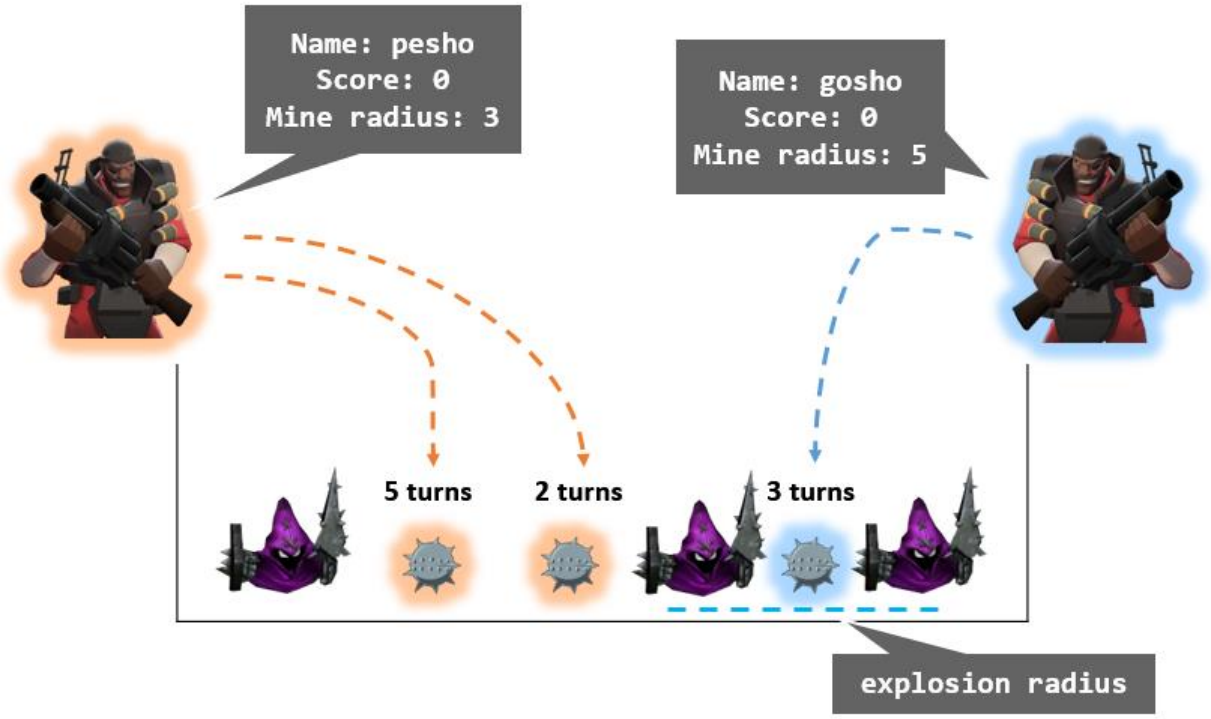
Pit Fortress – Data Structures Retake Exam

Pit Fortress is an interactive game. The rules are as follows:

- There is a fighting pit (a hole in the ground) where minions are put. **Minions** have a unique **id**, **position** in the pit (represented as a X-coordinate) and **health**.
- Players are outside the pit and have a unique **name**, **score** and **mine radius**. Each player can shoot mines down the pit.
- Mines have an **Id**, **timeout** and **explosion radius**. When the timeout expires (e.g. 3 turns), the mine explodes and all **minions in the explosion radius are damaged**. After exploding the mine is deleted from the game.
    - If a minion is killed in the explosion (its health falls to **0 or less**), it is **removed from the pit** and the player whose mine killed it **gets +1 score**.



Your task is to design a data structure in C# or Java that supports the below-listed functionality in a **fast and efficient way**.

- **AddPlayer(name, mineRadius)** – adds a new player to the game with the given name and mine radius (his mines will damage minions in that radius).
    - The player name is **unique**, there cannot be 2 players with the same name, in case of a duplicate name throws an **Argument Exception**.
    - The mineRadius must always be 0 or positive, in case of a negative mineRadius throws an **Argument Exception**.
    - The player has 0 score by default.

- **AddMinion(xCoordinate) –** adds a new minion with the **next id** at **position** {X-coordinate}.
    - The xCoordinate must always be a integer between **[0…1 000 000].** In case of an invalid coordinate throws an **Argument Exception**.
    - It is possible to have multiple minions on the same coordinate.
    - Minions will always start at **100 Health**.

