0. Óra – Egyszerű osztályok készítése (játéktér)

Készítsük el az alapvető osztályokat, amelyekre majd a játék építkezni fog. A játék minden résztvevője a JatekElem osztály leszármazottja lesz, és mindezeket egy JatekTer objektum tartja majd össze.

OE.Prog2.Jatek.Jatekter névtéren belül készítsük el az alábbiakat:

Új osztály: **JatekElem**

A *JatekElem* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: x – egész szám, az elem pozíciójának x koordinátája
- Új privát mező: y – egész szám, az elem pozíciójának y koordinátája
- Új publikus tulajdonság: X – visszaadja és módosítja az x értéket.
- Új publikus tulajdonság: Y – visszaadja és módosítja az x értéket.

Új osztály: **JatekTer**

A *JatekTer* osztályt egészítsük ki az alábbiakkal:

- Új konstans: **MAX_ELEMSZAM** – egész szám, a tárolható elemek maximális száma (1000).
- Új privát mező: **elemN** – egész szám, ami azt mutatja, hogy éppen hány elemet tárolunk.
- Új privát mező: **elemek** – egy *MAX_ELEMSZAM* méretű, *JatekElem* objektumokat tartalmazó tömb.
- Új privát mező: **meretX** – egész szám, a játéktér maximális mérete x irányban.
- Új publikus tulajdonság: **MeretX** – csak olvasható, visszaadja a *meretX* értékét.
- Új privát mező: **meretY** – egész szám, a játéktér maximális mérete y irányban.
- Új publikus tulajdonság: **MeretY** – csak olvasható, visszaaadja a *meretY* értékét.
- Új publikus konstruktor: **JatekTer** – két paramétere legyen, amelyek beállítják a *meretX* és *meretY* mezők értékét.
- Új publikus metódus: **Felvetel** – felveszi a paraméterként átadott *JatekElem* típusú objektumot az *elemek* tömbbe. Értelemszerűen növeli az *elemN* mező értékét is.
- Új publikus metódus: **Torles** – törli a paraméterként átadott *JatekElem* objektumot az elemek tömbből (és értelemszerűen csökkenti az *elemN* mező értékét is).
- Új publikus metódus: **MegadottHelyenLevok** – három paramétere van: egy x és y koordináta, illetve egy távolság, a visszatérési értéke pedig egy *JatekElem* objektumokat tartalmazó tömb. Az *elemek* tömb adatai alapján meg kell számolnia, hogy az x és y koordináták által meghatározott ponttól mért *távolság* távolságon belül hány darab *JatekElem* objektum található. Létre kell hoznia egy ekkora *JatekElem* tömböt, majd ebbe ki kell válogatnia az előző feltételnek megfelelő elemeket. Ez lesz a metódus visszatérési értéke.
- Új publikus metódus: **MegadottHelyenLevok** – ez előzőhöz hasonló, de csak x és y paramétere van. Visszatérési értéke egy *JatekElem* tömb, ami azokat az elemeket tartalmazza, amelyek pont az x és y által megadott helyen vannak (célszerű felhasználni az előző metódust, 0 távolsággal).

A már meglévő JatekElem osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **ter** – tárolja annak a *JatekTer* típusú játéktérnek a referenciáját, amelyikbe elhelyeztük.
- Új publikus konstruktor: **JatekElem** – paraméterként megkapja az x és y koordinátákat, illetve egy *JatekTer* referenciát. Mindhárom paramétert mentse el az ezekre szolgáló mezőkbe, majd vegye fel magát a *JatekTer* objektum *Felvetel* metódusával.

Tesztelés: A főprogramban készítsünk egy JatekTer objektumot, illetve néhány ebbe elhelyezkedő JatekElem objektumot. Próbáljuk ki a megvalósított metódusokat:

- *Töröljünk egy játékelemet!*
- *Adjuk meg, hogy egy megadott helyen épp hány játékelem található!*
- *Listázzuk egy megadott pont megadott környezetében lévő elemek koordinátáit!*

1. Óra – Öröklés és polimorfizmus (játékos, fal, kincs)

Készítsük el a játékosokat, amelyek majd mozogni fognak a játéktéren belül. Egyelőre csak az emberi játékost implementáljuk teljesen, aki a billentyűzettel lesz irányítható. Az örökléssel állítsuk fel az alapvető osztály hierarchiát, és implementáljuk az egyszerűbb osztályokat (fal, kincs, stb.).

OE.Prog2.Jatek.Jatekter névtéren belül készítsük el az alábbiakat:

A már meglévő *JatekElem* osztályt egészítsük ki az alábbiakkal:

- Maga a *JatekElem* osztály legyen absztrakt (az előző órai főprogramot emiatt ki is törölhetjük, mivel nem fog lefordulni).
- A **ter** mező legyen védett láthatósági szintű, hogy a leendő leszármazottak hozzáférhessenek.
- Új absztrakt publikus tulajdonság: **Meret** – lebegőpontos szám, csak olvasható, ezen a szinten még nem tudjuk meghatározni (0..1 közötti értéke lehet majd, egy mezőbe elhelyezkedő elemek mérete nem lehet 1-nél több).
- Új absztrakt publikus metódus: **Utkozes** – paraméterként egy *JatekElem* objektumot kap, ezen a szinten még nem tudjuk meghatározni (ez fogja majd kezelni a különböző típusú elemek ütközéseit).

Új absztrakt osztály: **RogzitettJatekElem**, ami a *JatekElem* leszármazottja

A *RogzitettJatekElem* osztályt egészítsük ki az alábbiakkal:

- Új publikus konstruktor: **RogzitettJatekElem** - paraméterként megkapja az x és y koordinátákat, illetve egy *JatekTer* referenciát. Ezeket továbbítsa az ősének konstruktorához.

Új absztrakt osztály: **MozgoJatekElem**, ami a *JatekElem* leszármazottja

A *MozgoJatekElem* osztályt egészítsük ki az alábbiakkal:

- Új publikus konstruktor: **MozgoJatekElem** - paraméterként megkapja az x és y koordinátákat, illetve egy *JatekTer* referenciát. Ezeket továbbítsa az ősének konstruktorához.
- Új privát mező: **aktiv** – logikai mező, ami azt mutatja majd, hogy működik/él-e még az adott objektum.
- Új publikus tulajdonság: **Aktiv** – visszaadja, illetve felülírja az *aktiv* mező tartalmát.
- Új publikus metódus: **AtHelyez** – két egész szám paramétert kap, ujx és ujy. Az ide való átlépés az alábbi műveletekből álljon:
  - A metódus kérdezze le, hogy az új koordináták által megadott helyen éppen milyen más játékelemek találhatók (ez a *JatekTer* segítségével egyszerű).
  - Ezeknek az elemeknek egyesével hívja meg az *Utkozes* metódusát, paraméterként átadva önmagát (tehát a mozgatandó objektum nekiütközik a cél helyen lévőknek).
  - Majd tegye meg ezt fordítva is (tehát a cél helyen lévő ütközik neki a mozgatandó objektumnak).

- Minden ütközés után ellenőrizze, hogy még *Aktiv*-e a mozgatandó objektum (mivel lehet, hogy valamelyik ütközés hatására meghalt). Ha már nem az, akkor ne folytassa az ütközéseket.
- Ha az ütközések után még *Aktiv* a mozgatandó objektum, akkor kérje le ismét a cél helyen található elemeket (mivel azok között is lehetett halál az üközések miatt).
- Számolja ki, hogy mennyi a cél helyen már meglévő elemek összesített mérete (*Meret* tulajdonságok).
- Ha ehhez még hozzáadjuk az odaléptetni kívánt elem méretét, akkor ez nem haladhatja meg az 1-et. Ha ez alapján elvégezhető a lépés, akkor az objektum *x* és *y* koordinátáit léptessük az új helyre. Ha nem, akkor ne változtassuk meg.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új osztály: **Fal**, ami a *RogzitettJatekElem* leszármazottja

A *Fal* osztályt egészítsük ki az alábbiakkal:

- Új konstruktor: **Fal** – az őséhez hasonlóan x és y és játéktér paramétereket kap, ezeket továbbítja az ősnek.
- Új publikus felülírt mező: **Meret** – az ősben még absztrakt méretet itt már meg tudjuk határozni. Mindig adjon vissza 1-et (tehát egy fal mellé más már biztos nem fog elférni).
- Új publikus felülírt metódus: **Utkozes** – ne csináljon semmit, a fal passzív.

