**bevezető**

Témaválasztás indoklása, annak elméleti és gyakorlati jelentősége. Célkitűzések és eredmények megfogalmazása. Használt technológiák, szoftverek felsorolása.

**tárgyalási rész**

Használt technológiák, szoftverek bemutatása.

* C#
* Visual Studio
* .Net
* Unity
  + Unity Editor
  + Unity néhány fontosabb ’fogalom’
  + Unity Common Components
* játék bemutatása(milyen részletességgel?)

**összefoglalás**

Megállapítások és következtetések, tapasztalatok.

**jatekrol par szo**

Célom nem az volt, hogy egy tökéletes, toplistákat vezető játékot készítsek, csupán, hogy bemutassam a Unity-ben rejlő lehetőségeket. A Játéknak nincsen mély története. Egy fantasy világban kapunk egy küldetést, melynek teljesítése a játék célját és végét jelenti.

Kezdéskor három kaszt közül választhatunk karaktert: harcos, mágus, vadász. Ezeknek különböző statisztikáik vannak, például a vadász gyorsabban mozog a harcosnál, de kevesebb életereje van.

Az ellenfelek legyőzése után tapasztalati pontokat és játékbeli fizetőeszközt kapunk, előbbit képességpontjaink növelésére, utóbbit egy NPC kereskedőnél lehet elkölteni.

A karaktert a billentyűzettel, az egyes menügombokat és inventoryt pedig az egérrel lehet vezérelni.

Az asset összefoglaló neve azoknak az eszközöknek, melyek szerepet kapnak a játékban, például: a karakter sprite-ok (képek), vagy az egyes hangeffektusok.

A játékban szereplő asseteket a Unity Asset store-ból szereztem be.

|  |  |  |  |
| --- | --- | --- | --- |
| Sprite-ok és animációk | | Hangok | |
| Danil Chernyaev: 2D Platformer Tileset | [2D Platformer Tileset | 2D Environments | Unity Asset Store](https://assetstore.unity.com/packages/2d/environments/2d-platformer-tileset-173155) | MGWSoundDesign: Footstep(Snow and Grass) | [Footstep(Snow and Grass) | Audio Sound FX | Unity Asset Store](https://assetstore.unity.com/packages/audio/sound-fx/footstep-snow-and-grass-90678) |
| Black Hamme:  Fantasy Wooden GUI : Free | [Fantasy Wooden GUI : Free | 2D GUI | Unity Asset Store](https://assetstore.unity.com/packages/2d/gui/fantasy-wooden-gui-free-103811) | VGcomposer: Action RPG Music Free | [Action RPG Music Free | Audio Music | Unity Asset Store](https://assetstore.unity.com/packages/audio/music/action-rpg-music-free-85434) |
| PONETI: GUI Parts | [GUI Parts | 2D Icons | Unity Asset Store](https://assetstore.unity.com/packages/2d/gui/icons/gui-parts-159068) | Dustyroom: FREE Casual Game SFX Pack | [FREE Casual Game SFX Pack | Audio Sound FX | Unity Asset Store](https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116) |

**C# programozási nyelv:**

…

[1. Introducing C# - Programming C# 8.0 [Book] (oreilly.com)](https://www.oreilly.com/library/view/programming-c-80/9781492056805/ch01.html)

I.1. A nyelv története

A C# programozási nyelv 2002-ben a Microsoft új fejlesztési környezete, a Visual Studio.NET programcsomag részeként jelent meg.

[C# könyv (elte.hu)](https://people.inf.elte.hu/szlavi/Magamnak/Csharp/Programozas_Csahrp_nyelven_IZ.pdf)

A Microsoft azért kényszerült a kifejlesztésére, mert a 90-es években beperelte a Java nyelv licenceit birtokló Sun Microsystems, és a Java nyelv eltávolítását kényszerítették ki a Windows rendszerekből. A per vádja az volt, hogy a Microsoft saját Java keretkörnyezetét a saját operációsrendszer-specifikus függvényeivel és szolgáltatásaival bővítette ki, így az abban fejlesztett alkalmazások nem lettek volna futtathatóak más rendszereken. Tehát sértették a Java platform-függetlenségre vonatkozó alapelvét.

