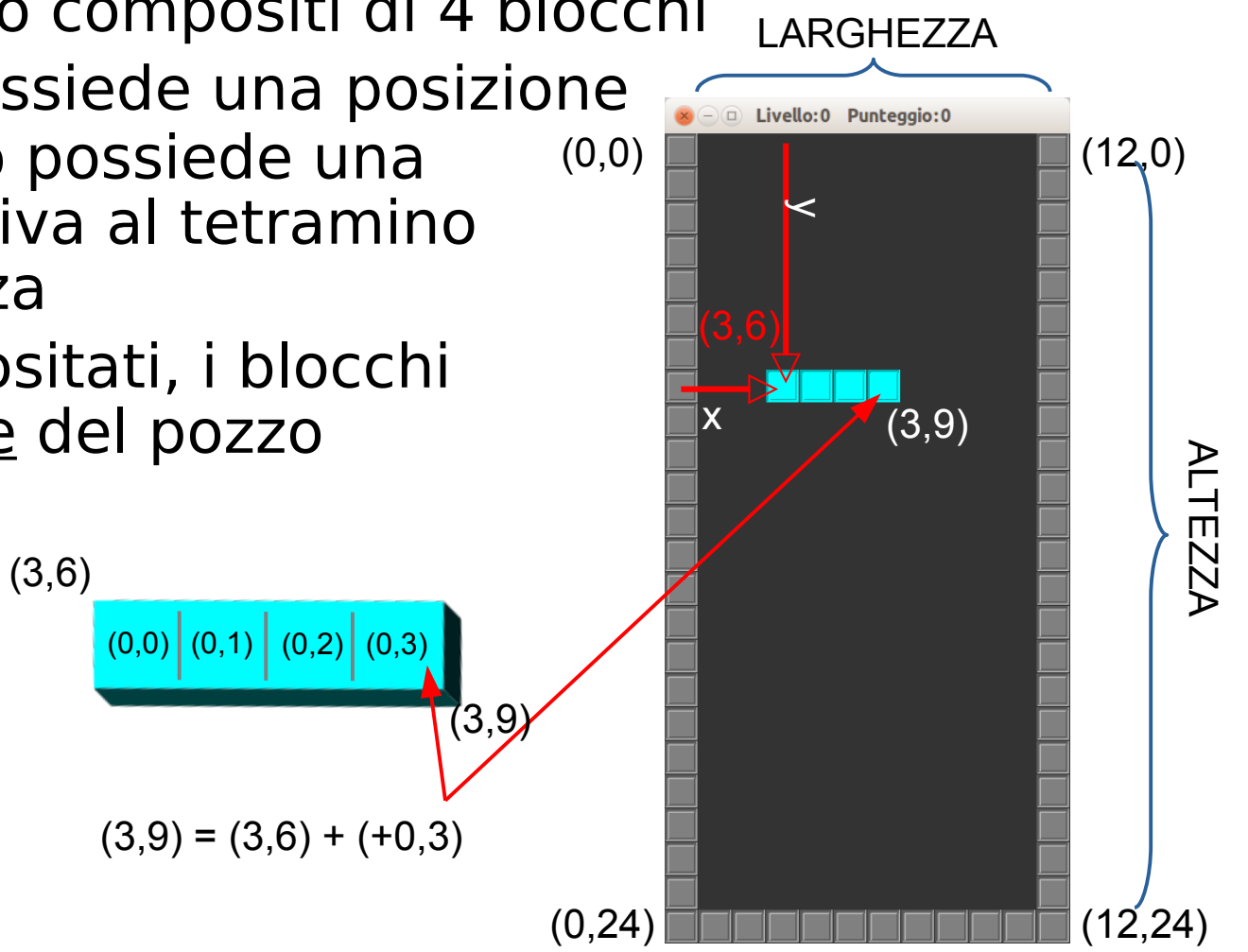