Új osztály: **Jatekos**, ami a *MozgoJatekElem* leszármazottja

A *Jatekos* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **nev** – szöveg, tárolja a játékos nevét.
- Új publikus tulajdonság: **Nev** – visszaadja a játékos nevét.
- Új konstruktor: **Jatekos** – négy paramétere van: név, x, y, játéktér. Az elsőt eltárolja a megfelelő mezőben, a többit továbbítja az ős konstruktorának.
- Új publikus felülírt mező: **Meret** – Mindig adjon vissza 0.2-t (tehát 5 játékos fér el egy mezőn, ha nincs ott más).
- Új publikus felülírt metódus: **Utkozes** – ne csináljon semmit.
- Új privát mező: **eletero** – egész szám, kezdőértéke 100. Tárolja a játékos életerejét.
- Új publikus metódus: **Serul** – paramétere egy egész szám. Ha a játékos életereje már eleve 0, akkor ne csináljon semmit. Ha nagyobb, akkor csökkentse az *eletero*-t a paraméterként átadott értékkel (de 0-nál kisebb ne legyen). Ha a játékos életereje elérte a 0-t, akkor az *Aktiv* tulajdonságot állítsa hamisra.
- Új privát mező: **pontszam** – egész szám, kezdőértéke 0. Tárolja, hogy a játékos eddig hány pontot szerzett a játékban.
- Új publikus metódus: **PontotSzerez** – paramétere egy egész szám. A megadott paraméterrel növeli a *pontszam* mező értékét.
- Új publikus metódus: **Megy** – két paramétere két relatív koordináta (rx és ry, tipikusan -1, 0, 1 értékeket várunk ezekben). Számolja ki, hogy az aktuális pozícióból ezzel az elmozdulással hova jutna, és hívja meg az ősétől örökölt *Athelyez* függvényt ezekkel a koordinátákkal.

Új osztály: **Kincs**, ami a *RogzitettJatekElem* leszármazottja

A *Kincs* osztályt egészítsük ki az alábbiakkal:

- Új konstruktor: **Kincs** – az őséhez hasonlóan x és y és játéktér paramétereket kap, ezeket továbbítja az ősnek.
- Új publikus felülírt mező: **Meret** – Mindig adjon vissza 1-et.
- Új publikus felülírt metódus: **Utkozes** – Ez akkor fog lefutni, ha valaki nekiütközött egy kincs objektumnak. Ilyekor a teendők:
  - Ellenőrizzük, hogy a nekiütköző (tehát a paraméterként kapott *JatekElem*) egy *Jatekos* típusú objektum-e.
  - Ha igen, akkor hívjuk meg annak *PontSzerez* metódusát, paraméterként 50-et átadva (tehát a kincsért kapott 50 pontot).
  - A kincs törölje önmagát a játéktérről (a *JatekTer* osztály *Torles* metódusával).

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

Új osztály: **Keret**

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Új konstans: **PALYA_MERET_X** – a pálya maximális mérete x irányba (21)
- Új konstans: **PALYA_MERET_Y** – a pálya maximális mérete y irányba (11)
- Új konstans: **KINCSEK_SZAMA** – a pályán lévő kincsek száma (10)
- Új privát mező: **ter** – egy *JatekTer* objektum, ami a játékteret fogja tárolni.
- Új privát metódus: **PalyaGeneralas** – hozzon létre és a *ter*-be helyezzen el **PalyaElemeket** az alábbiak szerint:
  - A pálya négy szélén legyenek falak (tehát *Fal* objektumok)
  - Véletlenszerűen szórjon szét *KINCSEK_SZAMA* darab kincset a pályán belül, ügyelve arra, hogy egy helyre nem fér el egynél több *Kincs* objektum (az 1,1 pozíciót is hagyjuk szabadon, innen indul majd a játékos).
- Új publikus konstruktor: **Keret** – hozza létre a *PalyaTer* objektumot a maximális mérettel, majd hívja meg a *PalyaGeneralas* metódust.
- Új privát mező: **jatekVege** – alapértelmezetten hamis, csak akkor lesz igaz, ha vége a játéknak.
- Új publikus metódus: **Futtatas** – ez fogja elindítani és működtetni az egész játékot. A metódus fő feladatai:
  - Létrehoz egy játékost „Béla” néven az 1,1 pozícióba a játéktérben
  - Egy ciklus addig fusson, amíg a *jatekVege* metódus nem vált igazra. A cikluson belül kérjen be egy billentyűleütést, és ha ez egy kurzor gomb volt, akkor mozgassa Bélát ebbe az irányba, ha az Esc billentyű, akkor pedig lépjen ki.

```
do
{
    ConsoleKeyInfo key = Console.ReadKey(true);
    if (key.Key == ConsoleKey.LeftArrow) jatekos.Megy(-1, 0);
    if (key.Key == ConsoleKey.RightArrow) jatekos.Megy(1, 0);
    if (key.Key == ConsoleKey.UpArrow) jatekos.Megy(0, -1);
    if (key.Key == ConsoleKey.DownArrow) jatekos.Megy(0, 1);
    if (key.Key == ConsoleKey.Escape) jatekVege = true;
} while (!jatekVege);
```

Tesztelés: A főprogramban hozzunk létre egy Keret objektumot, és hívjuk meg a Futtatas metódusát. Nincs még megjelenítési rétegünk, így nem látunk belőle semmit, de a vezérlés és az üzleti logika már működik, a játék használható (a játékos mozog, kincseket fel lehet venni, nem lehet rámenni a falakra, stb.). Bár határozottan elítéljük az üzleti logikából a konzolra való kiírást, de tesztelési célokból ez elfogadható: egészítsük ki úgy a Kincs osztályt, hogy a konstruktorban írja ki a képernyőre

az x és y koordinátát (tehát a program indulásakor megjelenik a 10 elkészített kincs helye), illetve az Utkozes metódusát úgy, hogy a pont növelésekor írja ki a konzolra a Jatekos nevét. Így a megjelenítést helyettesítendő, a kincseket papírra felrajzolva, vakon tudjuk teljesíteni a pályát.

2. Óra – Interfészek (megjelenítési szint)

Készítsük el a játék megjelenítési rétegét! Az alap működés nem változik, azonban a játéktérben lévő elemeket megjelenítjük a konzolablakban.

Töltsük le a **SzalbiztosKonzol.cs** filet, és másoljuk a projectünkhöz. Ennek az osztálynak a statikus metódusaival tudunk a képernyő megadott pozíciójába egy karaktert, vagy egy szöveget kiírni.

OE.Prog2.Jatek.Megjelenites névtéren belül készítsük el az alábbiakat:

Új publikus interfész: **IKirajzolhato**

Az *IKirajzolhato* interfészt egészítsük ki az alábbiakkal:

- Új tulajdonság: **X** – csak olvasható egész. A megjelenítendő alak x koordinátája.
- Új tulajdonság: **Y** – csak olvasható egész. A megjelenítendő alak y koordinátája.
- Új tulajdonság: **Alak** – csak olvasható karakter. A jel, ami megjelenik majd.

Új publikus interfész: **IMegjelenitheto**

Az *IMegjelenitheto* interfészt egészítsük ki az alábbiakkal:

- Új tulajdonság: **MegjelenitendoMeret** – csak olvasható egész tömb. Mindig egy két elemű egész tömböt várunk, ami tartalmazza a megjelenítendő terület szélességét és magasságát.
- Új metódus: **MegjelenítendőElemek** – nincs paramétere, visszaad egy *IKirajzolhato* elemekből álló tömböt.

Új publikus osztály: **KonzolosMegjelenito**

A *KonzolosMegjelenito* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **forras** – egy *IMegjelenitheto* típusú mező, innen jönnek majd a megjelenítendő adatok.
- Új privát mező: **pozX** – ennyivel kell majd eltolni a kirajzolandó adatok x koordinátáját (ez akkor lesz érdekes, ha több *KonzolosMegjelenito* is dolgozik egyidőben, akkor más-más helyre rajzolják ki az adataikat).
- Új privát mező: **pozY** – ennyivel kell majd eltolni a kirajzolandó adatok y koordinátáját.
- Új publikus konstruktor: **KonzolosMegjelenito** – paraméterként kapott három értékkel feltölti az előző három mezőt.
- Új publikus metódus: **Megjelenites** – kiolvassa a *forras*-ból a kirajzolandó elemeket, majd ezeket kirajzolja a *pozX* és *pozY* által eltolt helyre:
  - A *forras* objektum *MegjelenitendoElemek* metódusával kiolvassa a kirajzolandó elemek tömbjét.
  - Lekérdezi azt is, hogy mekkora a megjelenítendő terület mérete (szintén a forrás adja meg egy két elemű tömbben a szélességet és a magasságot).
  - Ezt követően két egymásba ágyazott ciklussal végigszaladunk a méret által megadott területen, ha
    - a forrás által visszaadott objektumok egyike sincs a megadott ponton, akkor kiírunk oda egy szóközt a *SzalbiztosKonzol* segítségével,

        - a megadott ponton van valami, akkor pedig annak az *Abra* tulajdonsága által visszaadott karaktert írjuk ki a megadott helyre.
    - A megjelenítés során a kirajzolandó elem x és y koordinátáit mindig toljuk el a megjelenítő *pozX* és *pozY* mezőjének értékével.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

A *Fal* osztály valósítsa meg az *IKirajzolhato* interfészt az alábbiak szerint:

- Az interfész által igényelt *X* és *Y* tulajdonságokat már az ős *JatekElem* osztály megvalósította, így ezeket implicit módon már implementáltuk.
- Új publikus tulajdonság: **Alak** – csak olvasható, visszatérési értéke '\u2593'

A *Jatekos* osztály valósítsa meg az *IKirajzolhato* interfészt az alábbiak szerint:

- Az interfész által igényelt *X* és *Y* tulajdonságokat már az ős *JatekElem* osztály megvalósította, így ezeket implicit módon már implementáltuk.
- Új publikus tulajdonság: **Alak** – csak olvasható, visszatérési értéke legyen attól függően, hogy ha még aktív, akkor '\u263A', különben '\u263B'

A *Kincs* osztály valósítsa meg az *IKirajzolhato* interfészt az alábbiak szerint:

- Az interfész által igényelt *X* és *Y* tulajdonságokat már az ős *JatekElem* osztály megvalósította, így ezeket implicit módon már implementáltuk.
- Új publikus tulajdonság: **Alak** – csak olvasható, visszatérési értéke '\u2666'

OE.Prog2.Jatek.JatekTer névtéren belül készítsük el az alábbiakat:

A *JatekTer* osztály valósítsa meg az *IMegjelenitheto* interfészt az alábbiak szerint:

- Új publikus tulajdonság: **MegjelenitendoMeret** – azt adja vissza, hogy mekkora területet kell majd megjeleníteni. A tulajdonság hozzon létre egy két elemű egész tömböt, ebbe helyezze el a *meretX* és *meretY* mezők értékét, majd ezt adja vissza.
- Új publikus metódus: **MegjelenitendoElemek** – ez adja vissza azoknak az elemeknek a listáját, amelyeket ki kell rajzolni. Ehhez:
    - Számolja meg, hogy az *elemek* tömbben hány olyan objektum van, aki megvalósítja az *IKirajzolhato* interfészt.
    - Hozzon létre egy ekkora, *IKirajzolhato* nevű tömböt **vissza** néven.
    - Ebbe a tömbbe válogassa ki az interfészt megvalósító elemeket.
    - Majd adja vissza ezt a tömböt visszatérési értékként.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a metódus elején hozzunk létre egy új *KonzolosMegjelenito* objektumot, paraméterként átadva a *ter* objektumot forrásként, és a 0,0 koordinátákat megjelenítési pozícióként.
- Meglévő metódus módosítása: **Futtatas** – a billentyűleütéseket figyelő ciklust egészítsük ki azzal, hogy minden iterációban meghívja ennek a megjelenítőnek a *Megjelenites* metódusát.

Már működik a megjelenítési szint, az előző órán, a *Kincs* osztályba írt konzol kiírásokat töröljük!

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

A *Jatekos* osztály valósítsa meg az *IMegjelenitheto* interfészt az alábbiak szerint:

- Új publikus tulajdonság: **MegjelenitendoMeret** – mivel a játékosnak van egy referenciája ahhoz a *JatekTer*-hez, amelyikben van, így adja vissza annak a méreteit.
- Új publikus tulajdonság: **MegjelenitendoElemek** – a játékos a pálya 5 sugarú környezetét látja be, így az itt található elemeket fogja visszaadni. Ehhez:
  - Kérje le a játéktértől az ő 5 sugarú környezetében található elemek tömbjét.
  - Ebben számolja meg, hogy hány olyan objektum van, ami megvalósítja az *IKirajzolhato* interfészt.
  - Hozzon létre egy ekkora, *IKirajzolhato* nevű tömböt **vissza** néven.
  - Ebbe a tömbbe válogassa ki az interfészt megvalósító elemeket.
  - Majd adja vissza ezt a tömböt visszatérési értékként.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a metódus elején hozzunk létre egy második *KonzolosMegjelenito* objektumot is, ennek paraméterként átadva a Béla objektumot forrásként, és a 25,0 koordinátákat megjelenítési pozícióként.
- Meglévő metódus módosítása: **Futtatas** – a billentyűleütéseket figyelő ciklust egészítsük ki azzal, hogy minden iterációban meghívja ennek a megjelenítőnek is a *Megjelenites* metódusát.

Tesztelés: A fentieket megvalósítva működik a vezérlés és az üzleti logika, emellett elkészült a megjelenítési szint. Ezt rá tudjuk kapcsolni vagy az egész pályára, akkor mindent látunk (debug mód), vagy rá tudjuk kapcsolni az egyes játékosokra, ez lesz a végleges játék irányító képernyője.

3. Óra – Események (automatizmusok, ellenfelek)

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **GepiJatekos**, ami a *Jatekos* osztály leszármazottja

A *GepiJatekos* osztályt egészítsük ki az alábbiakkal:

- Új konstruktor: **GepiJatekos** – Paraméterei ugyanazok mint az ős konstruktorának, csak hozzá továbbítsa a kapott adatokat.
- Új publikus metódus: **Mozgas** – Generáljunk egy véletlen számot 0 és 3 között. Majd ennek értékétől függően az objektum próbáljon meg elmozdulni (célszerű használni az örökölt *Megy* metódust) fel-jobbra-le-balra irányok közül valamerre. A véletlen számokhoz használt Random osztály legyen statikus mező, ezzel elkerülhető, hogy az időben közvetlenül egymás után példányosított játékosok random generátorai ugyanazokat a számokat adják.
- Új publikus felülírt mező: **Alak** – Visszatérési értéke legyen '\u2640'

Új publikus osztály: **GonoszGepiJatekos**, ami a *GepiJatekos* osztály leszármazottja

A *GonoszGepiJatekos* osztályt egészítsük ki az alábbiakkal:

- Új konstruktor: **GonoszGepiJatekos** – Paraméterei ugyanazok mint az ős konstruktorának, csak hozzá továbbítsa a kapott adatokat.
- Publikus felülírt mező: **Alak** – Visszatérési értéke legyen '\u2642'
- Új publikus metódus: **Utkozes** – Hívja meg az ős *Utkozes* metódusát. Ha még ezek után *Aktiv* a játékos, és a paraméterként átadott ütköző objektum egy *Jatekos* objektum (vagy annak leszármazottja), akkor hívja meg annak *Serul* metódusát paraméterként 10-et adva át.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a metódus elején hozzunk létre egy *GepiJatekos* objektumot Kati néven, aki az eddigi játékossal egy térben egy másik ponton indul, illetve egy *GonoszGepiJatekos* objektumot Laci néven, aki az eddigi játékossal egy térben, de másik ponton indul (próbaképp létrehozhatunk megjelenítőket is ezekhez az objektumokhoz, hogy nyomon követhetjük a mozgásukat).
- Meglévő metódus módosítása: **Futtatas** – a billentyűleütéseket figyelő ciklust egészítsük ki azzal, hogy mindkét gépi játékosnak hívja meg a *Mozgas* nevű metódusát. Az ellenfelek ezzel mozogni kezdenek, de mindig csak akkor, amikor az emberi játékos is lép.

OE.Prog2.Jatek.Automatizmus névtéren belül készítsük el az alábbiakat:

Új publikus interfész: **IAutomatikusanMukodo**

Az *IAutomatikusanMukodo* interfészt egészítsük ki az alábbiakkal:

- Új metódus: **Mukodik** – ez a paraméter nélküli metódus automatikusan meg fog hívódni bizonyos időközönként.
- Új tulajdonság: **MukodesIntervallum** – csak olvasható egész (tizedmásodperc). Az interfészt megjelenítő objektum Mukodik metódusa ilyen időközönként fog meghívódni.

Töltsük le az **OrajelGenerator.cs** filet és a benne lévő **OrajelGenerator** osztályt másoljuk be ebbe a névtérbe. Ez az osztály egy időzítőt tartalmaz, amihez a *Felvetel* metódus segítségével fel tudnak iratkozni *IAutomatikusanMukodo* interfészt megvalósító objektumok. Az órajel-generátor *Aktivalas* metódusa tizedmásodpercenként lefut egyszer, és elvégzi az alábbi tevékenységeket: 1) Megnézi az összes feliratkozott objektumot, hogy most épp meg kell-e hívni a *Mukodik* metódust 2) Ha igen, akkor meghívja ezt a metódust.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

A *GepiJatekos* osztály valósítsa meg az *IAutomatikusanMukodo* interfészt az alábbiak szerint:

- Új publikus metódus: **Mukodik** – hívja meg a már létező *Mozgas* metódust.
- Új publikus tulajdonság: **MukodesIntervallum** – mindig adjon vissza 2-t (tehát 2 tizedmásodpercenként szeretnénk léptetni az ellenfelet).

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **generator** – egy *OrajelGenerator* típusú objektum legyen.
- Meglévő konstruktor módosítása: **Keret** - hozza létre az előző *generator* nevű objektumot.
- Meglévő metódus módosítása: **Futtatas** – a metódust módosítsuk úgy, hogy a létrehozott Kati és Laci objektumokat adjuk át paraméterként az *OrajelGenerator* objektum *Felvetel* metódusának. Az ellenfelek ezzel már maguktól mozognak, de a kép frissítése még mindig a billentyű lenyomásokhoz igazodik. Ha a fő ciklusban előzőleg közvetlenül hívtuk az ellenfelek *Mozgas* metódusát, akkor ezt már törölhetjük.

OE.Prog2.Jatek.Megjelenites névtéren belül készítsük el az alábbiakat:

A *KonzolosMegjelenito* osztály valósítsa meg az *IAutomatikusanMukodo* interfészt az alábbiak szerint:

- Új publikus metódus: **Mukodik** – hívja meg a már létező *Megjelenites* metódust.
- Új publikus tulajdonság: **MukodesIntervallum** – mindig adjon vissza 1-t (tehát minden órajel-ciklusban frissíteni kell a képernyőt).

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a metódust módosítsuk úgy, hogy a létrehozott megjelenítő objektumokat is kapcsoljuk rá az órajel-generátorra. Ezzel a képernyő frissítés is független a billentyűzet kezeléstől.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új delegált típus: **KicsFelvetelKezelo** – két paraméterrel rendelkezzen:

- A *Kincs* referenciája amit felvettek.
- A *Jatekos* referenciája, aki felvette.

A *Kincs* osztályt egészítsük ki az alábbiakkal:

- Új publikus esemény: **KincsFelvetel** – legyen egy *KincsFelvetelKezelo* típusú eseménykezelő.
- Meglévő metódus módosítása: **Utkozes** – amennyiben egy játékos felvette a kincset, és valaki feliratkozott a fenti eseménykezelőre, akkor küldjön az eseményről egy értesítést.

Új delegált típus: **JatekosValtozasKezelo** – három paraméterrel rendelkezzen:

- Az eseményt küldő *Jatekos* referenciája.
- A játékos új pontszáma.
- A játékos új életereje.

A *Jatekos* osztályt egészítsük ki az alábbiakkal:

- Új publikus esemény: **JatekosValtozas** – legyen egy *JatekosValtozasKezelo* típusú eseménykezelő.
- Meglévő metódus módosítása: **Serul** – amennyiben változott a játékos életereje, és valaki feliratkozott a fenti eseménykezelőre, akkor küldjön az eseményről egy értesítést.
- Meglévő metódus módosítása: **PontotSzerez** – amennyiben változott a játékos pontszáma, és valaki feliratkozott a fenti eseménykezelőre, akkor küldjön az eseményről egy értesítést.

OE.Prog2.Jatek.Megjelenites névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **KonzolosEredmenyAblak**

A *KonzolosEredmenyAblak* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **pozX** – ennyivel kell majd eltolni a kirajzolandó adatok x koordinátáját (hasonló a *KonzolosMegjelenito*-höz).
- Új privát mező: **pozY** – ennyivel kell majd eltolni a kirajzolandó adatok y koordinátáját.
- Új privát mező: **maxSorSzam** – hány sor fér ki az eredményablakba.
- Új privát mező: **sor** – aktuális sorok száma (alapból 0).
- Új publikus konstruktor: **KonzolosEredmenyAblak** – adjon értéket az első három mezőnek.
- Új privát metódus: **JatekosValtozasTortent** – A *JatekosValtozasKezelo* delegáltnak megfelelő paraméterekkel rendelkezzen. A *SzalbiztosKonzol* segítségével írja ki a *pozX* és *pozY* által

megadott helyre, az aktuális sornak megfelelő sorba az alábbi adatokat „játékos neve:…, pontszáma:…, életereje:…”. Majd növelje a *sor* értékét. Ha ez nagyobb mint a *maxSorSzam*, akkor állítsa 0-ra (ezzel elkezdi felülírni a legrégebbi eseményeket).
- Új publikus metódus: **JatekosFeliratkozas** – paraméterként egy *Jatekos* objektumot kap. Az objektum *JatekosValtozas* eseményére iratkozzon fel az előző metódussal.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a metódust módosítsuk úgy, hogy hozzon létre egy új *KonzolosEredmenyAblak* objektumot a 0,12 pozícióban, maximálisan 5 sorral. Majd adjuk át a „Béla” játékos objektumot a *JatekosFeliratkozas* metódusának.
- Új privát mező: **megtalaltKincsek** – ez fogja számolni, hogy hány kincset vettek már fel.
- Új privát metódus: **KincsFelvetelTortent** – a *KincsFelvetelKezelo* delegáltnak megfelelő paraméterekkel rendelkezzen. Növelje a *megtalaltKincsek* változó értékét. Ha ez elérte a *KINCSEK_SZAMA* konstans értéket, akkor a *jatekVege* változo értéke legyen igaz.
- Meglévő metódus módosítása: **PalyaGeneralas** – a metódust módosítsuk úgy, hogy minden létrehozott *Kincs* objektum *KincsFelvetel* eseménykezelőjéhez kapcsoljuk hozzá az előző metódust.
- Új privát metódus: **JatekosValtozasTortent** – a *JatekosValtozasKezelo* delegáltnak megfelelő paraméterekkel rendelkezzen. Amennyiben a paraméterként kapott életerő 0, akkor állítsa a *jatekVege* változót igazra.
- Meglévő metódus módosítása: **Futtatas** – a metódust módosítsuk úgy, hogy az emberi játékos létrehozása után a *JatekosValtozas* eseményre iratkozzon fel az előző metódus.

Tesztelés: véletlenszerűen mozogniuk kell az ellenfeleknek, automatikusan frissülnie kell a képernyőnek, és a játékos eredményeinek meg kell jelennie az új ablakban. Mind a kincsek elfogyásakor, mind az életerő 0-ra csökkenésekor véget kell érnie a játéknak.

4. Óra – Kivételkezelés

Az eddig figyelmen kívül hagyott hibás illetve kivételes esetek kezelése.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **MozgasNemSikerultKivetel**, ami az *Exception* osztály leszármazottja

A *MozgasNemSikerultKivetel* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **jatekElem** – ez tárolja, hogy ki nem tudott lépni.
- Új publikus tulajdonság: **JatekElem** – a fentit adja vissza.
- Új privát mező: **x** – ez tárolja, hogy hova szerett volna lépni (x koordináta)
- Új publikus tulajdonság: **X** – a fentit adja vissza.
- Új privát mező: **y** – ez tárolja, hogy hova szerett volna lépni (y koordináta)
- Új publikus tulajdonság: **Y** – a fentit adja vissza.
- Új publikus konstruktor: **MozgasNemSikerultKivetel** – beállítja az előzőket.

Új publikus osztály: **MozgasHalalMiattNemSikerultKivetel**, ami a *MozgasNemSikerultKivetel* osztály leszármazottja

A *MozgasHalalMiattNemSikerultKivetel* osztályt egészítsük ki az alábbiakkal:

- Új publikus konstruktor: **MozgasHalalMiattNemSikerultKivetel** – a szükséges adatokat továbbítja az ősének.

Új publikus osztály: **MozgasHelyHianyMiattNemSikerultKivetel**, ami a *MozgasNemSikerultKivetel* osztály leszármazottja

A *MozgasHelyHianyMiattNemSikerultKivetel* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **elemek** – egy **JatekElem** tömb, ami azokat az elemeket fogja tároli, amelyek miatt nem sikerült a lépés.
- Új publikus csak olvasható tulajdonság: **Elemek** – visszaadja az előző tömböt.
- Új publikus konstruktor: **MozgasHelyHianyMiattNemSikerultKivetel** – paraméterei azonosak az ős konstruktor paramétereivel, kiegészítve az új elemek tömbbel. Az előzőket továbbítja az ős konstruktorának, az utóbbit beállítja.

A *MozgoJatekElem* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **AtHelyez** – egészítsük ki a meglévő működést az alábbiakkal:
  - Ha az ütközések hatására meghalt a mozgatni kívánt elem, akkor dobjunk egy *MozgasHalalMiattNemSikerult* kivételt a megfelelő paraméterekkel.
  - Ha az ütközéseket túlélte, de a cél területen már meglévő objektumok mellé már nem fért el az elem, akkor dobjon egy *MozgasHelyHianyMiattNemSikerult* kivételt a megfelelő paraméterekkel.

A *GepJatekos* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Mozgas** – Amennyiben a *Megy* metódus hívásakor egy *MozgasHelyHianyMiattNemSikerult* kivételt kapnánk, akkor automatikusan próbálkozzon a következő irányba lépni. Mindezt addig folytassa, amíg sikerül lépnie, vagy már mind a négy lehetséges irányt végigpróbálgatta.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – a program fő ciklusában a mozgatást kezelő feltételeket helyezzük egy kivételkezelő blokkba. Amennyiben valamelyik metódus hívásából egy MozgasHelyHanyMiattNemSikerult kivétel érkezne, akkor ezt kapjuk el, és a program sípoljon egyet (minél több dolognak ütköztünk, annál magasabb frekvencával): System.Console.Beep(500 + e.Elemek.Length*100, 10);

OE.Prog2.Jatek.Automatizmus névtéren belül készítsük el az alábbiakat:

Az *OrajelGenerator* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Aktivalas** – elképzelhető, hogy a *Mukodik()* metódus hívása egy kivételt dob. Nem szeretnénk, hogy emiatt álljon le a program, ezért kapjunk el minden kivételt, és a kivétel objektum adatait írjuk bele egy „log.txt” nevű fájlba.

Tesztelés: Külső szemlélő számára nincs lényeges változás, a program felépítése azonban jelentősen javult.

5. Óra – Backtrack (kincsek elhelyezése)

Labirintust generáló kód elkészítése:

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat

A meglévő *Keret* osztály meglévő *PalyaGeneralas* metódusát módosítsuk az alábbiak szerint:

A pályát körbeölelő falak létrehozását követően készítsük el a belső labirintust. Ehhez az alábbi adatszerkezeteket használhatjuk:

- Új lokális változó: ***falN*** – egész szám, ami a labirintusba eddig beépített falak számát mutatja.
- Új lokális változó: ***falak*** – *Fal* objektumok tömbje (mérete MeretX * MeretY), amely a labirintusba már beépített falakat tárolja.

A tömb első elemébe el is helyezhetünk egy 2,2 koordinátájú *Fal* objektumot, ennek megfelelően a *falN* változó értéke is legyen 1.

Ezt követően futtassuk az alábbi ciklust:

- A falak tömbben válasszunk ki egy véletlen elemet 0 .. falN-1 között (*Fal* objektumot).
- Vegyünk egy véletlen számot 1 és 4 között.
- A számnak megfelelő irányban (pl. 1 – fel, 2 – le. 3 – jobbra, 4 – balra) vizsgáljuk meg, hogy a véletlenül kiválasztott *Fal* objektumtól 2 lépésnyi távolságban van-e másik fal.
- Ha nincs, akkor hozzunk létre egy új *Fal* objektumot a vizsgált helyen, ezt tegyük be a *falak* tömbbe is. Emellett hozzunk létre egy új *Fal* objektumot a megadott irányba egy lépés távolságba is (tehát a régi és az új fal közá), ezt ne rakjuk be a *falak* tömbbe.
- Ha a megadott irányban már van másik fal, akkor növeljük az irányt adó szám értékét, hogy átvizsgáljuk mind a 4 irányt. Ha valamelyik irány szabad, akkor végezzük el a fentieket.
- Ha egyik irányba se találunk szabad terület, akkor nézzük a következő falat (ne újabb random számot dobjunk, hanem egyszerűen nézzük a *falak* tömbben lévő következő objektumot), és a fentiek szerint vizsgáljuk meg annak bővíthetőségét.
- Ha egy olyan falat se találtunk, amelyet lehetne bővíteni, akkor lépjünk ki a ciklusból (hiszen akkor átvizsgáltuk a labirintusba beépített összes fal összes irányát, és nincs további bővítési lehetőségünk).

Egészítsük ki a programot azzal, hogy mind a játékosok, mind pedig a kincsek elhelyezése az alábbi szabályok megtartásával történjen:

- *játékosok mindig csak a pálya szélén lévő határfalak melletti szabad területeken indulhassanak (külső gyűrű)*
- *kincsek mindig csak az ezen belül eső területeken lehessenek*
- *két játékos közötti távolság mindig legyen több mint 5 (légvonalban mérve)*
- *két játékos induláskor ne legyen ugyanabban a sorban vagy oszlopban (tehát ne láthassák egymást)*
- *két kincs közötti távolság mindig legyen több mint 2*
- *egy kincs és egy játékos közötti távolság mindig legyen több mint 2*

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **BacktrackElhelyezo**

A *BacktrackElhelyezo* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **ter** – egy **JatekTer** referencia, amely majd azt a játékteret mutatja, amelyiken belül az elemeket el kell helyeznünk.
- Új privát mező: **elemek** – egy *JatekElem* tömb, ami azokat az elemeket fogja tároli, amelyeket szét kell szórnunk.
- Új privát mező: **uresPoziciok** – egy kétdimenziós egész számokat tartalmazó tömb, a mérete csak később fog kiderülni.
- Új publikus konstruktor: **BacktrackElhelyezo** – paraméterként egy **JatekTer** objektumot kap. Egyrészt ezt eltárolja az erre szolgáló mezőbe. Ezt követően megszámolja, hogy a játéktérnek hány olyan pontja van, ami még üres (tehát nincs ott semmi), legyen ez az érték *db*. Ezt követően létrehozza az uresPoziciok tömböt db×2 dimenziókkal, és ebbe betölti a játéktér üres mezőinek X és Y koordinátáit. Ez a tömb mutatja majd, hogy hova lehet elhelyezni a kapott elemeket a játéktéren belül.
- Készítsük el a visszalépéses keresést, aminek a célja az *elemek* tömbben lévő játékosok és kincsek elhelyezése. Ezt az alábbiak szerint tehetjük meg:
  - Részfeladatok megfogalmazása: az *elemek* tömb minden egyes objektumának keresnünk kell egy helyet az *uresPoziciok* tömbben található üres helyek között úgy, hogy az megfeleljek a megadott szabályoknak (lásd ft, fk).
  - Részfeladatok száma: az *elemek* tömb mérete.
  - Lehetséges részmegoldások: minden részfeladat esetében az *uresPoziciok* tömb egy indexe, amelyik azt a helyet mutatja, ahová az adott elemet lerakhatjuk (tehát pl. ha a 3. részfeladat megoldása 5, akkor az *elemek[3]*-ban lévő elemet az *uresPoziciok[5]* helyre lehet lerakni).
  - Lehetséges részmegoldások száma: minden részfeladat esetében egyenlő az *uresPoziciok* tömb méretével

  Ehhez a megismert backtrack függvényeket az alábbiak szerint célszerű megvalósítani:
  - **Ft(szint, hely)** – azt ellenőrzi, hogy megadott elem lerakható-e megadott helyre. Tehát ha a *szint*-edik elem egy játékos, akkor csak a külső gyűrű üres pozícióiba helyezhető el, ha pedig kincs, akkor csak ezen belül.
  - **Fk(szint, hely, k, khely)** – azt ellenőrzi, hogy az ellenőrzéskor megadott két elem nem zárja-e ki egymást a megadott helyeken. Tehát attól függően, hogy két játékos/két kincs/egy kincs-egy játékos szerepel az összehasonlításban, figyelembe kell venni a kettejük közti távolságot, illetve két játékos esetén azt is, hogy ne láthassák egymást.
  - **Backtrack** – az előadáson megismert módon a fenti feltételekkel működő visszalépéses keresés megvalósítása
- Új publikus metódus: **Elhelyezés** – paraméterként egy *JatekElem* tömböt kap, amiben játékosok és kincsek lehetnek, amelyek induláskor mind a pálya -1,-1 koordinátájú pontjában vannak. Ezeket kell elhelyezni a pályán a fenti szabályok alapján. Hajtsa végre az alábbi feladatokat:
  - Mentse el a kapott tömböt az erre szolgáló mezőben.
  - Állítsa be a visszalépéses keresés szükséges paramétereit.
  - Indítsa el a visszalépéses keresést.
  - Amennyiben a visszalépéses keresés talált megoldást, akkor a kapott *JatekElem* tömb elemeinek a koordinátáit állítsuk át az eredménynek megfelelő helyekre.
  - Amennyiben nem talált megoldást, akkor dobjon egy **BackTrackNincsMegoldasException** kivételt (ehhez készítsük el a szükséges kivétel osztályt).

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **PalyaGeneralas** – töröljük az eddigi kincs elhelyezés részt
- Új konstans: **ELLENFELEK_SZAMA** – elhelyezendő ellenfelek száma (3)
  (PALYA_MERET_X=21, PALYA_MERET_Y=11, KINCSEK_SZAMA=7, ELLENFELEK_SZAMA=3)
- Meglévő metódus módosítása: **Futtatas** – töröljük az eddigi játékos létrehozást (és az ehhez kapcsolódó egyéb objektumok létrehozását, megjelenítés, generátor, stb.). Az új létrehozás az alábbiak szerint működjön:
  - Hozzon létre egy 1 + ELLENFELEK_SZAMA + KINCSEK_SZAMA méretű JatekElem tömböt
  - Ebbe helyezzen el egy emberi játékost, ELLENFELEK_SZAMA darab gépi játékost (50%-os eséllyel ez legyen gonosz), illetve KINCSEK_SZAMA darab kincset
  - Hozzon létre egy BacktrackElhelyező objektumot
  - Hívja meg ennek Elhelyezes metódusát, és adja át az előbbi tömböt, hogy el tudja helyezni az elemeket
  - Amennyiben az Elhelyezes metódus *BackTrackNincsMegoldasException* kivételt dobott, akkor véletlenszerűen töröljön ki egy belső fal objektumot a játéktérből, és hívja meg újra az elhelyezést.
  - Ha sikerült az elhelyezés, hozza létre a szükséges kiegészítő objektumokat (pl. órajelgenerátor ellenfelekre kapcsolása, megjelenítés bekapcsolása, stb.)
- Meglévő metódus módosítása: **Futtatas** – töröljük az eddigi játékos létrehozást (és az ehhez kapcsolódó egyéb objektumok létrehozását, megjelenítés, generátor, stb.)

Tesztelés: induláskor a szabályoknak megfelelő induló állapotot kell látnunk.

6. Óra – Láncolt lista (visszajátszás)

Egészítsük ki a játékot visszajátszási lehetőséggel. A fő ciklus leállítását követően lássuk a képernyőn az összes játékos megtett útját. Ehhez készítsünk egy speciális láncolt listát, ami az eltárolt tartalom mellett egy időbélyeget is tárol, hogy az illető adat mikor lett beszúrva.

OE.Prog2.Jatek.Automatizmus névtéren belül készítsük el az alábbiakat:

Új publikus interfész: **IVisszajatszhato**

Az *IVisszajatszhato* interfészt egészítsük ki az alábbiakkal:

- Új metódus: **Vegrehajt** – nincs se paramétere, se visszatérési értéke.

Új publikus osztály: **IdoFuggoLancoltLista<T>**, ahol T legyen *IVisszajatszhato*

Az *IdoFuggoLancoltLista* osztályon belül legyen egy beágyazott **ListaElem** osztály, ami az alábbi mezőket tartalmazza:

- Új publikus mező: **ido** – egy egész szám, ami a beszúrás idejét tárolja majd.
- Új publikus mező: **tartalom** – egy *T* típusú referencia, az eltárolni kívánt adat lesz.
- Új publikus mező: **kovetkezo** – egy *ListaElem* típusú referencia, ami a következő elemre mutat

Az *IdoFuggoLancoltLista* osztályt egészítsük ki az alábbiakkal is:

- Új felsorolás típus: **UzemmodTipus** – értékei: **Varakozas**, **Rogzites**, **Visszajatszas**
- Új privát mező: **uzemmod** – egy *UzemmodTipus* típusú változó, ez mutatja mindig, hogy a lista éppen melyik üzemmódban van. Kezdőértéke legyen *Varakozas.*
- Új privát mező: **ido** – egy egész szám, ami mindig a lista saját belső idejét mutatja (órajelciklusonként egyel növekszik majd).
- Új privát mező: **fej** – egy *ListaElem* típusú referencia, a lista első elemére hivatkozik (alapból *null*)

- Új privát mező: **utolsó** – egy *ListaElem* típusú referencia, a lista utolsó elemére hivatkozik (alapból *null* jelzi az üres listát).
- Új privát mező: **aktuális** – egy *ListaElem* típusú referencia, a lista lejátszásakor fogja mutatni az aktuális elemet
- Új publikus metódus: **RogzitesInditas** – egy paraméter és visszatérési érték nélküli metódus, ami az alábbi lépéseket hajtja végre: a lista üzemmódot rögzítésre állítja, nullázza a *fej*, *utolso* és *ido* mezők értékét.
- Új publikus metódus: **Rogzit** – ami paraméterként egy *T* típusú objektumot vár. Amennyiben a lista épp rögzítési üzemmódban van, akkor hozzon létre egy új *ListaElem*-et, állítsa be ennek a beszúrási idejét a lista *ido* mező aktuális értékének megfelelően, a tartalmat a paraméterként kapott tartalommal, majd vegye fel az elemet a láncolt lista végére. Mivel az idő mindig növekszik, így egy idő szerint rendezett listát fogunk kapni, ezt a lejátszáskor ki fogjuk használni.
- Új publikus metódus: **IdozitettBejaras** – egy paraméter és visszatérési érték nélküli metódus, ami az alábbi lépéseket hajtja végre: az *aktualis* mezőt állítsa a lista elejére, az időt nullára, majd a lista üzemmódot visszajátszásra. Magát a listán való végiglépkedést majd a *Mukodik* metódus fogja végrehajtani időzítetten.

Az *IdoFuggoLancoltLista* osztály valósítsa meg az *IAutomatikusanMukodo* interfészt:

- Új publikus tulajdonság: **MukodesIntervallum** – az interfész által igényelt tulajdonság mindig adjon vissza 1-et
- Új publikus metódus: **Mukodik** – az interfésznek megfelelően itt kell megvalósítanunk azt, hogy a lista aktualizálja a saját belső idejét, illetve visszajátszási módban hívja meg az ennél az időnél kisebb beszúrási idővel rendelkező elemek *Vegrehajt* metódusát. Ehhez minden *Mukodik* híváskor növelje az aktuális *ido* értéket egyel, majd visszajátszási üzemmód esetén végezze el az alábbiakat: az *aktualis* mutató elvileg mindig az utolsó, még nem visszajátszott elemre mutat a listában, ezért ezt léptessük addig, amíg nem lép a lista végére vagy olyan elemre, aminek a beszúrási ideje több mint a lista *ido* mező értéke. Az egyes továbblépésekkor mindig hívjuk meg az aktuális listaelemben lévő tartalom *Vegrehajt* metódusát. Amennyiben a lista minden elemét feldolgoztuk, a lista álljon vissza a *Varakozas* üzemmódra.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **MozgasAdatok**, ami megvalósítja az *IVisszajatszhato* interfészt.

A *MozgasAdatok* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **elem** – egy *MozgoJatekElem* típusú referencia, ez mutatja, hogy kinek a mozgását mutatja az elem.
- Új privát mező: **x** – egy egész szám, az új hely x koordinátája.
- Új privát mező: **y** – egy egész szám, az új hely y koordinátája.
- Új konstruktor: **MozgasAdatok** – beállítja a fenti három mező tartalmát a paraméterek alapján.
- Új metódus: **Vegrehajt** – a mezőben lévő *elem* objektumnak meghívja az *Athelyez* metódusát a mezőkben lévő koordinátákkal (tehát újra végrehajtja a mozgást).

A *Jatekos* osztályt egészítsük ki az alábbiakkal:

- Új privát mező: **felvevo** – egy időfüggő láncolt lista, amelyikben *MozgasAdatok* típusú elemeket lehet eltárolni.

- Új publikus metódus: **RogzitesInditas** – paraméterként egy *OrajelGenerator* objektumot vár. Létrehozza az előző láncolt listát, meghívja annak *RogzitesInditas* metódusát, majd rögzíti a játékos aktuális helyzetét. A paraméterként kapott órajelgenerátorra rákapcsolja a láncolt listát.
- Új publikus metódus: **VisszajatszasInditas** – Visszaállítja a játékos életerejét 100-ra és az *Aktiv* tulajdonságot igazra (különben nem mozogna a játékos a visszajátszáskor). Ezután meghívja a láncolt lista *IdozitettBejaras* metódusát.
- Meglévő metódus módosítása: **Megy** – a metódus végén a láncolt listába tároljuk el az új pozíciót.

OE.Prog2.Jatek.Keret névtéren belül készítsük el az alábbiakat:

A *Keret* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Futtatas** – végezzük ez az alábbi kiegészítéseket a visszajátszás érdekében:
  - A gépi játékosok elindítását követően egy hasonló ciklussal hívjuk meg az összes játékos *RogzitesInditas* metódusát, paraméterként átadva az órajelgenerátor objektumot.
  - A játék fő ciklusából való kilépéskor egy ciklussal vegyük le az összes gépi játékost az órajelgenerátorról (ezzel leállítva az aktuális mozgatásukat).
  - Majd egy újabb ciklussal hívjuk meg az összes játékos *VisszajatszasInditas* metódusát.
  - Ezt követően még várjunk egy billentyűleütést, hogy ne álljon le a program a visszajátszás ideje alatt.

Tesztelés: A játék végén automatikusan elindul a visszajátszás. A visszajátszás nem lesz tökéletes (mivel pl. a kincseket nem raktuk vissza a kezdő állapotba), de a játékosok mozgását látnunk kell kb. azonos sebességgel, mint ahogy az valóban megtörtént.

7. Óra – Bináris keresőfa (játékos memória)

Készítsünk a játékosoknak egy saját belső memóriát, hogy bizonyos elemek helyzetére akkor is emlékezzenek, ha azt éppen nem látják. Mivel ezt gyakran kell frissíteni, ezért használjunk egy speciális bináris keresőfát, amibe koordináták(ból képzett kulcs) alapján tudunk keresni.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus interfész: **IMemoriabanTarolhato**

Az *IMemoriabanTarolhato* interfészt egészítsük ki az alábbiakkal:

- Új tulajdonság: **X** – egész szám, visszaadja a tárolandó elem x koordinátáját.
- Új tulajdonság: **Y** – egész szám, visszaadja a tárolandó elem y koordinátáját.

Új delegált típus: **MegszunesKezelo** – egy paraméterrel rendelkezzen:

- Egy *IMemoriabanTarolhatoEsMegszuno* referencia, aki küldte az eseményt (lásd alább).

Új publikus interfész: **IMemoriabanTarolhatoEsMegszuno**

Az *IMemoriabanTarolhatoEsMegszuno* interfészt egészítsük ki az alábbiakkal:

- Új esemény: **Megszunes** – egy *MegszunesKezelo*-nek megfelelő metódussal lehessen feliratkozni rá.

Új publikus osztály: **MemoriaFa**

A *MemoriaFa* osztályon belül legyen egy beágyazott **FaElem** osztály, ami az alábbi mezőket tartalmazza:

- Új publikus mező: **kulcs** – egy egész szám, az elem azonosítója a fában. Ez a rendezés alapja is.
- Új publikus mező: **tartalom** – egy *IMemoriabanTarolhato* típusú referencia, az eltárolni kívánt adat lesz.
- Új publikus mező: **bal** – egy *FaElem* típusú referencia, ami a baloldali gyerekre mutat.
- Új publikus mező: **jobb** – egy *FaElem* típusú referencia, ami a jobboldali gyerekre mutat.

A *MemoriaFa* osztályt egészítsük ki az alábbiakkal is:

- Új privát mező: **gyoker** – egy *Faelem* típusú referencia, a fa gyökere.
- Új privát mező: **meret** – egy egész szám, a fában lévő elemek számát mutatja mindig.
- Új publikus metódus: **Beszuras** – paraméterként egy *IMemoriabanTarolhato* objektumot vár. Ennek koordinátái alapján kiszámolja a fabeli kulcsot (*X * 1000 + Y*), majd meghívja a rekurzív beszúrást. Az algoritmus megfelel az előadáson tanultaknak az alábbi kiegészítésekkel:
  - Egy új elem felvétele után növeljük a *meret* mező értékét.
  - Az új elem felvételét követően ellenőrizzük, hogy annak tartalma megvalósítja-e az *IMemoriabanTarolhatoEsMegszuno* interfészt is. Ha igen, akkor a fa iratkozzon fel a *Megszunes* eseményére a *Torles* metódussal (lásd alább).
  - Ha van már adott kulcsú elem, akkor ne történjen semmi (elvileg nem lesz egy helyen két tárolandó elem, de ha mégis, alapvetően az se lenne hiba, ilyenkor csak az egyikre fog emlékezni).
- Új publikus metódus: **Torles** – paraméterként egy *IMemoriabanTarolhato* objektumot vár. Ennek koordinátái alapján kiszámolja a fabeli kulcsot (*X * 1000 + Y*), majd meghívja a rekurzív törlést. Az algoritmus megfelel az előadáson tanultaknak az alábbi kiegészítésekkel:
  - Minden sikeres törlés után csökkentsük a *meret* mező értékét.
  - Ha a törlendő elemet megtaláltuk, és az megvalósítja az *IMemoriabanTarolhatoEsMegszuno* interfészt is, akkor a fa iratkozzon le erről.
  - Ha a fából hiányzik a törlendő elem, akkor dobjon egy új **FabolTorlesSikertelen** kivételt (ehhez készítsük el a szükséges kivétel osztályt).
- Új publikus metódus: **Bejaras** – visszatérési értéke egy tömb, ami tartalmazza az összes fában lévő *IMemoriabanTarolhato* elemet. Az algoritmus megfelel az előadáson tanultaknak, bármelyik bejárás megfelelő, a sorrend nem számít (a tömb mérete lehet a *meret* mező értéke).

A *Jatekos* osztályt egészítsük ki az alábbiakkal:

- Új védett mező: **memoria** – egy memória fa objektum, ami tárolni fogja a szükséges interfészt megvalósító elemeket.
- Meglévő metódus módosítása: **Megy** – minden mozgás után frissítsük a memória tartalmát. Ehhez a játékos nézzen körül, tehát töltse le az új pozíciójának 5 sugarú környezetében lévő játékelemeket. Ezek közül azokat, amelyek megvalósítják az *IMemoriabanTarolhato* interfészt, szúrja be a fába.
- Meglévő metódus módosítása: **MegjelenitendoElemek** – a megjelenítéskor a memória tartalmát is ki kell rajzolni. Ennek megfelelően a javasolt lépések:
  - A környező elemek lekérdezése egy tömbbe (már kész).
  - Memória tartalom lekérdezése egy tömbbe a bejárás használatával (új).
  - Környező elemek közül kirajzolhatók megszámolása (már kész)
  - Memóriabeli elemek közül a kirajzolhatók megszámolása (új)
  - Tömb létrehozás a két darabszám összegének megfelelő mérettel (módosítandó)

- Környező elemek közül a kirajzolhatók átmásolása az új tömbbe (már kész)
- Memóriabeli elemek közül a kirajzolhatók átmásolása az új tömbbe (új)
- Az új tömb visszaadása

A *Fal* osztály valósítsa meg az *IMemoriabanTarolhato* interfészt az alábbiak szerint: mind az *X* és *Y* megvalósítása történhet implicit módon a már meglévő tulajdonságokkal.

A *Kincs* osztály valósítsa meg az *IMemoriabanTarolhatoEsMegszuno* interfészt az alábbiak szerint: az *X* és *Y* megvalósítása megtörténhet implicit módon a már meglévő tulajdonságokkal. Továbbá:

- Új publikus esemény: **Megszunes** – legyen egy *MegszunesKezelo* típusú eseménykezelő.
- Meglévő metódus módosítása: **Utkozes** – amennyiben egy játékos felvette a kincset, és valaki feliratkozott a fenti eseménykezelőre, akkor küldjön egy *Megszunes* eseményt is a törlés előtt.

Tesztelés: A játékos saját nézetében látni kell mindazokat a falakat és kincseket, amelyekkel már legalább egyszer találkozott. A kincseknek pedig azonnal el kell tűnniük, ha valaki felvette azokat (még akkor is, ha épp nem látjuk azt, aki felvette). Érdemes megnézni egy gépi játékosra kapcsolt nézetet, hasonlóan kell működnie (pl. az általunk felvett kincseknek nála is el kell tűnnie)

8. Óra – Gráf (ellenség okosítása útkereséssel)

A gépi játékosok mozgatását egészítsük ki azzal, hogy ha van egy kincs a játékos memóriájában, akkor keressen egy ahhoz vezető utat, és azon induljon el. Mivel menet közben folyamatosan változhat a környezet, így minden lépés előtt futtassuk le ezt a keresést, és ez mindig csak egy lépésnyi ajánlást adjon.

OE.Prog2.Jatek.Szabalyok névtéren belül készítsük el az alábbiakat:

Új publikus osztály: **KincskeresoGraf**

A *KincskeresoGraf* osztályon belül legyen egy beágyazott **GrafPont** osztály, amely a játéktér egy szabad mezőjét képviseli. Ez az alábbiakat tartalmazza:

- Új publikus mező: **x** – a gráfpont által képviselt mező x koordinátája.
- Új publikus mező: **y** – a gráfpont által képviselt mező y koordinátája.
- Új publikus mező: **elozo** – egy *GrafPont* típusú referencia, ami azt mutatja, hogy melyik másik ponton keresztül értük el ezt a mezőt a bejárás során.

A *KincskeresoGraf* osztályt egészítsük ki az alábbiakkal is:

- Új privát konstans mező: **LEPES_KORLAT** – egy egész szám, értéke legyen 1000. Ez az érték fogja korlátozni az útkeresést, hogy az ne tartson túl sokáig (és ne kerüljön végtelen ciklusba).
- Új privát mező: **memoria** – egy *IMemoriabanTarolhato* típusú elemeket tartalmazó tömb, mérete csak később derül majd ki.
- Új publikus konstruktor: **KincskeresoGraf** – beállítja a fenti mező tartalmát a paraméterként átadott memória tömb értékével.
- Új privát metódus: **SzabadSzomszed** – egy *GrafPont*-ot vár paraméterként, és egy listában visszaadja ennek szabad szomszédait (*GrafPont* objektumokként). Szabad szomszéd alatt a 4 szomszédos mező közül azokat értjük, amelyekben a memória alapján nincs fal (kincs lehet, arra ráléphetünk, más pedig nem lesz a memóriában). A visszaadott *GrafPont* objektumok *elozo* mezőjének értéke legyen a paraméterként átadott kiindulómező.

- Új privát metódus: **CélPont** – egy *GrafPont*-ot vár paraméterként, visszatérési értéke pedig az, hogy érdemes-e idejönnie a gépi játékosnak. Amennyiben a memória szerint egy kincs van ezen a helyen, akkor igen, különben nem.
- Új publikus metódus: **JavasoltIrany** – egy x és y koordinátapárt vár paraméterként, amelyek a játékos jelenlegi helyzetét adják meg. Visszatérési értéke pedig egy kételemű egész tömb, ami a javasolt lépés irányát mutatja (relatív x és y koordináták).
  Készítsünk egy egyszerű szélességi bejárást a legközelebbi kincs, és a hozzá vezető út meghatározására. A gráf csúcsai a pálya nem falat tartalmazó mezői, az utak pedig az egymás melletti mezőket kötik össze. A gráfot a memória aktuális tartalma alapján képzeljük el. A szélességi bejárás nem igényli, hogy előre létrehozzuk a teljes gráfot, elég ha a kezdőpontból kiindulva fokozatosan térképezzük fel (nem is tudnánk létrehozni a gráfot, hiszen amíg a játékos nem látja a határoló falakat, addig számára egy végtelen térnek tűnik a játékpálya):
  - hozzunk létre egy **erintett** nevű, *GrafPont* objektumokat tartalmazó listát (*List*), ez tartalmazza majd azokat a csúcsokat, amelyekkel már kapcsolatba kerültünk (feldolgoztuk őket, vagy feldolgozásra várnak).
  - hozzunk létre egy **varakozo** nevű, *GrafPont* objektumokat tartalmazó sort (*Queue*), ami a fedolgozásra váró csúcsokat tartalmazza majd.
  - hozzunk létre egy *GrafPont* objektumot, ami az induló koordinátákat tárolja, az *elozo* mezőjének értéke pedig legyen *null.* Ezt tegyük be az *erintett* listába, és a *varakozo* sorba.
  - indítsunk egy ciklust, ami addig fut, amíg van elem a *varakozó* sorban.
    - vegyük ki a következő elemet a sorból.
    - ha ez az elem egy célpont (kincs van itt), akkor a *GrafPont* objektum *elozo* mezőjén keresztül látjuk, hogy hol jutottunk ide. Folyamatosan kövessük ezeket az előző mezőket, amíg eljutunk a kiinduló ponttól való első lépésig. Ennek a kiindulóponttól való relatív koordinátáit adjuk vissza a függvény visszatérési értékeként (ezzel kiugorva a ciklusból).
    - ha nem célpont, akkor kérdezzük le a szabad szomszédok listáját. ha ezek között van még olyan, amelyiket még nem érintettük, és az *erintett* lista mérete még nem érte el a *LEPES_KORLAT* értékét, akkor ezt a pontot tegyük bele az *erintett* listába és a *varakozo* sorba is.
  - ha a ciklus végetér, akkor dobjunk egy **NincsKincshezVezetoUt** kivételt (hozzuk létre az ehhez szükséges osztályt is).

A *GepiJatekos* osztályt egészítsük ki az alábbiakkal:

- Meglévő metódus módosítása: **Mozgas** – hozzunk létre egy *KincskeresoGraf* objektumot, aminek átadjuk a gépi játékos memóriájának aktuális tartalmát. Hívjuk meg ennek *JavasoltIrany* metódusát, majd ezekkel a relatív koordinátákkal hívjuk meg a *Megy* metódust. Ha eközben *NincsKincshezVezetoUt* vagy *MozgasHelyHianyMiattNemSikerult* kivétel keletkezett, ezeket kapjuk el, és próbálkozzunk a hagyományos módon (mind a négy irányt véletlenszerűen végigpróbálgatva).

Tesztelés: a gépi játékosok, ha meglátnak egy kincset, akkor elkezdenek afelé mozogni. Mivel nem látják át a teljes pályát, így elképzelhető, hogy néha rossz irányba indulnak, de a mozgás közben amint észreveszik, hogy zsákutcába kerültek, vagy valaki felvette a megcélzott kincset, azonnal irányt kell váltaniuk. Ha nem látnak kincset, akkor a megszokott módon véletlenszerűen kezdenek el mozogni.

Disclaimer: mivel az órajelgenerátor és a fő ciklus, amely a billentyű leütéseket figyeli, különböző szálakon futnak, ez bizonyos esetekben hibákat okozhat (pl. a JatekTer osztály MegadottHelyenlevok metódusában, ha az elemek megszámolása és a tömbbe való kigyűjtése közötti időben a másik szálon valaki éppen felvesz egy új elemet a játéktérbe, akkor az a tömb túlcímzéséhez vezet). Ezek az esetek természetesen kezelhetők, azonban az ehhez szükséges technikák túlmutatnak a Programozás 2 tárgy keretein. A leírás részletei (pl. bizonyos műveletek javasolt sorrendje, néhány feleslegesnek tűnő lépés) észrevétlenül próbálja minimalizálni a fentiek bekövetkezési valószínűségét, a hibátlan megoldáshoz vezető további kerülőutak bevezetése azonban már feleslegesen bonyolítaná a program szerkezetét. Ezért a fentiek tökéletes megvalósítása mellett is előfordulhatnak meglehetősen ritka (nagy valószínűséggel soha nem tapasztalható) futási hibák.

Ha valakit (egyéként nagyon helyesen) mégis zavar a hibák elvi lehetősége, próbálja meg szálbiztossá tenni az adatszerkezeteket, vagy a billentyű leütéseket figyelő kódot a főprogramban futó végtelen ciklus helyett írja át az órajelgenerátorra kapcsolható metódussá.