**bevezető**

Témaválasztás indoklása, annak elméleti és gyakorlati jelentősége. Célkitűzések és eredmények megfogalmazása.

**tárgyalási rész**

**irodalomkutatás:**

Irodalomkutatás (kb. 2-2.5 oldal) gyanánt összeszedheti a játékkészítéshez felhasználható keretrendszerek fejlődését a nullától a kényelmes Unity, UE világig valami ehhez illő cím alatt.

**Használt technológiák, szoftverek bemutatása.**

Amikor eljut erre a pontra, akkor leírja, hogy a fejlesztés során az Unity-t használta, ezért a következő (al)fejezetben ezt fogja részletesen bemutatni.

Unity, Unity néhány fontosabb ’fogalom’, Unity Editor

Majd a Unity-nél kitér, arra, hogy C#-ben írhatunk szkripteket, ezért adja majd magát, hogy bemutassa a C#-et és a Visual Studio-t.

Tömören, röviden tudassa, hogy miről van szó, milyért van rá szükség, ha alternatív lehetőségek is léteznek, akkor itt is indokolhatja a választást.

Unity Common Components

**játék bemutatása(milyen részletességgel?)**

Először nagyvonalakban szokás bemutatni a játékot, ezt már meg is tette. Utána már részletesebben kell írni az egyes részek megvalósításáról: ennek során beszúrhat rövidebb kódrészleteket is, képernyőképeket is, de ezek terjedelme (egyenként nézve) ne haladja meg a fél oldalt, mert akkor már a függelékben lenne a helye.

Amikor a programot magyarázza, akkor érdemes a lényeget leírni: mi a célja annak a résznek, mit ér el vele.

Tesztelés.

**összefoglalás**

Megállapítások és következtetések, tapasztalatok.

**bevezető**

**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) |

**irodalomkutatás:**

**A Játékmotorok születése**

In early 1989 one sci-fi game engine named Ultima Underworld was developed. That engine had the same name as the game itself. But after releasing of Space Rougue game, Origin System had undertaken the Ultima Underworld engine and developed the algorithm for texture mapping which can be applied to floors, ceilings, walls etc. The maximum system requirement for this was 386 based PC.

In 1993, ID software developed Doom engine , which is not a 3D engine at all, but had capacity to represent objects, characters and whole level map by 2D sprite representation. Rendering was very fast and it needs 386 based PC with standard VGA support to run. Though it was a 2D engine, the illusion created by the developer made it a 3D title.

NovaLogic’s prprietory engine Voxel engine (1992) was the basic engines for all Comanche games. Voxel had their own way to represent volumetric objects as three dimensional bitmaps. Before that all engines applied vector graphics, which was little bit slower in speed and less detailing than 3D bitmap presentation. Blade Runner, Command & Conquer are the noteworthy games developed by Voxel engine.

In late 1993, another game called Duke Nukem 3D was released into the market ,which was developed with the help of Build engine. Like Doom engine it also created 3D effects in 2D interface. It simply varied the sectors with different heights to achieve the illusions. By applying special tags to various spot within a particular sector, developers could make it so that whenever a player move over to that particular spots it switches over to a different sector giving the illusion of changing the levels or environment at run time.

XnGine(1995) was the first ever 3D engine which is developed in DOS base. It would later make use of high resolution graphics and be compatible with 3dfx video cards.

Quake engines(1996) was the first truly 3D game engine by Id software. It had an unique processing capacity to render maps by purging certain areas from processing that the player wouldn’t be able to see. It actually took advantages of Z-buffering, which simply is a method for determining which parts of the maps are visible to the player and only rendering those sections.

Renderware(1996) was the most popular engine for multiplatform games. It supports PlayStation 2, Wii, GameCube, Xbox, Xbox 360, PlayStation 3 and PSP platforms.

Quake II/ id Tech 2 engine(1997) supports native OpenGL , colored lightening effects and C language support. The moddability was increased because of it’s DLL support.

GoldSRC(1998) pushed PC games to a new era. It supports both OpenGL and Direct3D. Some successful game examples which are based on OpenGL are Half-Life, Day of Defeat and Counter Strike etc.