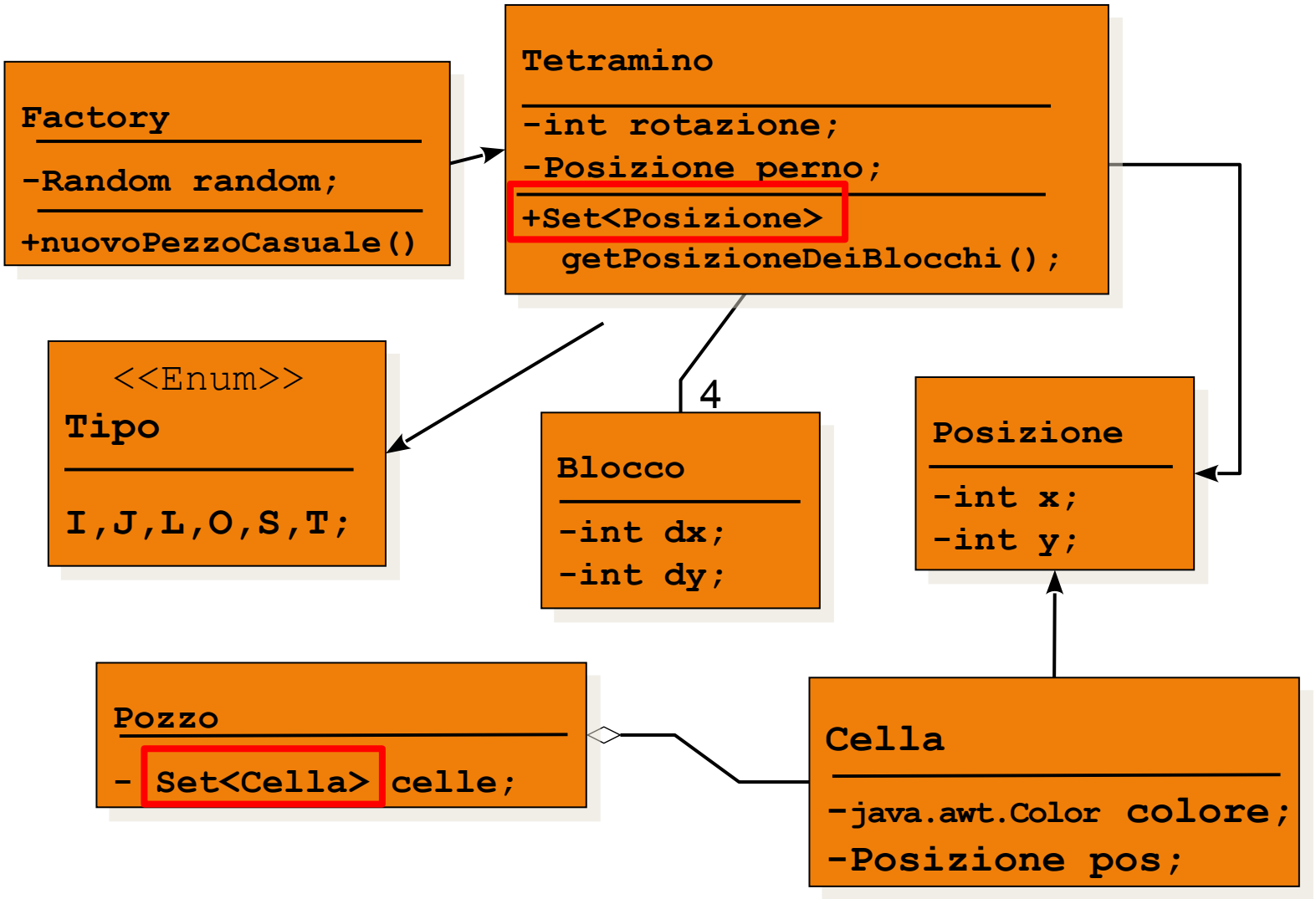


Diagramma delle Classi: Package `tetris.tetramino/pozzo`



Pozzo.java

- Contiene la logica di gestione delle celle del pozzo
- Consiste, approssimativamente, di tre parti

1) **Logica di creazione del pozzo e dei suoi bordi**
2) Il metodo che realizza la logica per aggiungere celle quando un tetramino tocca il fondo del pozzo

```
public int aggiungiCelleErimuoviRigheCompletate(Set<Cella> celle)
```

3) Altri metodi di interrogazione

Pozzo.java: Logica di Creazione

```
public class Pozzo {
...
  final private NavigableSet<Cella> celle;
  final private int larghezza;
  final private int altezza;

  public Pozzo() {
    this(LARGHEZZA,ALTEZZA);
  }

  public Pozzo(int l, int h) {
    this.celle = null; /*DA COMPLETARE*/
    this.larghezza = l;
    this.altezza = h;
    this.addBordo(l, h);
  }
...
}
```

Classe `Cella`
Criterio di Ordinamento >>

Per conservare le celle un insieme **ordinato** e *navigabile*

Esercizio 1

a) Le classi **`Posizione`** e **`Cella`** sono usate all'interno di insiemi (es. metodo **`getPosizioneDeiBlocchi()`** nella classe **`Tetramino`**): stabilire un criterio di equivalenza e di ordinamento per gli oggetti di queste classi

b) Scrivere dei test di unità per verificare il comportamento di questi criteri

c) Implementare i criteri di equivalenza decisi, aggiungendo alle due classi opportuni metodi **`equals()`**/ **`hashCode()`**

d) Modificare le classi affinché implementino l'interfaccia **`Comparable`**, dotandoli dei metodi **`compareTo()`**

e) Verificare il successo dei test scritti al punto b)

f) Completare il costruttore della classe **`Pozzo`**

Esercizio 2

```
/**
 *
 * @param inputCelle - un insieme di celle
 * @return l'insieme ({@link NavigableSet}) delle ordinate (y)
 *         delle posizioni delle celle passate. L'insieme è
 *         ordinato crescente.
 * @see {@link Posizione},  {@link Cella}
 */
public NavigableSet<Integer> getInsiemeOrdinateY(Set<Cella> inputCelle) {
  /* DA COMPLETARE (esercizio 2) */
  return null;
}
```

- Il metodo **`getInsiemeOrdinateY()`** è il primo dei metodi di interrogazione a supporto del metodo **`aggiungiCelleErimuoviRigheCompletate()`**
  a) Scrivere una batteria di test di unità *minimali* per verificare il comportamento di questo metodo
  b) Implementare il corpo del metodo **`getInsiemeOrdinateY()`**
  c) Verificare il successo dei test scritti al punto a)

Game.java: Invocazione del Metodo aggiungiCelleErimuoviRigheCompletate()

```
/**
 * Ferma la caduta del pezzo nel pozzo decomponendo
 * i suoi {@link Blocco} in {@link Cella} del {@link Pozzo}
 *
 * @return il numero di righe complete eliminate
 */
private int fermaCaduta() {
  final Set<Cella> celle = getCelleDelTetramino();
  return this.pozzo.aggiungiCelleErimuoviRigheCompletate(celle);
}
```

- La classe **Game** invoca il metodo **fermaCaduta()** non appena il tetramino corrente tocca il fondo del pozzo per fermarne la caduta; quindi sceglie (casualmente) il prossimo tetramino
- Mediante il metodo **getCelleDelTetramino()**, si ottengono le celle che compongono il tetramino corrente da aggiungere al pozzo

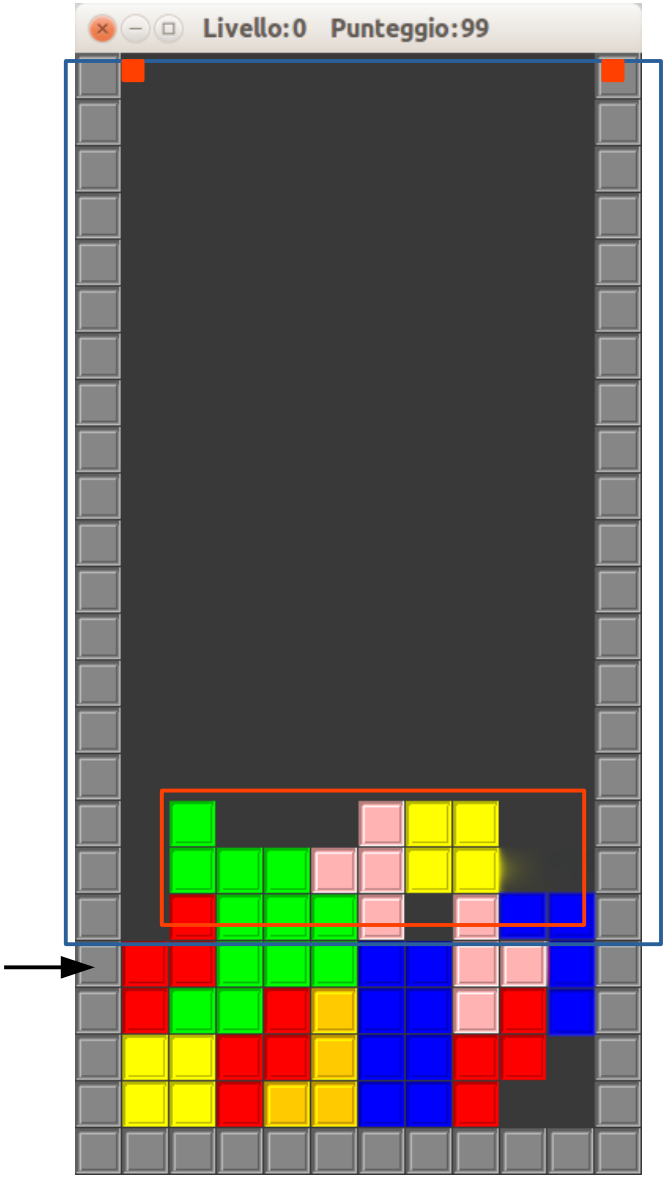
Esercizio 3

- Nella classe **`Pozzo`**, il metodo **`isCompleta()`** è un metodo di interrogazione a supporto del metodo **`aggiungiCelleErimuoviRigheCompletate()`**
  a) Scrivere una batteria di test di unità *minimali* per verificare il comportamento dal metodo **`isCompleta()`**
  b) Scrivere una batteria di test di unità *minimali* per verificare il comportamento del metodo **`getCelleDellaRigaSenzaBordo()`** della classe **`Pozzo`**.
     - Creare allo scopo dei pozzi *minimali* utilizzando il costruttore **`Pozzo(int l, int h)`** che permette di specificare le dimensioni del pozzo creato
  c) Implementare, senza utilizzare iterazioni di alcun genere, il metodo **`getCelleDellaRigaSenzaBordo()`**
  d) Verificare il successo dei test scritti ai punti a) e b)

Esercizio 4

- Sempre nella classe **`Pozzo`**, il metodo **`rimuoviRigaScendendoCelleSopra()`** è un metodo di interrogazione a supporto del metodo **`aggiungiCelleErimuoviRigheCompletate()`**
  a) Scrivere una batteria di test di unità *minimali* per verificare il comportamento atteso dal metodo **`rimuoviRigaScendendoCelleSopra()`**
  b) Scrivere una batteria di test di unità *minimali* per verificare il comportamento atteso dal metodo **`getCelleSopraRigaYdecrescente()`**
     - ✔ N.B. il risultato del metodo **`getCelleSopraRigaYdecrescente()`** non include le celle del bordo nel risultato
  c) Implementare, senza utilizzare iterazioni di alcun genere, il metodo **`getCelleSopraRigaYdecrescente()`** (vedi suggerimenti nella prossima slide >>)
  d) Verificare il successo dei test scritti ai punti a) e b)

Suggerimenti esercizio 4



- Per l'implementazione del metodo **`getCelleSopraRigaYdecrescente():`**
  1) Utilizzando i metodi di **`NavigableSet:`** prendere il sottoinsieme (evidenziato il blu) delle celle *minori* della prima cella della riga
  2) Aggiungere le celle a un secondo **`NavigableSet`**, che definisce un ordinamento diverso: prima sulla base delle *colonne* ed a parità delle colonne sulla base delle *righe.* Da questo set prendere il sottoinsieme compreso tra i due punti rossi
  3) Le celle risultanti vanno aggiunte ad un set ordinato in maniera *descrescente*

Riflessioni Finale

- Confrontare il tempo che si spende facendo debugging nei due casi:
  - ✔ Mediante unit-testing con test minimali
  - ✗ Svolgendo intere partite a Tetris solo per attivare il metodo che si vuole debuggare