[SUN MICROSYSTEMS VS. MICROSOFT - Chicago Tribune](https://www.chicagotribune.com/news/ct-xpm-1997-10-08-9710080149-story.html)

A Microsoft a C++ nyelvet vette alapul a C# kifejlesztése során. Olyan nyelvet igyekeztek létrehozni, mely megtartja a C, C++ nyelvek hatékonyságát, ugyanakkor kiküszöböli a komplexitását, hosszú fejlesztési idejét.

I.2. A nyelv jellemzői

A C# egy modern objektumorientált, komponens orientált és típusbiztonságos programozási nyelv. Kényelmes és gyors lehetőséget biztosítva ahhoz, hogy .NET keretrendszer alá alkalmazásokat készítsünk, legyen az akár számolás, akár kommunikációs alkalmazás.

Az új .NET keretrendszer bázisnyelve. Tipikusan ehhez a keretrendszerhez tervezték, nem véletlen, hogy a szabványosítási azonosítójuk is csak egy számmal tér el egymástól. A nyelv teljesen komponens orientált. A fejlesztők számára a C++ hatékonyságát, és a Visual Basic fejlesztés gyorsaságát, egyszerűségét ötvözték ebben az eszközben

[C# könyv (elte.hu)](https://people.inf.elte.hu/szlavi/Magamnak/Csharp/Programozas_Csahrp_nyelven_IZ.pdf)

**.NET keretrendszer (átírás alatt)**

A szoftverfejlesztők .NET-nyelveken, például C# vagy F# nyelven írják meg a forráskódot. Minden megírt kódsor egy utasítást vagy parancsot fejez ki, amelyet a számítógépnek kell végrehajtania a program futásakor.

Ahhoz, hogy a szoftverfejlesztők futtatni tudják a kódjukat, először le kell fordítaniuk azt. A *.NET-fordító* egy program, amely a forráskódot egy speciális, úgynevezett *köztes nyelvre* (IL) konvertálja. A .NET-fordító az IL-kódot egy *.NET-szerelvénynek* nevezett fájlba menti. A kód „köztes” formátumra fordításának köszönhető, hogy ugyanaz a kód bárhol futtatható, Windows vagy Linux rendszeren, és 32 bites vagy 64 bites számítógép-hardveren egyaránt.

A *.NET-futtatókörnyezet* a lefordított .NET-szerelvény végrehajtási környezete. Ez azt jelenti, hogy a gazda operációs rendszeren futó alkalmazást a .NET-futtatókörnyezet hajtja végre és kezeli. A .NET-futtatókörnyezet szerepéről hamarosan bővebben is szó lesz.

Szerencsére a .NET-tel csak most ismerkedő szoftverfejlesztőknek eleinte nem muszáj teljesen megérteniük ezeknek a mechanizmusoknak a működését. Ahogyan gyakorlatra tesz szert, egyre jobban megérti és értékeli majd a .NET-nyelvek és -fordítók, és a futtatókörnyezet hatékonyságát.

C# programs run on .NET, a virtual execution system called the common language runtime (CLR) and a set of class libraries. The CLR is the implementation by Microsoft of the common language infrastructure (CLI), an international standard. The CLI is the basis for creating execution and development environments in which languages and libraries work together seamlessly.

.NET encompasses both the runtime and the main class library that C# programs use. The runtime part is called the Common Language Runtime (usually abbreviated to CLR) because it supports not just C#, but any .NET language. Microsoft also offers Visual Basic, F#, and .NET extensions for C++, for example. The CLR has a Common Type System (CTS) that enables code from multiple languages to interoperate freely, which means that .NET libraries can normally be used from any .NET language—F# can consume libraries written in C#, C# can use Visual Basic libraries, and so on.

[1. Introducing C# - Programming C# 8.0 [Book] (oreilly.com)](https://www.oreilly.com/library/view/programming-c-80/9781492056805/ch01.html)

[Mi az a .NET? - Learn | Microsoft Docs](https://docs.microsoft.com/hu-hu/learn/modules/dotnet-introduction/2-what-is-dotnet) / [What is .NET? - Learn | Microsoft Docs](https://docs.microsoft.com/en-us/learn/modules/dotnet-introduction/2-what-is-dotnet)

## Nyílt forráskód

A .NET egy ingyenes, nyílt forráskódú fejlesztő platform, különböző típusú alkalmazások fejlesztésére.