One of the most popular game engine is Unreal engine(1998) , which gave birth to Unreal Tournament game. It integrates its own scripting language called UnrealScript and map editor named UnrealEd. A modified version of Quake II engine was Quake III designed in 1999 which supports 32-bit color, shaders and advance networking. John Slagel, Red Faction's lead programmer, developed the Geodmod engine(2001), which stands for Geometry Modification. As Red Faction's lead designer Alan Lawrence explained to Gamespot, "When a rocket hits a wall, we take this shape and basically subtract that shape from the world. So we boolean with that 'bit' -- we call them GeoMod bits -- and that takes a chunk out of the world."[8]. In 2001 Torque engine was developed to modify the FPS Tribes 2 game. It had on the fly rendering option with less polygon counts and also had a world map editor integrated within it. Serious engine(2001) was designed to allow large spaces and large numbers of on-screen characters at any given time and give birth to popular Serious Sam series. Later in Doom 3 (2004) most light surfaces were also done in real-time, allowing more realistic shadows, but at the expense of being able to render soft shadows. To get around this, projected lights could be used to create the illusion of soft shadows. Half Life 2 creator Source engine (2005) including advanced shader technologies, dynamic lighting and shadows, physics, several effects such as realistic looking reflective water surfaces and real-time motion blur, and much more. CryEngine's(2004) used pixel shaders for realistic water in Far Cry. Developed by Crytek company it produced the game named Crysis, a shader heavy DirectX 10 game. Rockstar Advanced Game Engine(RAGE)(2006) combines a rendering framework, physics engine, audio engine, network libraries, animation engine and scripting language in one package. Grand Theft Auto IV was the finest product of that particular engine. In 2007 Unreal engine and 2008 Frostbite engine took the advancement of game engines to the next level. Battlefield: Bad Company, a game which increased it and up to 92 percent of the environment can be destroyed with unbelievable blasting effects was created by using Frostbite engine[7]. Cry engine 3 by Crytek was launched in 14 Oct’2009 which is again a cross platform engine supports C/C++. As for the PC platform, the engine is said to support development in DirectX 9, 10, and 11.As of June 1, 2009, it was announced that Crysis 2 would be developed by Crytek on their brand new engine.

**Használt technológiák, szoftverek bemutatása.**

Amikor eljut erre a pontra, akkor leírja, hogy a fejlesztés során az Unity-t használta, ezért a következő (al)fejezetben ezt fogja részletesen bemutatni.

Tömören, röviden tudassa, hogy miről van szó, milyért van rá szükség, ha alternatív lehetőségek is léteznek, akkor itt is indokolhatja a választást.

**Unity**

Az olyan általános osztályok összességét, amelyek vélhetoleg sok különféle játék- ˝ ban felhasználhatók, együttesen játék könyvtárnak, vagy játékmotornak (game engine) nevezzük.

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.

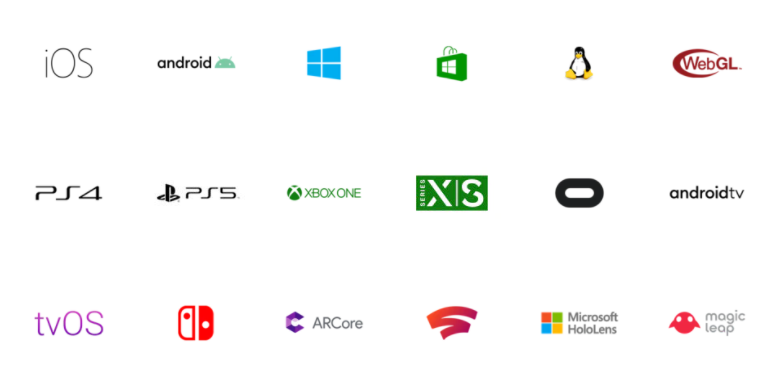
Olyan funkcionalitásokkal rendelkezik, melyek képesek a játékok különböző aspektusait kezelni, mint a

* 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.



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

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.

A Unity nagy versenytársa az Unreal Engine, melyről általánosságban jobb vélemények vannak. Választásom azért esett mégis a Unity-re, mert számomra átláthatóbb felhasználói felülettel rendelkezik, rengeteg oktatóanyag elérhető hozzá és a benne használható programozási nyelv szimpatikusabb.

A Unity-ben lehetőségünk van szkripteket írni a támogatott nyelveken, melyek a C#, UnityScript (úgy is ismert, mint JavaScript Unity-hez) és a Boo.

