c#, vs, hasznos billentyűkombinációk??

unity

github

jatekrol par szo

assetek: felsorolas szinten par szo roluk es hivatkozas

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

**I(a) gameobjects and their components** (scripteket elotte v utana? utana lehet celszerubb az oroklodes vegett))

1. grid, tilemap
2. karakterek

* player
* enemy melee
* enemy ranged
* npcs

1. projectiles

* arrow
* sting
* fireball

1. items

* coin
* potions

1. sima sprites, decor
2. background

**I(b) scripts**

**monobehaviour, variables, declaring**

1. karakterek

* player
* enemy melee
* enemy ranged

1. projectiles

* arrow
* sting
* fireball

1. items

* coin
* potions

1. background

**I(c) animations**

basicly how to anim and scripts

**II UI**

1. canvas
2. healthbar
3. menuk, gombok, tmp

Előszó???

**C# programozási nyelv:**

I.1. A nyelv története

A C# programozási nyelv a Microsoft új fejlesztési környezetével, a 2002-ben megjelent Visual Studio.NET programcsomaggal, annak részeként jelent meg. Bár a nyelv hosszú múlttal nem rendelkezik, mindenképpen elődjének tekinthetjük a C++ nyelvet, a nyelv szintaktikáját, szerkezeti felépítését. A C, C++ nyelvekben készült alkalmazások elkészítéséhez gyakran hosszabb fejlesztési időre volt szükség, mint más nyelvekben, például a MS Visual Basic esetén. A C, C++ nyelv komplexitása, a fejlesztések hosszabb időciklusa azt eredményezte, hogy a C, C++ programozók olyan nyelvet keressenek, amelyik jobb produktivitást eredményez, ugyanakkor megtartja a C, C++ hatékonyságát. Erre a problémára az ideális megoldás a C# programozási nyelv. A C# egy modern objektumorientált nyelv, kényelmes és gyors lehetőséget biztosítva ahhoz, hogy .NET keretrendszer alá alkalmazásokat készítsünk, legyen az akár számolás, akár kommunikációs alkalmazás. A C# és a .NET keretrendszer alapja a Common Language Infrastructure(CLI).

I.2. A nyelv jellemzői

A C# 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)

A C# egy modern, objektum-orientált és típusbiztonságos programozási nyelv. A C programnyelvi családba sorolható. Lehetőséget biztosít a fejlesztőknek nagy méretű, különböző célú alkalmazások készítésére.

A C# egy objektum-orientált, komponens-orientált programozási nyelv.

**.NET keretrendszer**

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.

## 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

.NET lets you use platform-specific capabilities, such as operating system APIs. Examples are Windows Forms and WPF on Windows and the native bindings to each mobile platform from Xamarin.

### **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.

többi???

### **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.

**github**

A játékom fejlesztése során GitHub-ot használtam verzió-követés céljából.

**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) |

**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:**

**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 2D component places an object under the control of the physics engine. Many concepts familiar from the standard Rigidbody component carry over to Rigidbody 2D; the differences are that in 2D, objects can only move in the XY plane and can only rotate on an axis perpendicular to that plane.

Usually, the Unity Editor’s Transform component defines how a GameObject (and its child GameObjects) is positioned, rotated and scaled within the Scene. When it is changed, it updates other components, which may update things like where they render or where colliders are positioned. The 2D physics engine is able to move colliders and make them interact with each other, so a method is required for the physics engine to communicate this movement of colliders back to the Transform components. This movement and connection with colliders is what a Rigidbody 2D component is for.

**Sprite Renderer**

Sprites are 2D Graphic objects. If you are used to working in 3D, Sprites are essentially just standard textures but there are special techniques for combining and managing sprite textures for efficiency and convenience during development.

Sprites are rendered with a [Sprite Renderer](https://docs.unity3d.com/Manual/class-SpriteRenderer.html) component rather than the [Mesh Renderer](https://docs.unity3d.com/Manual/class-MeshRenderer.html)  
 used with 3D objects. Use it to display images as Sprites for use in both 2D and 3D scenes.

**Camera**

Just as cameras are used in films to display the story to the audience, **Cameras** in Unity are used to display the game world to the player. You will always have at least one camera in a **scene**, but you can have more than one. Multiple cameras can give you a two-player splitscreen or create advanced custom effects. You can animate cameras, or control them with physics. Practically anything you can imagine is possible with cameras, and you can use typical or unique cameras to fit your game’s style.

**Animator**

Use the Animator component to assign animation to a GameObject in your Scene. The Animator component requires a reference to an Animator Controller which defines which animation clips to use, and controls when and how to blend and transition between them.

**Canvas**

The Canvas is the area that all UI elements should be inside. The Canvas is a Game Object with a Canvas component on it, and all UI elements must be children of such a Canvas.

Creating a new UI element, such as an Image using the menu GameObject > UI > Image, automatically creates a Canvas, if there isn't already a Canvas in the scene. The UI element is created as a child to this Canvas.

The Canvas area is shown as a rectangle in the Scene View. This makes it easy to position UI elements without needing to have the Game View visible at all times.

Canvas uses the EventSystem object to help the Messaging System.

UI elements in the Canvas are drawn in the same order they appear in the Hierarchy. The first child is drawn first, the second child next, and so on. If two UI elements overlap, the later one will appear on top of the earlier one.

To change which element appear on top of other elements, simply reorder the elements in the Hierarchy by dragging them. The order can also be controlled from scripting by using these methods on the Transform component: SetAsFirstSibling, SetAsLastSibling, and SetSiblingIndex.

**Text**

The Text component, which is also known as a Label, has a Text area for entering the text that will be displayed. It is possible to set the font, font style, font size and whether or not the text has rich text capability.

**Image**

An Image has a Rect Transform component and an Image component. A sprite can be applied to the Image component under the Target Graphic field, and its colour can be set in the Color field. A material can also be applied to the Image component.

**Button**

A Button has an OnClick UnityEvent to define what it will do when clicked.

**Grid**

The Grid component is a guide which helps to align GameObjects, such as Tiles, based on a selected layout. The component transforms Grid cell positions to the corresponding local coordinates of the GameObject. The Transform component then converts these local coordinates to world space or global coordinates.

**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**

Scripting is an essential ingredient in all applications you make in Unity. Most applications need scripts to respond to input from the player and to arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behaviour of objects or even implement a custom AI system for characters in the game.

A szkriptek, és a kódolás alapvető eleme minden Unity-ben készült alkalmazásnak.

[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 elotte v utana? utana lehet celszerubb az oroklodes vegett))**

**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.

\*\*lö 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