

## Feladateleírás

A Doom 1993-ban jelent meg, és alapjaiban változtatta meg a videojátékok világát: grafikája és játékmenete az akkori technológiai szinten lenyűgözően részletgazdag volt. A játékban egy űrgárdista, *Doomguy* szerepét veheted át, aki a Mars egyik holdján csapdába esik és démonok seregeivel kell megküzdenie. Több mint 30 év távlatából is inspirálja a megjelenő játékokat, és alapvető hatással volt arra, amit ma az akció- és lövöldözős játékok nyújtanak.

A leírás alapján egy Doom-inspirálta, felülnézetes, kétdimenziós, karakter-alapú játékot készíthetsz el. A játékban *Doomguy*-t kell kijuttatnod a sötét, démonokkal teli folyosókon keresztül a pálya kijáratáig, miközben megküszdesz az utadba kerülő ellenségekkel. A játék elkészítéséhez többségében olyan ismeretekre lesz szükséged, amelyet a Problémamegoldás programozással tárgy keretében már elsajátítottál. Néhány olyan probléma esetén, amely túlmutat a tárgy anyagán (pl. hanghatások lejátszása), mutatunk egy lehetséges megoldást, amelyet elegendő a saját forráskódodba illeszteni.

A játék természetesen nagyon sok tekintetben továbbfejleszthető, a kód felépítése pedig sok esetben jelentősen egyszerűsíthető vagy javítható a további félévekben tanult objektumorientált alapelvek (öröklés és polimorfizmus) és koncepciók (interfészek, eseménykezelés) alkalmazásával.

## 1. Kezdeti lépések

### A játékban szereplő objektumok pozíciója

A játék minden objektumának (játékos, démonok, tárgyak) helyét egy kétdimenziós koordinátarendszerben adjuk meg. Hozd létre a `Position` nevű osztályt.

- Az koordináta  $x$  és  $y$  komponensét tároljuk egy-egy egész típusú mezőben, az értékek legyenek publikusan lekérdezhetők és módosíthatók az `X` és `Y` tulajdonságokon keresztül.
- A mezők kezdőértéke a konstruktoron keresztül adható meg.
- Legyen egy statikus `Add` nevű metódusa, amely két `Position` típusú objektumot, `p_1`-et és `p_2`-t vár paraméterként, és egy új `Position` objektumot ad vissza, amelynek koordinátái a `p_1` és `p_2` objektumok  $x$  és  $y$  koordinátáinak összegéből adódnak.
- Legyen egy statikus `Distance` metódusa, amely két `Position` típusú objektumot, `p_1`-et és `p_2`-t vár paraméterként, és visszaadja kettőjük (euklideszi) távolságát egy lebegőpontos számként.

#### Tesztelés

Hozz létre két `Position` példányt, majd állítsd elő az összegüket, illetve számítsd ki a két pozíció távolságát. Ellenőrizd, hogy helyes eredményeket kaptál-e.

### A játékban szereplő objektumok megjelenése

A játékban szereplő objektumok megjelenítésekor kirajzolt egyszerű ábrákat (színes karaktereket) *sprite*-nak fogjuk nevezni. Hozd létre a `ConsoleSprite` nevű osztályt.

Egy sprite objektum esetén tároljuk a háttér színét (**Background**), az előtér színét (**Foreground**), illetve a kirajzolandó karaktert (**Glyph**). Mivel a játék a megjelenítéshez a parancssort fogja használni, így a színek leírására használjuk a **ConsoleColor** típust, a kirajzolandó karakter pedig legyen **char**. Mindhárom érték legyen publikusan lekérdezhető, a beállításuk pedig a konstruktor hívásakor történjen meg az átadott argumentumok alapján.

#### Tesztelés

Hozz létre egy **ConsoleSprite** példányt. Írd ki a képernyőre a példány **Glyph** jellemzőjét, de előtte állítsd be a kiíráshoz használt előtér- és háttérszíneket a **Foreground** és **Background** értékeknek megfelelően.

## A játékos

Doomguy jellemzőit egy *állapot-objektumban* fogjuk tárolni. Hozd létre a **Player** osztályt.

- A játékos aktuális pozícióját tároljuk egy **Position** típusú mezőben, amely legyen publikusan lekérdezhető és módosítható egy **Position** nevű tulajdonságon keresztül.
- A játékos kinézetét tároljuk egy **ConsoleSprite** típusú mezőben, amely szintén legyen publikusan lekérdezhető egy **Sprite** nevű tulajdonságon keresztül.
- Az osztály konstruktora egy  $x$  és egy  $y$  egész értéket vár, amely alapján létrehoz egy új pozíciót, amelyet beállít a játékos aktuális helyzetének. A játékos megjelenítéséhez használt sprite háttere legyen fekete, előtere zöld, az alakzat pedig egy 'O' karakter.

#### Tesztelés

Hozz létre egy **Player** példányt. A **ConsoleSprite** tesztjénél leírthoz hasonló módon jelenítsd meg a játékos sprite-ját a **Position**-nek megfelelő helyen a képernyőn.

## A játékmotor

A játék megjelenítéséért (renderelés), a játékos és a démonok mozgásáért és interakcióiért (játéklogika és fizikai motor), valamint hanghatásokért, a pályák betöltésért a játékmotor felel. Hozd létre a **Game** nevű osztályt.

- Tároljuk a játékos jellemzőit leíró objektumot egy **Player** típusú mezőben.
- Tároljuk a játék aktuális állapotát (fut vagy véget ért) egy logikai típusú mezőben, amely legyen publikusan lekérdezhető és módosítható az **Exited** tulajdonságon keresztül.
- A konstruktor hozza létre a játékos objektumot a (0,0) pozícióban.
- A **RenderSingleSprite** nevű privát metódus egyetlen sprite-ot rajzol ki a megadott pozícióra. A metódus egy **Position** és egy **ConsoleSprite** típusú paramétert vár. Ellenőrzi, hogy a pozíció az aktuális képernyőn belül van-e (ehhez használjuk a **Console.WindowWidth** és **WindowHeight** tulajdonságait), és ha igen, akkor erre a pozícióra kiírja a sprite-nak megfelelő színekkel a benne eltárolt karaktert.
- A **RenderGame** nevű privát metódus kirajzolja a játék aktuális állapotát a parancssorra. Előbb a kurzor láthatóságát (**CursorVisible**) hamisra állítja, alapértékre állítja a parancssori színeket (**ResetColor**), törli a képernyőt, majd a `codemethodRenderSingleSprite` meghívásával kirajzoltatja a játékos aktuális pozíciójára a játékos sprite-ját.
- A **UserAction** nevű privát metódus a felhasználói bemenetet (billentyű leütések) fogja kezelni. Elsőként ellenőrzi, van-e lenyomott billentyű (**Console.KeyAvailable**), majd igaz válasz esetén elvégzi a lenyomott billentyűtől függően a szükséges módosításokat (lásd az alábbi kódrészletet).

```

if (Console.KeyAvailable)
{
    ConsoleKeyInfo pressed = Console.ReadKey(true);
    switch (pressed.Key)
    {
        case ConsoleKey.Escape:
            exited = true;
            break;

        // ...
    }
}

```

Legyen lehetőség a játékos mozgatására a nyílbillentyűk lenyomásával (ilyenkor egyszerűen módosítsuk a játékos  $x$  vagy  $y$  koordinátáját), az Escape billentyű leütésekor pedig a játék érjen véget (az **Exited** tulajdonság igaz értékre állításával).

- Hozzunk létre egy publikus **Run** nevű metódust, amelynek meghívásával indítható a játék. A metódusban helyezzünk el a **while** ciklust, ami az alábbi lépéseket ismétli mindaddig, amíg a játék **Exited** tulajdonsága hamis értéket ad:
  1. Jelenítse meg a játék aktuális állapotát a **RenderGame** meghívásával.
  2. Kezelje a felhasználói bemenetet a **UserAction** meghívásával.
  3. A `Thread.Sleep(25);` utasítással várakoztassa a végrehajtást 25 ms-ig, így jelenleg nagyjából 40 FPS képkockasebességet kapunk a játékban. A konkrét érték beállításával a parancssor folyamatos törlése és újrarajzolása miatti "villogást" tudjuk valamelyest csökkenteni (opcionális).

## Tesztelés

Hozz létre egy **Game** példányt, majd hívd meg a **Run** metódusát. A képernyőn megjelenik *Doomguy*, a **ConsoleSprite** tesztjénél leírthoz hasonló módon jelenítsd meg a játékos sprite-ját a **Position**-nek megfelelő helyen a képernyőn.

## 2. Játékelemek

### A játékelemek típusai és jellemzőik









A játékban megjelenő statikus objektumokat *játékelemeknek* fogjuk nevezni. A játékos és a démonok előre megadott szabályok szerint interakcióba léphetnek a játékelemekkel, amelyek ennek hatására megváltoztathatják a játékos vagy a démon állapotát. Hozd létre a **GameItem** osztályt.

- Tároljuk a játékelem példány pozícióját egy **Position** típusú mezőben, amelynek értéke legyen publikusan lekérdezhető egy **Position** nevű tulajdonságon keresztül.
- Tároljuk a játékelem példányhoz tartozó sprite-ot egy **ConsoleSprite** típusú mezőben, amely szintén legyen publikusan lekérdezhető egy **Sprite** nevű tulajdonságon keresztül.
- Definiáljuk a játékelemek típusát egy **ItemType** felsorolással. A játékelemek lehetséges típusait foglalja össze az 1. táblázat. A pontos működésüket a leírás további részében adjuk meg.

```

enum ItemType { Ammo, BFGCell, Door, LevelExit, Medikit, ToxicWaste, Wall }

```

Megnevezés	Rövid leírás	Kitöltési tényező	Sprite
Ammo	Lőszer csomag, ami 5 lövedéket tartalmaz.	0.0	
BFGCell	Lőszer a BFG-hez, ami egy BFG lövedéket tartalmaz.	0.0	
Door	Ajtó, ami lehet zárva vagy nyitva.	1.0 vagy 0.0	 vagy 
LevelExit	Kijárat a pályáról.	1.0	
Medikit	Elsősegély csomag, ami 25 életerőt tartalmaz.	0.0	
ToxicWaste	Mérgező hulladék, csökkenti az életerőt.	0.0	
Wall	Fal, áthatolhatatlan akadály.	1.0	

1. táblázat. Játékelemek és jellemzőik.

- Tároljuk a játékelem példány típusát egy `ItemType` típusú mezőben, amelynek értéke legyen publikusan lekérdezhető egy `Type` nevű tulajdonságon keresztül.
- Minden játékelem a pozíciójához tartozó térrészt (csempét) bizonyos arányban tölti ki, ezt a kitöltési tényezőt használjuk majd az ütközések vizsgálatakor (például nem léphetünk majd olyan csempére, amelyen egy 1.0 kitöltési tényezőjű játékelem van). Tároljuk a játékelem példány kitöltési tényezőjét egy `double` típusú mezőben, amelynek értéke legyen publikusan lekérdezhető egy `FillingRatio` nevű tulajdonságon keresztül.
- Bizonyos játékelemek (pl. lőszercsomag) a játék során felhasználódhatnak, ami miatt eltűnnek a pályáról. Tároljuk a játékelem példány elérhetőségének állapotát egy logikai típusú mezőben, amelynek értéke legyen publikusan lekérdezhető egy `Available` nevű tulajdonságon keresztül.
- A `SetInitialProperties` nevű paraméter nélküli privát metódus a játékelem típusától függően elvégzi a játékelem sprite-jának és kitöltési tényezőjének beállítását. Használjuk az 1. táblázatban szereplő adatokat.
- Az osztály konstruktora egy  $x$  és  $y$  értéket, valamint egy játékelem-típust vár paraméterként, ezek alapján beállítja a játékelem pozícióját és típusát. A játékelem kezdetben legyen elérhető (lásd az `Available` tulajdonságot). Hívjuk meg a `SetInitialProperties` metódust a típustól függő kezdeti jellemzők beállításához.
- Az `Interact` nevű publikus metódus a játékelemek állapotváltozását kezeli olyan esetekben, amikor a játékos (vagy egy démon) interakcióba lép az elemmel.
  - Ha a játékelem típusa `Ammo`, `BFGCell` vagy `Medikit`, akkor az interakció hatására a játékelem elérhetősége változzon hamisra (ilyenkor a játékos felszedte az adott játékelemet).
  - Ha a játékelem típusa `Door`, akkor az interakció hatására a játékelem kitöltési tényezője változzon 1.0-ről 0.0-ra (ilyenkor az ajtó kinyílt), vagy 0.0-ról 1.0-re (ilyenkor az ajtó bezárult). A kitöltési tényező módosítása mellett rendeljünk új sprite-ot is a játékelemhez (változtassuk meg az elem színét az 1. táblázatban látható módon).

## Játékelemek tárolása, megjelenítése és törlése

Bővítsük a `Game` osztályt az alábbiak szerint.

- Tároljuk az elérhető játékelemeket egy `Items` nevű listában. Az üres listát a `Game` konstruktorában hozzuk

létre.

- Módosítsuk a **RenderGame** metódust úgy, hogy a játékos kirajzolása előtt egy ciklussal bejárja az **Items** listát, és annak minden elemét rajzolja ki a **RenderSingleSprite** metódus hívásával.
- Hozzunk létre egy **CleanUpGameItems** nevű privát metódust, amely törli az **Items** listából azokat az elemeket, amelyek már nem elérhetők (**Available** tulajdonság). Ügyeljünk rá, hogy a lista bejárása közben történő törlés hibát okozhat. Egy lehetséges megoldási stratégia: előbb gyűjtsük le a törlendő elemeket egy külön listába, majd a leggyűjtött elemek listáját bejárva töröljük azokat az **Items**-ből.

## Tesztelés

Kizárólag a tesztelés idejére végezd el az alábbi módosításokat a **Game** osztályon belül.

- Tedd publikussá az **Items** listát.
- A **UserAction** metódusban kezeld le a D billentyű lenyomását is. Ennek hatására járd be az **Items** listát, és minden elemére hívd meg az **Interact** metódust.

A főprogramban adj az **Items** listához egy Door, egy Medikit, egy ToxicWaste és egy Wall játékelemet még a **Run** meghívása előtt. A játék elindítása után a játékosra rá tudsz lépni a játékelemekkel kitöltött csempékre (ilyenkor a játékos sprite-ja eltakarja a játékelemét). A D billentyű lenyomásakor az ajtó állapota megváltozik, a felszedhető játékelem pedig eltűnik.

## 3. Ütközésvizsgálat

### A játékos kitöltési tényezője

A **Player** osztályban vegyünk fel egy **double** típusú mezőt, amely a játékos kitöltési tényezőjét adja meg, hasonlóan a játékelemeknél leírtakhoz. A mező értéke legyen publikusan lekérdezhető egy **FillingRatio** nevű tulajdonságon keresztül. A játékos kitöltési tényezőjét állítsuk be a **Player** konstruktorában 0.5-re.

### Kitöltési tényezőtől függő mozgás

A játékban egy leegyszerűsített ütközésvizsgálatot fogunk megvalósítani úgy, hogy minden nem statikus játékbjektum (játékos és démonok) mozgása előtt ellenőrizzük, hogy a lépés nem vezet-e ütközéshez.

Bővítsük a **Game** osztályt az alábbiak szerint.

- Hozzunk létre egy **GetGameItemsWithinDistance** nevű privát metódust, amely paraméterként egy **Position** értéket és egy **double** távolságot vár. A metódus gyűjtse le az **Items** listából azokat a játékelemeket, amelyek nincsenek messzebb a paraméterként átadott távolságnál a paraméterként átadott pozícióhoz viszonyítva. A pozíciók távolságának számításához használjuk a **Position** osztály **Distance** statikus metódusát.
- Hozzunk létre egy **GetTotalFillingRatio** nevű privát metódust, amely paraméterként egy **Position** értéket vár. A metódus előbb meghatározza az adott pozícióban lévő játékelemeket a **GetGameItemsWithinDistance** metódus hívásával (az átadott távolság értéke legyen 0), majd kiszámítja és visszaadja a leggyűjtött játékelemek kitöltési tényezőinek összegét.
- Hozzunk létre egy **Move** nevű privát metódust, amely paraméterként egy **Player** példányt és egy **Position** értéket vár, az utóbbi a cél pozíció, ide szeretne elmozdulni a játékos. Határozzuk meg az átadott pozícióban lévő játékelemek összesített kitöltési tényezőinek (**GetTotalFillingRatio**) és a játékos kitöltési tényezőjének összegét. Ha ez az összeg nem nagyobb 1.0-nál, akkor állítsuk be a játékosnak az átadott cél pozíciót, egyébként ne történjen semmi. Megjegyzés: a játékos paraméterként felsorolása feleslegesnek tűnik, azon-

ban ezzel egységesebb alakúak lesznek a mozgatót végző metódusok, amikor a démonokat is beépítjük a játékba.

Módosítsuk a **Game** osztályban a **UserAction** metódust úgy, hogy a mozgáshoz rendelt billentyűk lenyomásakor kiszámítjuk a megfelelő cél pozíciót (ehhez használjuk a **Position** osztály **Add** statikus metódusát), majd a játékoskal és a kiszámított cél pozícióval hívjuk meg a **Move** metódust.

### Tesztelés

Az előbbi tesztet újra lefuttatva már nem tudunk a falat vagy zárt ajtót tartalmazó pozíciókra lépni. Az ajtó nyitása után viszont az oda történő mozgás megengedett.

*Folytatása következik.*