## Platformfüggetlen

A .NET az alábbi operációs rendszerekre történő alkalmazásfejlesztést támogatja:

* Windows
* macOS
* Linux
* Android
* iOS
* tvOS
* watchOS

Támogatott processzor architektúrák:

* x64
* x86
* ARM32
* ARM64

### **Támogatott Programozási nyelvek**

.NET kezdetben három programozási nyelvet támogatott, integrált fejlesztői környezeteket (IDEs), és egyéb eszközöket biztosít.

Ezek a programnyelvek az alábbiak:

* [C#](https://docs.microsoft.com/en-us/dotnet/csharp/)

Amelyről már korábban szót ejtettem.

* [F#](https://docs.microsoft.com/en-us/dotnet/fsharp/)

Az F# funkcionális, objektum-orientált és imperatív programozási modelleket támogat.

* [Visual Basic](https://docs.microsoft.com/en-us/dotnet/visual-basic/)

A három programnyelv közül, ennek a szintaktikája áll a legközelebb az általános emberi nyelvhez. Ez nagyban megkönnyíti a nyelv tanulását.

### **Visual Studio**

Az általam is használt integrált fejlesztői környezet a .NET keretrendszerhez.

Csak a Windows operációs rendszereken elérhető. Széleskörű beépített funkcionalitásokkal bír, melyek a .NET keretrendszerhez lettek tervezve. Az ún. Community Edition verziója ingyenesen letölthető a Microsoft weboldaláról.

[A Tour of C# - C# Guide | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/)

[.NET introduction and overview | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/core/introduction)

**unity**

A Unity a Unity Technologies által fejlesztett játékmotor. A játékmotorok tulajdonképpen olyan keretrendszerek, melyek segítségével gyorsan és hatékonyan lehet főként játékokat fejleszteni. Olyan felhasználói felületet biztosítanak, amellyel könnyű dolgozni, és hatalmas segítséget jelentenek a fejlesztés során.

Olyan funkcionalitásokkal rendelkezik, melyek képesek a játékok különböző aspektusait kezelni, mint a

* 2D, és 3D grafikus megjelenítés, ide sorolva az animációkat, animálás lehetőségét.
* Általános, játékbeli fizikai tulajdonságok.
* Hangok kezelése.
* Mesterséges Intelligencia.
* Felhasználó által írt szkriptek kezelése.

A Unity játékmotor segítségével kétdimenziós, illetve háromdimenziós videójátékokat, építészeti és mérnöki látványterveket, animációkat, MI megoldásokat, ezeken kívűl pedig egyéb interaktív tartalmakat lehet létrehozni VR szimulációk által.

[Solutions | Unity](https://unity.com/solutions)

A Unity segítségével lehetőségünk van fejleszteni több, mint 25 platformra.

Például:

\*\*\*kép logójukról??

* Windows
* macOS
* Linux
* Android
* iOS
* tvOS
* watchOS
* PS4, PS5
* XBOX ONE, XBOX SERIES XS
* NINTENDO SWITCH
* Ocolus Rift
* Stadia

[Unity Real-Time Development Platform | 3D, 2D VR & AR Engine](https://unity.com/)

A Unity által támogatott programnyelvek a C, C++, C# és a JavaScript.

Ingyenesen letölthető a Unity Personal kiadása, azonban, ha igazoljuk diák státuszunk, akkor feliratkozhatunk az úgynevezett Student plan csomagra, amely magában foglalja a Unity Pro kiadását, hozzáférést oktató videókhoz és más hasznos szolgáltatást.

**An asset**

Asset-nek nevezünk minden, a projektben használt elemet???. Vizuális és audio elemeket képes reprezentálni, mint például háromdimenziós modelleket, textúrákat, sprite-okat, hangeffekteket, vagy zenét.

Egy asset érkezhet külső forrásból, de az Editoron belül is van lehetőségünk létrehozásukra.

**Asset store**

A Unity Asset Store egy olyan könyvtár, mely ingyenes és fizetős asseteket tartalmaz, melyeket a Unity Technologies, vagy a közösség tagjai készítenek és tesznek közzé. Találunk ott többek közt textúra csomagokat, modelleket, animációkat, egész projekteket.

Miután az Asset Store-ból lementünk egy asset-et, azt az Editoron belül a Package Manager segítségével tudjuk letölteni és importálni.

**Package manager**

A Package egy olyan konténer, amely különböző funkciókat, vagy asset-eket tartalmaz, például:

* Editor eszközöket és könyvtárakat, például szövegszerkesztőt, animáció megtekintőt.
* Runtime eszközöket és könyvtárakat, mint Physics API vagy Graphics pipeline.
* Asset gyűjteményeket, például textúrák, animációk.
* Projekt sablonokat.

Az Editorban a Package Manager ablak a következő módon érhető el: Window -> Package Manager.

**A GameObject**

A GameObject egy olyan, az Editoron belüli objektum, amely komponenseket tartalmaz. Ezek a komponensek határozzák meg az objektum viselkedését és kinézetét. A Hierarchy ablakban tudunk létrehozni objektumokat.

**Components**

A komponensek a játékon belüli események mozgatórugói. Minden objektum funkcionális részei.

Egy objektum kijelölése után megtekinthetjük annak komponenseit az Inspector ablakban. Itt és szkripten keresztül is lehetőségünk van ezeket módosítani, törölni, vagy újat létrehozni.

**Leggyakrabban előforduló komponensek:**

**\*\*mindegyikről egy kép, esetleg paramétereik magyarázata**

**Transform**

Alapértelmezetten minden objektum rendelkezik ezzel a komponenssel. Nem lehet, eltávolítani, vagy olyan objektumot létrehozni, ami ne rendelkezne vele, hiszen ez a komponens adja meg, hogy az objektum hol helyezkedik el, hogyan van forgatva, és méretezve.

**RectTransform**

A RectTransform a sima Transform kétdimenziós megfelelője. Míg a Transform egy konkrét pontot határoz meg a térben, a RectTransform egy négyzetet egy Canvas komponensen, melybe UI elemeket lehet illeszteni.

**Collider**

A Collider-ek adják meg az egyes objektumok körvonalait, így lehetővé téve a fizikai szimuláció során az ütközéseket. 3D esetén a Collider alakjának meg kell egyeznie a Mesh (3D modell, pl. egy kocka) alakjával. 2D esetén olyat kell választani, amely megfelel a neki szánt célnak. Például egy emberi karakter kaphat Box Collider 2D-t, vagy Capsule Collider 2D-t.

Ez az egyik legszéleskörűbben használható komponens. Beépített függvényekkel lehet meghatározni például, hogy mi történjen, ha két Collider (pl. két karakter a játékban) ütközik egymással.

**Rigidbody**

A Rigidbody komponens által vehet részt egy objektum a fizikai szimulációban. Az egyes karaktereket is ezen komponensen keresztül tudja irányítani a játékos szkript segítségével.

Ugyebár a Transform komponens adja meg egy objektum pozícióját. Amikor az objektum helyzete változik, akkor a Transform elküldi az új pozíciót a többi komponensnek, ami így frissít olyan dolgokat, mint például hol legyen megjelenítve az adott objektum, és hogyan vannak a Collider-ek pozícionálva.

A Unity rendelkezik egy a fizikai elemeket szimuláló motorral, amely a Collider-eket mozgatja, és lehetővé teszi, hogy interakcióba lépjenek egymással. Ezt a mozgást viszont továbbitani kell a Transformnak, különben az objektum helyzete nem változna. Így ezért a mozgásért, és kommunikációért, illetve a Collider-ekkel való kapcsolattartásért a Rigidbody felelős.

**Sprite Renderer**

A Sprite-ok kétdimenziós grafikai elemek, tulajdonképpen képek. Ezek a Sprite Renderer komponens által kerülnek megjelenítésre.

**Camera**

A Camera jeleníti meg a játékteret a játékos számára. Legalább egy mindig van egy Scene-ben.

**Animator**

Az Animator komponens biztosítja, hogy animációkat tudjunk hozzáadni az adott objektumhoz. Paraméterként vár egy Animator Controller-t, amely kezeli, hogy mely animációkat mikor, és hogyan játszhatja le, illetve a köztük lévő átmenetet is.

**Canvas**

A Canvas egy olyan terület, amelyre az összes UI elemet kell helyezni. Minden UI objektumot a Canvas objektum gyermekeként kell beállítani, mely rendelkezik egy Canvas komponenssel.

Új UI objektum, például egy Image (GameObject -> UI -> Image) létrehozásakor, automatikusan létrehozásra kerül egy Canvas objektum, ha még nincsen a Scene-ben. A UI objektum pedig ennek a Canvas-nak a gyermekeként jön létre.

A Canvas területe a Scene nézetben egy téglalapként jelenik meg.

A UI elemek a Canvason olyan sorrendben jelennek meg, ahogyan a Hierarchy ablakban szerepelnek. Az első gyermek lesz először megjelenítve, a második másodjára, és így tovább. Ha két elem fedi egymást akkor az utóbb megjelenített lesz felül.

**Text**

A Text komponens, amelyet Label-nek is neveznek, tartalmaz egy mezőt, melybe a megjelenítendő szöveget írhatjuk. Beállíthatjuk a betűtípust, stílust és méretet.

**Image**

Az Image objektum tartalmaz egy RectTransform és Image komponenst. Az Image komponenshez egy sprite-ot tudunk megadni, amelyet megjelenít.

**Button**

A Button tartalmaz egy OnClick UnityEvent, amellyel beállíthatjuk mi történjen, ha rákattintunk.

**Grid**

A Grid tulajdonképpen egy olyan háló, amely beborítja az egész játékteret, és megkönnyíti az egyes objektumok elhelyezését, például Rectangular Grid esetén a négyzet alakú csempe elemek könnyen elhelyezhetőek.

A komponens az egyes cellák helyzetét felelteti meg az objektum lokális helyzetének (pozíciójuk a komponens középpontjához képest). A Transform pedig ezeket a lokális pozíciókat konvertálja globálissá.

**Tilemap**

A Tilemap komponens tárolja és kezeli a kétdimenziós pályák készítésére szánt csempe elemeket. Továbbadja a rá helyezett csempékről a szükséges információkat, a többi kapcsolódó komponensnek, mint a Tilemap Renderer és a Tilemap Collider 2D.

Első alkalommal szükséges letölteni, és importálni a 2D Tilemap Editor csomagot a Package Manager segítségével, mivel a Unity Editor nem tartalmazza alapértelmezetten.(kivéve, ha a unity hubba úgy hozzuk létre a projektet).

Amikor létrehozunk egy Tilemap objektumot, akkor a Grid objektum a Grid komponenssel automatikusan létrehozásra kerül, és a gyermekének állítja be a Tilemap objektumot.

**Script**

A szkriptek, és a kódolás alapvető eleme minden Unity-ben készült alkalmazásnak. Leggyakrabban a játékos által adott input-ot (pl. billentyűlenyomás), és a játékmenetet, eseményeket kezeli.

Például a szkriptekkel irányítjuk a játékost, kezeljük a tárgyait, animációit.

[Unity - Manual: Unity User Manual 2020.3 (LTS) (unity3d.com)](https://docs.unity3d.com/Manual/)

**unity editor: alap layout es windowok bemutatasa, how to import assets**

A Unity Editor a játékmotor grafikus felhasználói felülete, melyben a tényleges fejlesztést végezzük.

\*\*kép editorrol

(A1)Az Editoron belül talákható egy általános menüszalag. Itt tudjuk például menteni, buildelni projektünket, illetve haladó beállításokat végezni rajta.

(A)A Toolbar a menüszalag alatt helyezkedik el. Bal oldalán olyan alapvető eszközöket találunk, amelyekkel a játéktérbeli elemek pozícióját és méretét tudjuk szabályozni. Középen a Play, Pause és Step gombok találhatóak, melyek a szimuláció futását kontrollálják. Jobb oldalon pedig a Layout lenyitható listából módosíthatjuk az Editoron belüli ablakok elhelyezkedését, de azokat manuálisan is a kívánt helyre tudjuk igazítani.

(B)A Hierarchy ablakban láthatjuk az aktív Scene-eket és a hozzájuk tartozó GameObjecteket. Itt tudunk új komponenseket létrehozni és kezelni őket. Ahogyan az ablak neve is sugallja, ez egy hierarchikus, szöveges reprezentáció a játékbeli komponensekről.

(C)A Game nézet szimulálja le, hogyan is fog kinézni, illetve futni az éppen aktív Scene, az elsődleges kamerán keresztül. A szimuláció a Play gomb megnyomásakor kezdődik.

(D)A Scene nézet lehetőséget biztosít a játéktérben történő vizuális navigálásra, és annak elemeinek módosítására. Két-, illetve háromdimenziós megjelenítésre is képes, attól függően milyen projekten dolgozunk.

(E)A fejlesztés során talán az Inspector ablakot használjuk a leggyakrabban. Itt lehet az éppen kiválasztott játékelemhez olyan komponenseket, illetve tulajdonságokat kapcsolni melyek meghatározzák annak viselkedését, kinézetét.

(F)A Project ablak tulajdonképpen egy Editoron belüli fájlkezelő, amely a rendelkezésre álló asseteket jeleníti meg. A projectbe importált asseteket, a létrhozott szkriptjeinket itt találjuk.

(G)A Console ablakban az Editor által, a felhasználó számára küldött üzenetek jelennek meg a szimuláció futása során. Ezek lehetnek hibaüzenetek, figyelmeztetések, vagy a felhasználó által kiíratott üzenetek hibakezelés során.

(H)A Status szalag értesítéseket jelenít meg egyes folyamatokról, illetve gyors elérést biztosít a hozzájuk kapcsolódó eszközökhöz és beállításokhoz.

**I(a) gameobjects and their components (scripteket utana)**

**Common objects and components**

Létrehozása: Hierarchy ablakban jobbklikk-> 2D Object -> Tilemap -> Rectangular

1. **grid, tilemap**

Létrehozáskor a rectangulart választottuk ki ezért a grid tulajdonképpen egy négyzetrácsos háló lesz, melyre pakolhatjuk a csempe elemeket.

Ahoz, hogy létrehozzunk, szerkesszünk, és kiválasszuk a festeni kívánt csempe elemet, a Tile Palette ablakot kell megnyitnunk. (menu: Window > 2D > Tile Palette)

[Unity - Manual: Tilemap (unity3d.com)](https://docs.unity3d.com/Manual/class-Tilemap.html)

\*\*kép tile palettrol

Az ablak felső részén találhatóak olyan eszközök, mint például a kiválasztás, radírozás, festés.

Miután pedig a Creat New Palette gombra kattintás után kiválasztuk a beimportált TileSet assetünket, egy négyzetrácsos hálón látjuk a rendelkezésünkre álló csempekészletet. Amíg ezzel dolgozunk érdemes az ablakot rögzíteni a Hierarchy ablak mellé, nehogy bezáródjon.

Hogy a Scene nézetben található négyzetrácsok mérete megegyezzen a csempék méretével, érdemes ennek megfelelően állítani a méretet. Én fordítva dolgoztam, ami azt jelenti, hogy minden egyes beimportált sprite asset pixels per unit tulajdonságát 256-ra állítottam be.

Ezután a pálya készítés igazán egyszerű. A Tile Palette-en kiválasztjuk a csempét egy kattintással, majd a Scene nézetbe katintással helyezzük le.

Miután elkészítettük a pályát. A Hierarchy ablakban kiválasztjuk a Tilemap objektumot és az Inspector ablakban hozzá kell adnunk az alábbi komponenseket:

* Tilemap Collider 2D:

Biztosítja, hogy a fizikai szimulációban résztvevő objektumok, ne zuhanjanak le, tartsa meg őket. Ennek megfelelően Az Inspector ablak tetején adtam hozzá egy új Layert, amit groundnak neveztem el.

* Composite Collider:

A Tilemap Collider 2D used by composite paraméterét jelöljük be.

Szükségességét al alábbi kép szemlélteti.

\*\*összehasonlító kép

A Tilemap Collider 2D minden egyes csempéhez külön collidert adott. A Composite Collider

ezen collidereket egyesíti.

* Rigidbody 2D:

Composite Collider-el együtt automatikusan hozzáadásra kerül. Mivel nem szeretnénk, hogy a játék futása során a pálya is lezuhanjon a gravitáció hatására, ezért ennek a komponensnek a Body Type paraméterét Static-ra állítottam.

Érdemes megemlíteni, hogy van egy ún. Simulated bejelölhető tulajdonsága, amely ha nincs bepipálva, nem vesz részt a komponenshez tartozó objektum a szimulációban. Ekkor nem lenne aktív a Collider sem, így átzuhannának rajta az objektumok.

1. karakterek

* player
* enemy melee
* enemy ranged
* npcs

1. projectiles

* arrow
* sting
* fireball

1. items

* coin
* potions

1. sima sprites, decor
2. background