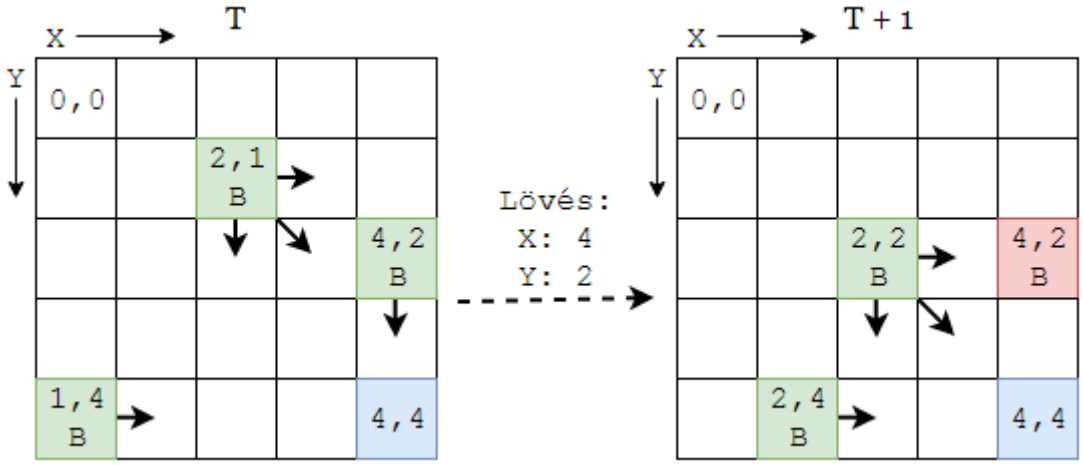
2. ZH – Gyakorló Feladat – Bölény vadászat

Készítsen konzolos bölényvadász játékot!

A játék egy **W*H** méretű pályából áll, ahol $\mathbf{B_S}$ darab bölény található. A bölények vagy véletlenszerű, vagy előre meghatározott helyen kezdenek a játéktéren, és a pálya jobb alsó sarkába, a célba szeretnének eljutni. A játék körökre osztott, minden körben a bölények lépnek egyet véletlenszerűen a cél felé, vagyis léphet jobbra (X + 1), lefele (Y + 1) vagy átlósan lefelé (X + 1, Y + 1), úgy, hogy a pálya határain belül maradnak (a pálya jobb szélén csak lefele, a pálya alján csak jobbra léphetnek). A felhasználó feladata a bölények lelövése mielőtt legalább egy bölény eljutna a célba. A felhasználó minden körben lead egy lövést egy X, Y koordinátára, ahol az összes ott található bölény meghal. A játék véget ér és a bölények győznek, ha bármelyik elér a célba; szintén véget ér a játék, de a felhasználó győz, ha minden bölény meghal, mielőtt eljutnak a célba.



A pálya állapota a T. és a T+1. időpillanatban, jelölve a bölények lehetséges lépési irányát.

A játék inicializációs paramétereit az exe mellett elhelyezett `bullhunter.init` fájlból kell beolvasni. A fájl tartalmazza a bölénycsorda ($1 \leq \mathbf{B_S} \leq 100$) és a pálya ($5 \leq \mathbf{W},\mathbf{H} \leq 15$) méretét továbbá a bölények kezdőpozíciójára vonatkozó információ(ka)t a következő formában:

- Ha a fájl 3. sorában az `RND` szó szerepel, akkor a bölényeket véletlenszerűen kell szétosztani a pályán.
- Ha a fájl 3. sorában egy szám van, akkor be kell olvasni a számnak megfelelő számú további sort, ami megadja `{X,Y}` koordináta formában, hogy az adott bölény hol helyezkedik el a pályán.

A játék köreiben meg kell jeleníteni:

- a pályát, rajta a halott bölényekkel (az élőket nem),
- a célhoz legközelebbi élő bölény céltól vett távolságát,
- a felhasználó utolsó lövése hány bölényt ölt meg,
- továbbá a halott és az élő bölények számát.

A játékot a következő lépéssorozat írja le:

1. A játék inicializálása.
2. A játék körei, amíg a bölények nem halnak meg, vagy amíg az egyik nem ér el a célba.
    a. A pálya és a további információk megjelenítése.
    b. A lövés X, Y koordinátájának bekérése.
    c. A bölények léptetése.
3. A játék vége, a győztes és a pálya megjelenítése a halott és az élő bölényekkel.

