Feladatok

1 Egy futsal csapat egy kapusból és négy mezőnyjátékosból áll (egy csatár, két szélső és egy védő). A feladatunk, hogy egy olyan programot készítsünk, amely segítségével az elérhető játékosokból helyes felépítésű csapatot állíthatunk elő.

Minden játékosról ismert a neve és a pozíciója, amely a fent felsorolt értékek egyike lehet. Egy csapatba pontosan 5 játékos kell, a fentebb definiált módon. A program indításkor generáljon játékosokat, és a felhasználónak legyen lehetősége csapathoz rendelni őket.

Készítsük el a játékosokat reprezetáló `Player` osztályt az alábbi tagokkal.

- string name: a játékos neve
- Position pos: a játékos pozíciója (`Goalkeeper, Forward, Winger, Defender`)

Az adattagok beállítása a konstruktoron keresztül lehetséges. Definiáljuk felül az osztály **ToString** metódusát, amely a játékos nevét és pozícióját tartalmazó karakterláncot adja vissza.

A csapatot a `Team` osztály reprezentálja. Valósítsuk meg az alábbi tagokat az osztályban.

- Player[ ] players: a csapatban levő játékosok tömbje
- int NumberOfPlayers: a csapatban levő játékosok száma
- bool IsFull: a játékosok száma elérte az ötöt?
- bool IsIncluded(Player): eldönti, hogy a parameter szerepel-e már a csapatban
- bool IsAvailable(Player): eldönti, hogy a parameter pozíciója szabad-e
- void Include(Player): játékos hozzáadása a csapathoz

A főprogramban készítsük el az alábbi statikus metódust.

- Player[] RandomPlayers(int): adott mennyiségű játékos generálása

A **Main**-ben a véletlenszerűen generált játékosok közül választhat a felhasználó, és a csapatba rendelheti őket.

2 Készítsünk bölényvadász játékot. A bölénycsorda N bölényből áll, amelyek egy $M \times M$-es játéktéren a bal felső sarokból, a $(0, 0)$ koordinátáról indulnak, és a játéktér jobb alsó sarkának irányába haladnak. Az N és M értékeket a felhasználó adhatja meg a játék kezdetén.

A játék körökre osztott. Minden körben minden bölény véletlenszerűen lép egyet jobbra $(x + 1)$, lefelé $(y + 1)$ vagy átlósan jobbra lefelé $(x + 1, y + 1)$, de mindig a pálya határain belül maradva.

A felhasználó minden körben lövést ad le egy kiválasztott koordinátára, így eltalálva az ott lévő bölények mindegyikét. Ha eltalált egy bölényt, az kiesett a játékból. A bölények akkor győznek, ha bármelyik elér a célba, a felhasználó pedig akkor, ha sikerül minden bölényt kiiktatnia, mielőtt bármelyik a célba érne.

A játékteret a `Field` osztály reprezentálja.

- Tároljuk el a játéktér méretét (M) egy mezőben.
- A mező értékét a konstruktorban állítsuk be a paraméterként átadott értéknek megfelelően.
- Készítsünk egy `TargetX` és egy `TargetY` tulajdonságot, amelyek lekérdezésekor a játéktér cél koordinátáit (a pálya jobb alsó sarkának koordinátáit) kapjuk vissza.
- Definiáljunk egy **AllowedPosition** nevű metódust, amelynek két egész értéket fogad paraméterül, majd visszaadja, hogy az ezek által leírt koordinánta a játéktér része-e. (Vagyis ha ide lépne egy bölény, akkor még a játéktéren belül maradna-e.)
- A **Show()** nevű metódus meghívásakor rajzoljuk ki a játéktér körvonalát a képernyőre (például | és – karakterek segítségével).

Készítsük el a `Buffalo` osztályt a bölények reprezentálására.

- Tároljuk el a bölény aktuális pozícióját (x és y koordinátáit) egy-egy változóban. A bölény állapotát (aktív/nem aktív) egy logikai típusú mezőben tároljuk.
- Készítsünk egy `X` és egy `Y` tulajdonságot, amelyek lekérdezésekor a bölény aktuális koordinátáit kapjuk vissza.
- A **Move** metódus egyetlen paramétere egy `Field` típusú példány. A metódus meghívásakor valósítsuk meg a bölény egy lépését a fenti szabályok szerint, de csak olyan mezőre léphet, amelyre a paraméterként kapott példány **AllowedPosition** igaz értékkel tér vissza.
- A **Deactivate** metódus meghívásakor a bölény állapota nem aktív (hamis) értéket vesz fel.
- A **Show** metódus hívásakor a bölény korrdinátájának megfelelő helyre írjunk ki egy `B` karaktert zöld színnel, ha a bölény aktív, vagy piros színnel, ha nem aktív.

Készítsük el a `Game` osztályt az alábbi tagokkal.

- Legyen egy `Field` típusú mezője, amely a játékteret leíró példányt tárolja. A bölényeket tároljuk egy tömbben vagy listában.
- Az `IsOver` tulajdonság a játék aktuális állapotát adja meg: igaz értékkel tér vissza, ha a játék véget ért, és hamis értékkel, ha még folyamatban van (lásd a fenti szabályokat).
- A konstruktor a játéktér méretét és a bölények számát várja paraméterül, majd ezeknek megfelelően létrehozza a játékteret és a szükséges számú bölényt.
- A privát **VisualizeElements** metódus törli a képernyőt, majd kirajzolja a játékteret és a bölényeket a képernyőre a **Show** metódusok meghívásával.
- A privát **Shoot** metódus egy x és egy y koordinátát kér a felhasználótól, majd minden olyan bölényt deaktivál a megfelelő metódus meghívásával, amely a felhasználó által megadott pozícióban tartózkodik.
- A **Run** metódus meghívásakor rajzoljuk ki a játék állapotát a **VisualizeElements** meghívásával, majd hívjuk meg a **Shoot** metódust. Ismételjük ezeket a lépéseket az `IsOver` aktuális értékétől függően.

A főprogramban készítsünk egy példányt a `Game` osztályból, majd a **Run** metódus hívásával indítsuk el a játékot.