Majd a Unity-nél kitér, arra, hogy C#-ben írhatunk szkripteket, ezért adja majd magát, hogy bemutassa a C#-et és a Visual Studio-t.

**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)

**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 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.

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.

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

**.NET keretrendszer (átírás alatt)**

[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)

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

A .NET az alábbi operációs rendszerekre történő alkalmazásfejlesztést támogatja:

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

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

Ezek a programnyelvek az alábbiak:

* C#
* F#
* Visual Basic

### **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)

**Néhány fogalom tisztázása**

Mielőtt bemutatnám a Unity Editor felületét, nem árt néhány fogalmat tisztázni, melyekkel később még sokszor találkozunk.

**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.

**GameObject**

A GameObject (a továbbiakban objektum) egy olyan, az Editoron belüli elem, 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.

**Unity Editor**

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.

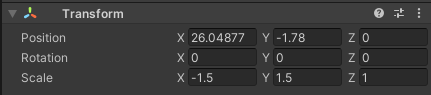
(H)A Status szalag értesítéseket jelenít meg egyes folyamatokról, illetve gyors elérést biztosít a hozzájuk kapcsolódó eszközökhöz és beállításokhoz.

**Leggyakrabban használt komponensek**

Ebben az alfejezetben bemutatom az általam leggyakrabban használt komponenseket, hogy a játék bemutatása gördülékenyebben történjen, ne akkor kelljen a legtöbbet ismertetni.

**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.



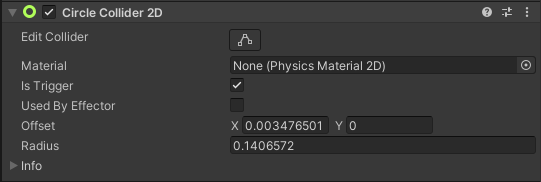
**RectTransform**

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**

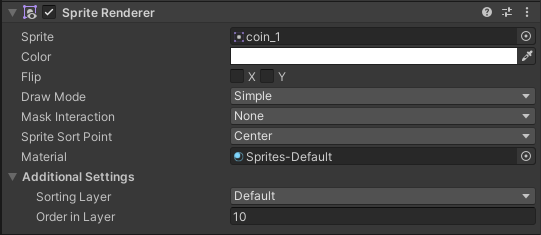
A Rigidbody komponens által vehet részt egy objektum a fizikai szimulációban. Az egyes karaktereket is ezen komponensen keresztül tudja irányítani a játékos szkript segítségével.

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.

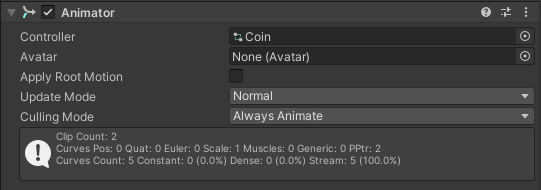


**Camera**

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.

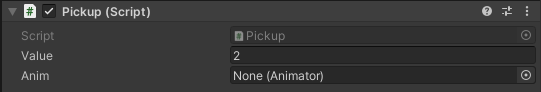
Első alkalommal szükséges letölteni, és importálni a 2D Tilemap Editor csomagot a Package Manager segítségével, mivel a Unity Editor nem tartalmazza alapértelmezetten.(kivéve, ha a unity hubba úgy hozzuk létre a projektet).

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/)

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

**Asset store**

A Unity Asset Store egy olyan könyvtár/online áruház, 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.

\*\*általam használt asseteket inkább ide?

**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.

**how to crate objects+add components**

Új objektum létrehozásához jobb egérgombbal kattintsunk a Hierarchy ablakba, és válasszuk ki a kívánt objektumot. ezután tetszőlegesen mozgatható az egérrel, a szülő-gyermek kapcsolatok beállítása érdekében.

Komponens hozzáadása kétféleképpen lehetséges. Először is az objektum kijelölése után (amely történhet a Hierarchy ablakban, vagy a Scene nézetben) az egyik lehetőségünk, hogy az Inspector ablakban a New Component gombra kattintunk. A másik lehetőség, hogy a menüszalag Components fülére kattintunk. Véleményem szerint az első lehetőség jobb, mivel ekkor egy keresőmezőben is lehetőségünk van kikeresni a kívánt komponenst.

1. **grid, tilemap**

Tulajdonléppen a kétdimenziós játéktér létrehozásának az alapja.

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 szintén 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**

A karakter objektumok felépítése a következő:

\*kép hierarchy-ról

Van egy szülő objektum, mely az adott karakter nevét hordozza. Ezen objektumra vannak beállítva a fizikai szimulációhoz szükséges komponensek, mint a RigidBody 2D, CapsuleCollider 2D és az adott karakter főleg irányítására használatos szkript.

Ezen objektum gyermekei a következőek:

Attackpoint: Warrior esetén ez lesz annak a területnek a középpontja, amelybe ha támadáskor ellenfél található, akkor megsebzi azt.

Hunter és Mage esetén ez az objektum adja meg azt a pontot, ahol támadáskor a nyílvessző, vagy a tűzgolyó inicializálásra kerül.

Sprite: Ezen objektum felelős a karakter vizuális megjelenítéséért és animációjáért. Ennek megfelelően rendelkezik a Sprite Renderer komponenssel és az animációkat kezelő szkripttel. Mage esetén nem létezik ez az objektum, ezek a komponensek a szülő objektumban találhatóak. Ennek okát később fedem majd fel, amikor az animálásról fogok írni.

**enemy melee, enemy ranged**

Ezen objektumok felépítése hasonló a játékos karakterekhez, viszont itt a gyermekek között megtaláljuk még a See objektumot. Ez egy Collidert tartalmaz, melynek funkciója, hogy érzékelje az esetleges ütközést a pályaelemekkel (ground), így ugrásra késztetve a karaktert.

**npcs**

A Non Playable Character rövidítése npc.

Ezeknek általában két funkciója van egy játékban:

* + Életet lehel a játékba a játéktér kitöltésével.
  + Valamiféle interakcióba lehet velük lépni.

Az általam készített játékban 2 NPC található. Az egyik a toronyőr, mely a játékost fogadja a játék kezdetekor, és megadja a kellő iránymutatást. A másik a kereskedő, amelynél vásárolni lehet. Ezek egy alap animációval rendelkeznek, mozogni nem tudnak.

1. **projectiles**

Az Arrow, Sting, FireBall objektumok. A távolsági harcos karakterek lövedékeként kerülnek inicializálásra támadáskor. A viselkedésüket meghatározó szkriptet és a Rigidbody 2D, Collider 2D és Sprite Renderer komponenseket tartalmazzák

1. **items**

**coin**

Gyűjthető objektum, mely fizetőeszközként szolgál. Egy Collider-rel és Animator Controller-el, illetve az interakciót menedzselő szkripttel rendelkezik.

**potions**

Felhasználható objektum, mely a játékos karakterének tulajdonságpontjait növelik.

Négy féle van:

* Health potion: 20 életerőpontot tölt vissza.
* Agility potion: Növeli a mozgási sebességet és az ugrás magasságát.
* Strenth potion: Növeli az alap és a speciális támadás sebzését és a támadási sebességet.
* Special potion: Növeli a speciális támadás sebzését és támadási sebességét. Warrior esetén pedig a területét is.

1. **background**

Háttérként az úgynevezett paralax backgroundot alkalmaztam. Ennek a lényege, hogy több rétegben helyezkednek el egymáson az egyes háttérelemek, és a Camera objektummal együtt mozognak, eltérő sebességgel. Ettől olyan hatást kelt, mintha tényleg lenne mélysége, mintha valóban távol lennének a távoli, és közel a közeli elemek. Maga az objekum egy szülő, amely azon gyermekeket fogja össze, amelyek az egyes háttérelemeket és a mozgatásukat végző szkriptet tartalmazzák.

1. **UI felületek**

A felhasználói felületet maga a Canvas objektum biztosítja. Ezen találhatóak az egyes UI elemek.

A Játékomban ezeket az elemeket 3 csoportba lehet sorolni:

* Dizájn szerepet betöltők: Tipikusan ilyenek az egyes ablakok hátterét, alapját biztosító Image objektumok.
* Interakcióra képes elemek: Például a gombok.
* Tájékoztató szerepet betöltők: Ide sorolnám például az életerő jelző csíkot, és a Text objektumokat, melyek információt adnak át.

1. **sima sprites, decor**

Egyszerű objektumok, melyek csak dekoráció céljából vannak jelen. Ezek csak a Sprite Renderert tartalmazzák megjelenítés céljából. Ilyenek például az épületek vagy a növényzet.