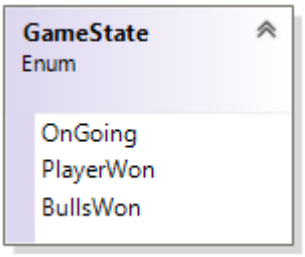
Továbbá az alkalmazás log-olja a működésének fontosabb lépéseit, amit a játék végeztével egy fájlba ír. A log üzeneteknél lehessen szinteket meghatározni és tárolja az üzeneten kívül a létrehozás dátumát is.

A FELADAT MEGOLDÁSA SORÁN A KÖVETKEZŐ STRUKTÚRÁT VALÓSÍTSA MEG

Az alábbi felépítést követve készítsen osztályokat, és valósítsa meg a leírás alapján az alkalmazás működését. Ügyeljen rá, hogy betartsa az objektum-orientált programozásban használatos egységbezásári és adatrejtési elveket. Ahol szükséges az adatmezőkhöz hozzon létre tulajdonságokat, ha elegendő csak olvashatót, illetve a példányhoz nem kötődő metódusokat, és adatmezőket tegye statikussá.

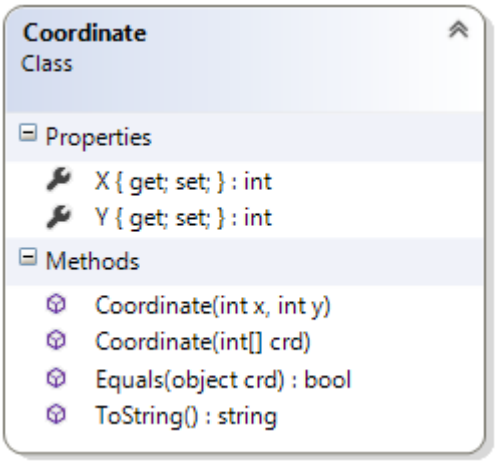
`enum GameState`

A típusban a játék lehetséges állapotai jelennek meg. A játék `OnGoing`, ha elkezdődött és még nincsen nyertese, `PlayerWon`, ha a játékot a felhasználó nyerte meg és `BullsWon`, ha a játékot a bölények nyerték meg.



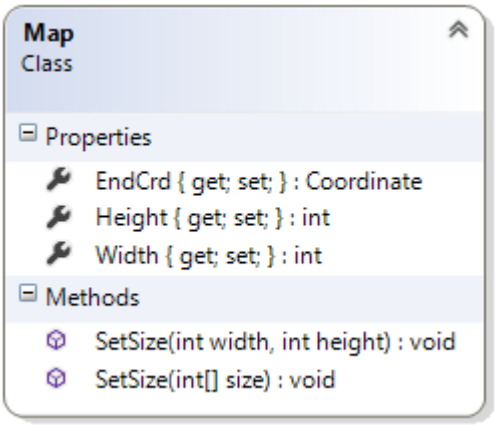
`class Coordinate`

Az osztály példánya egy `X` és egy `Y` tulajdonságban eltárolja a koordinátát, amit a konstruktoron keresztül kétféle képpen lehet megadni. Egyik esetben egy `x` és egy `y` értéket, másik esetben egy számtömböt kap bemenetként. Ezeken kívül az osztály kettő metódussal rendelkezik, az egyik felüldefiniálja a `ToString()` metódust, hogy formázottan lehessen megjeleníteni az osztály tartalmát, a másik felüldefiniálja az `Equals(…)` metódust, hogy két `Coordinate` osztályt össze lehessen hasonlítani egymással.



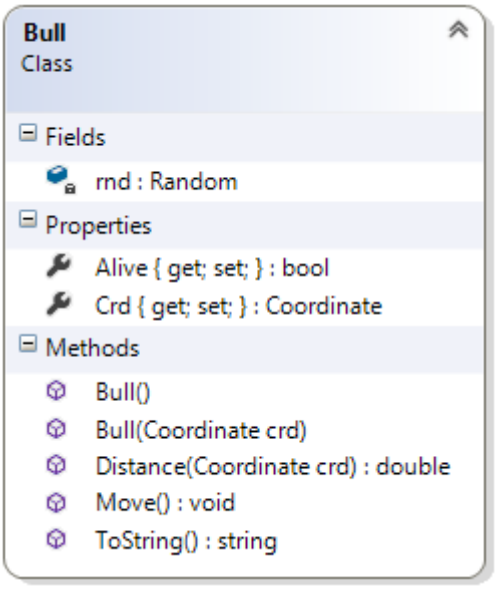
`class Map`

A pálya méretét tárolja el az osztály egy `Width` és egy `Height` tulajdonságban, amiknek értékét csak az osztályon belül lehet módosítani. A bölények célját az osztály egy `EndCord` tulajdonságban tárolja, ami szintén csak az osztályon belül módosítható. A tulajdonságok módosítását a `SetSize` metódussal lehet megtenni, ami kétféleképpen paraméterezhető. Egyik esetben a pálya szélességét és magasságát, másik esetben egy számtömböt kap bemenetként. Az osztály minden tagja osztályszintű.



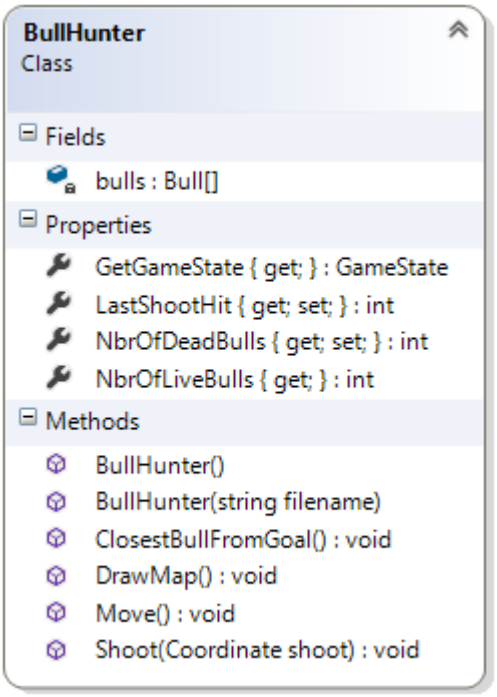
class Bull

A bölény reprezentálásához szükséges osztály, amiből a bölény objektumok fognak létrejönni. A bölények működéséhez szükséges a `Random` osztály osztályszintű példánya, ami a kezdőpozíció meghatározásán túl a léptetésnél is felhasználásra kerül. A bölény életét (`Alive` – `true` kezdőértékkel) és aktuális pozícióját (`Crd`) egy-egy tulajdonságban tároljuk el, melyek közül a pozíciót csak az osztályon belül lehet majd módosítani. A konstruktorból kettőfélét definiál az osztály, az egyik egy `Coordinate` példányt kap bemenetén, amit átad a megfelelő tulajdonságnak, a másik paraméter nélküli. A paraméter nélküli konstruktor egy véletlen pozícióba helyezi el a bölényt, úgy, hogy az nem lehet a célba, illetve annak közvetlen közelében. Az osztály rendelkezik még a `Move()` metódussal, ami a bölény mozgatásáért, a `Distance(…)` metódussal, ami a lövéstől vett távolság kiszámításáért és a felüldefiniált `ToString()` metódussal, ami az osztály tartalmának formázott megjelenítéséért felelős.



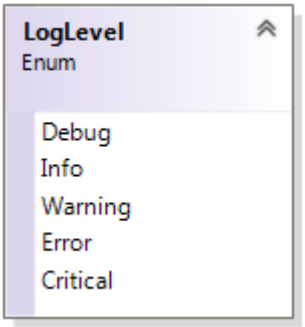
class BullHunter

A játék állapotát nyilvántartó osztály. A bölényeket egy `Bull` típusú tömbben tárolja el adatmezőként. Tulajdonságai az osztálynak megadják, hogy a játék milyen állapotban van (`GetGameState` – meghívásakor meghatározza a játék állapotát), hány bölényt lőtt le a felhasználó az utolsó lövésével (`LastShotHit` – írása csak az osztályon belül lehetséges), a halott (`NbrOfDeadBulls` – írása csak az osztályon belül lehetséges) és az élő (`NbrOfLiveBulls` – csak olvasható) bölények számát. Az osztály kettő konstruktorral rendelkezik, melyek közül a paraméter nélküli a paraméterrel rendelkező konstruktort hívja meg úgy, hogy a fájlnévként a `bullhunter.init`-et adja át. A paraméterezhető konstruktor feladata, hogy a fájlt beolvassa és létrehozza a `Bull` objektumokat. A `Move()` metódus hívásakor minden élő bölény léptetésre kerül. A `Shoot(…)` metódus a paramétereként kapott koordinátán lévő élő bölényeket megöli. A `ClosestBullFromGoal()` metódus kiírja a képernyőre a célhoz legközelebbi bölény távolságát. A `DrawMap()` metódus megjeleníti a pályát a rajta lévő halott bölényekkel, ha a játék véget ért, akkor az élő bölényeket is megjeleníti.



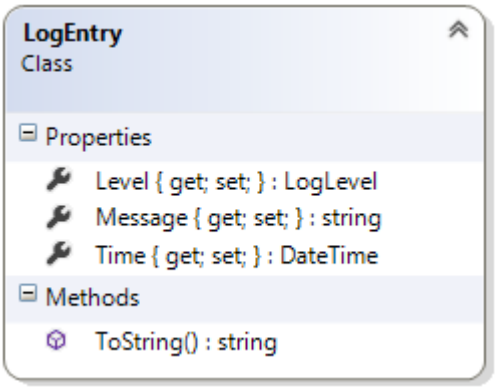
enum LogLevel

A típusban a logolás szintjei jelennek meg.



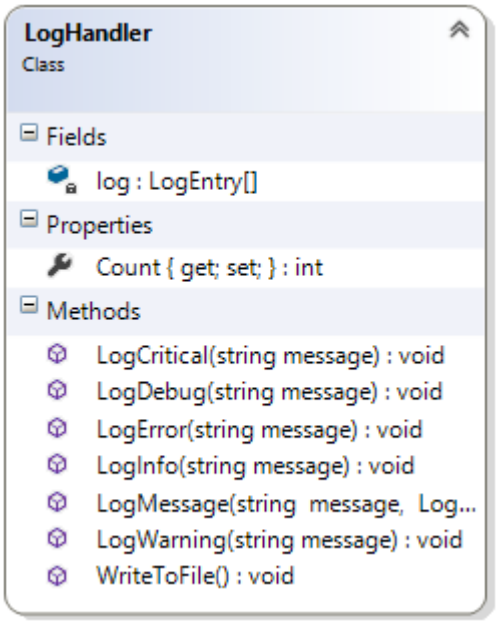
class LogEntry

Az osztály példánya fog leírni egy log bejegyzést. Három tulajdonsággal és egy metódussal rendelkezik az osztály, ami az osztály tartalmának megjelenítéséért felelős. A tulajdonságok a példányon kívül olvashatók és írhatók.



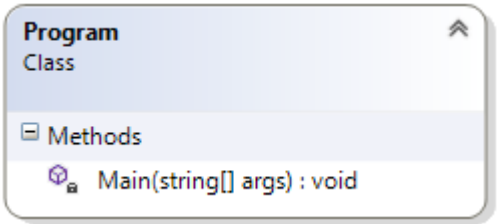
class LogHandler

A loggolást megvalósító osztály, aminek minden tagja osztályszintű. A keletkezett log üzeneteket egy `LogEntry` típusú tömbben tárolja el adatmezőként (aminek kezdeti mérete `4`). A tömbben lévő bejegyzések számát egy `Count` nevű, osztályon kívül csak olvasható tulajdonságon keresztül lehet elérni. A tömb bővítését, vagyis az új log üzenet eltárolását a `LogMessage(…)` metódus végzi, úgy, hogy a nagyobb tömb mérete a 2 következő hatványának méretével legyen egyenlő (4, 8, 16, 32…). Az osztály többi, loggolásért felelős metódusa ezt a metódust fogja hívni. A `WriteToFile()` metódus a log tömb tartalmát fájlba írja.



class Program

Az osztály létrehozza a `BullHunter` osztály példányát, majd a feladat leírásnak megfelelően körökbe szervezve lehetőséget biztosít a felhasználónak a játék lejátszására és a végeredmény megjelenítésére.



bullhunter.init

```
10
10 10
RND
```

bullhunter.init

```
5
10 10
5
{0,0}
{1,1}
{2,2}
{3,3}
{4,4}
```