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

- The minion **Id**s are **unique -** they start from 1 and grow with 1 for every new minion (ex. first minion will have Id = 1, second - Id = 2 and so on).

- **SetMine(playerName, X, delay, damage) –** The given player shoots a mine at position X-coordinate. After **{delay}** turns, all minions in the range **[X – explosion radius, X + explosion radius]** receive **{damage}** damage.
  - If a minion falls at **0 or less** health as a result to the damage, it is removed from the game. The player that shot the mine receives **+1 score**.
  - If a player with the given name does not exist, throws an **Argument Exception**.
  - The **xCoordinate** must always be a integer between **[0…1 000 000].** In case of an invalid coordinate throws an **Argument Exception**.
  - The **delay** must be an integer between **[1…10 000]**. In case of an invalid delay throws an **Argument Exception**.
  - The **damage** must always be an integer between **[0…100]**. In case of an invalid damage, throws an **Argument Exception**.
  - The mine **Id**s are **unique** - they start from 1 and grow with 1 for every new mine (ex. first mine will have Id = 1, second - Id = 2 and so on).
- **ReportMinions() –** returns all minions in ascending order, sorted first by X coordinate and then by Id.
- **Top3PlayersByScore() –** returns the 3 players with highest score in descending order (the first one is the one with most points). If multiple players with the same score exist order them by name in descending order as a second criteria. **Example** if Pesho and Gosho both have the same score, Pesho would be before Gosho because in descending order the name "Pesho" is before "Gosho".
  - If there are less than 3 players, throws an **Argument Exception**.
- **Min3PlayersByScore() –** returns the 3 players with least score in ascending order (the first one is the one with least points). If multiple players with the same score exist order them by name in ascending order as a second criteria. **Example** if Pesho and Gosho both have the same score, Gosho would be before Pesho because in ascending order the name "Gosho" is before "Pesho".
  - If there are less than 3 players, throws an **Argument Exception**.
- **GetMines()** – returns all mines ordered first by delay in ascending order then by Id in ascending order.
- **Skip()** – advances the game with 1 turn.
  - All mines' delay is reduced by one every time this command is called.
  - If any mine's delay reaches 0 as a result of this command, it explodes and deals its damage to minions in its radius.
  - If multiple mines would explode on the same turn, they explode in ascending order based on their Ids (the one with the lowest Id would explode first). The results of the explosion should be resolved before the next mine explodes.

Input and Output

You are given a **Visual Studio C# project skeleton** (unfinished project) / **IntelliJ Java project skeleton** (unfinished project) holding the interface for the **PitFortressCollection**, the unfinished class **PitFortressCollection**, interfaces for all classes that the structure will operate with (Player,Minion,Mine) and **tests** covering the **functionality** and **performance** of the collection.

Your task is to **finish this class** to make the tests run correctly.

- You are **not allowed to change the tests**.
- You are **not allowed to change the interfaces**.

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us:

Constraints

- All names:
  - Consist of **Latin letters and digits**.
  - Have length in the range **[1...100]**.
- All string matching operations are **case-sensitive**.
- The **xCoordinates** will be integer numbers between **[2^{-32}...2^{32}].**
- **Damage** will be a valid integer between **[2^{-32}...2^{32}].**
- **Delay** will be a valid integer between **[2^{-32}...2^{32}].**
- The **mineRadius** will be a valid integer between **[2^{-32}...2^{32}].**

Submissions

Submit an archive (.zip) of the source code + external libraries.

Scoring

Each implemented method brings you a specific amount of points, some of the points are awarded for correct behavior, others for performance. Performance tests are directly dependent on correctness, if a method does not work correctly you will not receive points for performance. You need to cover all tests in a given group in order to receive points. Bellow is a breakdown of all points by methods:

Method	Correct Behaviour	Performance	Total
Add Player	4	8	12
Add Minion	4	8	12
Set Mine	6	12	18
PlayTurn	8	16	24
Report Minions	4	12	16
Top 3 Players By Score	3	8	11
Min 3 Players By Score	3	8	11
Get Mines	4	12	16
Overall:	36	84	120

SOFTWARE UNIVERSITY FOUNDATION
© Software University Foundation (softuni.org). This work is licensed under the CC-BY-NC-SA license.
Follow us: