EÖTVÖS LORÁND TUDOMÁNYEGYETEM A képen szöveg, tároló, porcelán látható

Automatikusan generált leírás

INFORMATIKAI KAR

PROGRAMOZÁSI NYELVEK ÉS FORDÍTÓPROGRAMOK TANSZÉK

**Hálózaton játszható szerepjáték Unity alapon**

|  |  |  |
| --- | --- | --- |
| **Témavezető:**  Pataki Norbert  Adjunktus, PhD |  | **Szerző:**  Prucs Ákos András  Programtervező Informatikus Bsc |
| Budapest, 2022 | | |

[Bevezetés 4](#_Toc102945365)

[Motiváció a témaválasztás mellett 4](#_Toc102945366)

[A szakdolgozat témája 4](#_Toc102945367)

[Felhasználói dokumentáció 6](#_Toc102945368)

[Rendszerkövetelmények 6](#_Toc102945369)

[A program telepítése és indítása 6](#_Toc102945370)

[Szerver létrehozásához szükséges hálózati beállítás 7](#_Toc102945371)

[A játék célja 7](#_Toc102945372)

[Irányítás 7](#_Toc102945373)

[Felvehető tárgyak 7](#_Toc102945374)

[Színkód 8](#_Toc102945375)

[Fejlesztői dokumentáció 9](#_Toc102945376)

[Unity bemutatása 9](#_Toc102945377)

[Netcode for Gameobjects bemutatása 10](#_Toc102945378)

[Felhasználói esetek 11](#_Toc102945379)

[Program szerkezete 13](#_Toc102945380)

[Gameobjectek hierarchiája **Hiba! A könyvjelző nem létezik.**](#_Toc102945381)

[Modell 13](#_Toc102945382)

[Item és csomagoló osztályok (collider, rigidbody, spawn, InetworkSerializable, NetworkBehaviour, NetworkObject, NetworkTransform) 13](#_Toc102945383)

[Inventory (események, delegációs függvények) 13](#_Toc102945384)

[Chest 13](#_Toc102945385)

[Enemy (logika, navmeshagent, prefab) 13](#_Toc102945386)

[Bullet Movement (spawn stratégia) 13](#_Toc102945387)

[Damageable (komponens) 13](#_Toc102945388)

[Player Controller (input, iflocalclient, rotation, kamera, egyedi collision detection, prefab, NetworkClientTransform) 14](#_Toc102945389)

[UiHealthBar, UiExperienceBar 14](#_Toc102945390)

[UiInventory (drag and drop) 14](#_Toc102945391)

[ItemAssets (Spriteok, felbontások) 14](#_Toc102945392)

[Pálya felépítése (tilemap, collider, layers) 14](#_Toc102945393)

[Game Manager 14](#_Toc102945394)

[Connection Manager 14](#_Toc102945395)

[UI Manager 14](#_Toc102945396)

[Játék tesztelése 14](#_Toc102945397)

[Játék bővíthetősége 14](#_Toc102945398)

[Hivatkozások 15](#_Toc102945399)

[Köszönetnyilvánítás 16](#_Toc102945400)

# Bevezetés

## Motiváció a témaválasztás mellett

A videójátékok már gyerekkorom óta részesei az életemnek. Hálával tartozom, amiért általuk olyan sok barátot sikerült megismernem. Barátokat, akik noha különböző utakat járnak és másfajta életet élnek, de a közös hobbink összehoz minket. A játék remek kapocs egy közösségben és segít egy nyelvet beszélni. Szerencsésnek tartom magam, amiért már olyan korán belecsöppentem ebbe a világba.

A játékfejlesztés gondolata gyakran megfordult a fejemben. Mindig is érdekelt a kérdés, hogy vajon hogyan készül el egy-egy játék, próbáltam mögéjük látni, kíváncsi voltam az alkotásuk mögötti technikai részletekre. Sokáig azonban nem éreztem magam késznek, hogy saját magam is belevágjak egy játék elkészítésének. Megijesztett az összetettségük és kételkedtem a saját képességeimben.

Az egyetemi képzésem alatt, ahogy bővült a tudástáram, úgy egyre magabiztosabb lettem. Elkezdtem úgy érezni, hogy a gondolataimban létező akadályok eltűnnek. Többé már nem riadtam vissza a játékfejlesztéstől. Így jutottam el ahhoz a gondolathoz, hogy a szakdolgozatomban egy játék programot fogok megvalósítani.

## A szakdolgozat témája

A szakdolgozatomban egy „Top-down RPG” játékot fogok elkészíteni. A top-down játékok legtöbbször egy kétdimenziós világot képeznek, ahol a játékos a karakterét fenti nézetből képes irányítani. Az RPG a role-playing game avagy szerepjáték rövidítése, ahol a játékos felveheti egy képzeletbeli karakter szerepét.

A tipikus játékelemei egy ilyen műfajnak, hogy a karakter szabadon mozoghat a világban és interakciókat végezhet az abban lévő objektumokkal, más játékosokkal vagy a számítógép által irányított lényekkel. Az interakciók által megszerezhető tárgyak a játékos hátizsákjában tárolódnak. A megszerzett ruhákat és fegyvereket felszerelheti a karakterére, amik így bónuszokat nyújtanak neki a világban.

A megvalósítandó játékhoz a fent említett játékelemeket fogom megalkotni. Ehhez a Unity Technologies által fejlesztett Unity videójáték-motort fogom használni és a C# nyelvet. A hálózati kommunikációért a Unity-hez készült Netcode for Gameobjects nevezetű fejlesztői csomagot fogom használni.

# ****Felhasználói dokumentáció****

## Rendszerkövetelmények

A program a Unity *2021.1.14f1* verziója alatt fut, és a rendszerkövetelmények igazodnak a verzió követelményeihez:

* Legalább Windows 7-es operációs rendszer vagy annál jobb.
* SSE2 utasításkészlettel rendelkező 32 vagy 64 bites processzor.
* DirectX10, 11 vagy 12 kompatibilis videokártya.
* Maximum 2 GB memória.
* Maximum 50 MB szabad tárhely a háttértárolón.

A Unity lehetőséget ad arra, hogy több platformmal is kompatibilis legyen az alkalmazásunk, azonban ennél a játéknál én csak Windowsra fókuszáltam.

A játék fejlesztéséhez Windows 10 operációs rendszert használtam, Intel Core i7-8750H 2.20 gigahertzes processzort, GeForce RTX 2060 típusú videokártyát és 16 GB memóriát.

## A program telepítése és indítása

A program nem igényel telepítést. Az állomány letöltése is kicsomagolását követően a futtatható bináris fájlal egyből indítható. A játék megnyitását követően a felhasználó egy menüben találja magát, ahol két opciója van. A „Host Game” gomb megnyomásával saját játékot indíthat. A „Join Game” gomb megnyomásával pedig lehetősége nyílik már futó, a hálózaton elérhető játékhoz csatlakoznia.

Saját játék indításakor a program kéri egy port megadását. A szerver ezen a porton figyeli a kívülről érkező kliensek kapcsolódási kérvényeit és üzeneteit. Amennyiben a felhasználó egyedül kíván játszani a portot figyelmen kívül hagyhatja és elindíthatja a játékot. Amennyiben egy meglévő hálózathoz kíván csatlakozni a program egy címet és egy portot kér, amin a szerver elérhető.

## Szerver létrehozásához szükséges hálózati beállítás

Amennyiben a játékot készítő fél (host) belső hálózattal rendelkezik, szükséges lehet a porttovábbítás beállítása, hogy a kívülről érkező felek is kapcsolódni tudjanak a játékba. Ez a belső hálózatért felelős router konfigurálásával lehet megtenni. A folyamat lényege, hogy azokat a publikus címre érkező üzeneteket melyeket az általunk megadott portra küldenek, a router továbbítsa a saját állományunkhoz.

Amennyiben a játékhoz a belső hálózatról akarnak kapcsolódni a kliensek, a porttovábbításra nincsen szükség. Csupán a hálózatunkon kiosztott ip címünket kell megadnunk nekik.

Ha a játékhoz egy gépről akarunk csatlakoztatni több klienset, akkor a hálózati beállításoktól eltekinthetünk. A játék által felkínált ip címet és portot használjuk, mivel az alapértelmezetten a lokális állományunk címe van megadva.

## A játék célja

Sikeres indításkor megjelenik a karakterünk az előre elkészített pályán. Két fajta pályarész létezik: a kezdő, ahonnan a játék indul és a dinamikusan megjelenő részek, avagy szobák, amik feloldódnak amint teljesítettük az adott szintet. A cél, hogy minden szobát kitisztítsunk és minél magasabb szintre jussunk el. Idővel nehezedik a játék, megnő az ellenfelek száma és erőssége, néhány körönként pedig egy különleges ellenfél jelenik meg, az eddigiekhez képest kiemelkedő értékekkel.

### Irányítás

A karaktert irányítani a w, a, s, d gombokkal és a nyilakkal lehet a billentyűzeten. Az iránya követi az egér mozgását és támadni a bal klikk lenyomásával lehet, amennyiben rendelkezik felszerelt fegyverrel. A TAB lenyomásával megnyílik a táska, itt lehetőség van a megszerzett tárgyak használatára vagy eldobására. Amennyiben az adott tárgy egy ruhadarab vagy fegyver, akkor azt fel lehet ruházni a karakterünkre a megfelelő rubrikába húzással vagy a jobb klikkel.

### Felvehető tárgyak

Ahhoz, hogy minél nagyobb szintre eljussunk szükség van minél jobb minőségű ruhákra és fegyverekre. Minden ellenfél legyőzésekor esély nyílik rá, hogy tárgyakat - a továbbiakban itemeket - hagyjon maga mögött a földön. Ezeket úgy vehetjük fel, ha rásétálunk, feltéve, hogy nem telt meg a hátizsákunk, ekkor ki kell dobnunk a már nem szükséges dolgokat. A pályán elhelyezkednek kincsesládák is, melyek kinyitáskor itemeket hagynak hátra a földön. Egy ládát érintkezéssel lehet kinyitni.

Fegyvereken és ruházaton kívül három másik típusú item vehető fel a földről:

* Health Potion: Használatkor feltölti a karakterünk életerejét.
* Gold: Nincs különösebb haszna.
* Experience point: Növeli a karakterünk tapasztalatát.

A kör elején maximum életerővel rendelkezünk. Amennyiben a harcok során az életerő nullára csökken a karakterünk veszít, elvesznek a tárgyai és a kezdő pályarészen indul újból. Egy másik játékelem a tapasztalat pont. Lehetőségünk van a karakterünk erősítésére azáltal, hogy tapasztalatot gyűjt. Amikor elér egy mennyiséget szintet lép és erősödik. Ebben a játékban szintlépéskor megnő a mozgási sebességünk.

### Színkód

Ahogy az egy tipikus RPG játékban lenni szokott, a ruhák és fegyverek minőségét a színük adja meg. A minőséget négy kategóriába van sorolva a gyengétől a legerősebbig:

* Common, szürke
* Good, zöld
* Rare, lila
* Legendary, sárga

# ****Fejlesztői dokumentáció****

## Unity bemutatása

A Unity elsősorban a játékfejlesztésről híres, azonban az alkalmazást használják animációk készítéséhez, építészeti vizualizációkhoz és szimulációkhoz is. A motor képes két- és háromdimenziós „világok” megjelenítésére is. A játékom elkészítéséhez a kétdimenziós konfigurációt választom és a *Unity Editorban* végzem a fejlesztést.

A Unity által szimulált világban megjelenő elemek a ***GameObjectek***. Ezeket az objektumokat képesek vagyunk egymásba ágyazni és ezáltal fa struktúrákat kialakítani, ezzel megkönnyítve az összetett tárgyak elkészítését. GameObjectek egy általunk megadott gyűjteményét **jelenetnek** nevezünk és könnyedén válthatunk jelenetek között futási időben és a Unity Editorban is. Egy egyszerű példa a *jelenetek* alkalmazására egy videójáték esetében a főmenü és a játék jelenetének váltakozása. Az általam elkészített játékban egyetlen jelenetet használok és kód segítségével futási időben változtatom annak állapotát.

Ahhoz, hogy egy Gameobjectet tulajdonságokkal ruházzunk fel ***komponenseket*** kell hozzárendelnünk. A Unityben rengeteg előre elkészített komponenst található. Némelyek csupán egy egyszerű geometriai alakzatot rendelnek hozzá az objektumunkhoz és vannak, amik bonyolultabb logikai képességeket. Minden GameObject tartalmaz egy *Transform* komponenst, ami a világban lévő pozícióját, rotációját és méretét tartja számon. Játékoknál és szimulációknál hasznos komponensek a Colliderek, amik detektálják és jelzik, ha két objektum összeütközik. Valamint a Rigidbody komponensek, amik a gravitációt és fizikai erőt szimulálni képes logikával rendelkeznek.

Lehetőségünk van saját logikát létrehozni, majd egy komponens formájában hozzárendelni a GameObjecthez. Ehhez egy olyan C# nyelven írt osztályt kell létrehoznunk, amely leszármazottja a Unity könyvtárában lévő **MonoBehaviournak**. A leszármazottak öröklik a *Start()* és *Update()* metódusokat. Az előbbi egyszer fut le, az objektum inicializálásakor, ellenben az *Update()* minden képkocka frissítéskor lefut. Ezeket a metódusokat nem kötelező használni minden MonoBehaviour leszármazottnál, azonban hasznos grafikai elemeknél és a felhasználótól érkező input feldolgozásánál.

Még egy említésre való elem a Unityben a **Prefabek** jelenléte. A Unity Editorban előre elkészíthetünk adott komponensekkel felszerelt GameObjectet és elmenthetjük későbbi használatra. A Prefabek segítenek elkerülni a kód redundanciáját és futási időben elérhetőek. A későbbiekben én is alkalmazom őket a játékos karaktere előkészítéséhez vagy a pályán elhelyezkedő szobák elkészítéséhez.

## Netcode for Gameobjects bemutatása

A Netcode for Gameobjects – továbbiakban Netcode – egy magas szintű fejlesztői csomag, ami hálózati képességeket nyújt a GameObject és MonoBehaviour alapú munkafolyamatoknak. A használatához a jelenetben létre kell hozni egy **NetworkManager** komponenssel (scripttel) rendelkező Gameobjectet, ami egy *Singleton* osztály és elérhető lesz a hálózat bármelyik oldalán.

A hálózaton szereplő feleknek három fajtája létezik: *client (kliens)*, *server (szerver)* és *host*. A játékban azonban csak a játékot készítő host, és az ahhoz kapcsolódó clientek lesznek jelen. A host egy olyan fél, amin egyszerre fut a server és a client programja. Amikor lentebb a szerverre utalok, akkor a host állományán futó szerverre gondolok. Amikor kliensről van szó, akkor a szerverhez csatlakozó állományokra és a hoston futó kliensre is gondolok egyaránt.

A Netcode kibővíti a MonoBehaviour osztályt egy **NetworkBehaviour** osztály kiterjesztésével, ezáltal a hálózati működéshez szükséges funkcionalitást hozzáadva az objektumoknak. A NetworkBehaviour használatának feltétele, hogy az adott GameObject vagy egy szülője rendelkezzen a **NetworkObject** komponenssel, ami futási időben egy a hálózat minden felén megegyező azonosító számot rendel az objektumhoz.

A NetworkBehaviour rendelkezik az *OnNetworkSpawn()* és *OnNetworkDespawn()* metódusokkal, amik ***spawnolás*** és ***despawnoláskor*** hívódnak meg. Unityben, ha egy prefabet létre akarunk hozni a jelenetben ahhoz a UnityEngine könyvtár statikus Instanciate() metódusát kell meghívnunk. Ez viszont nem elég, ha a hálózat minden felén létre akarunk hozni egy objektumot. Ehhez regisztrálnunk kell a prefabet a NetworkManager konfigurációjában, majd ezt követően a NetworkObject Spawn() metódusával minden kliensnél létrehozhatjuk az objektumot.

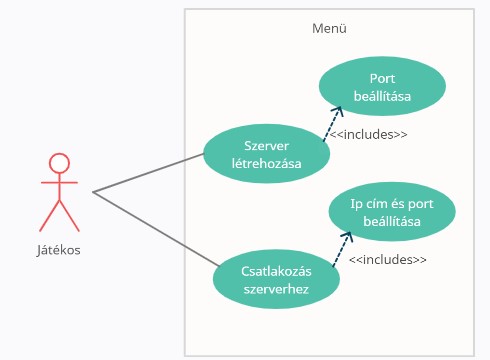
Egy felmerülő probléma, hogy ha minden félnél jelen van az objektum, rajta a logikáért felelős kóddal, akkor hogyan különítjük el a szerver és a kliens feladatait. Erre nyújt megoldást a NetworkManager *IsClient, IsServer, IsHost* boolean típusú változói, melyekkel eldönthetjük, hogy éppen a hálózat melyik oldalán fut a kódunk. A játékban legtöbbszőr ezek segítségével választom el a szerver (ez esetben host) és a kliens logikáját. Az ellenfelek esetében például a host végzi a logikáért felelős számításokat, a kliens csupán lefrissíti az ellenfél pozícióját az Update() metódussal.

A kommunikáció állományok között távoli eljáráshívással vagy **RPC**-vel (Remote Procedure Call) történik. A Netcode megkülönböztet Client RPC-t és Server RPC-t. Az előbbit a szerver hívja meg, és a kliensek állományán fut le a kód. Az utóbbit egy kliens hívja meg és a szerveren fut a kód. Ahol pedig szükség van állapotokra és azok szinkronban tartására a felek között ott a Netcode könyvtár által biztosított **NetworkVariable<T>** generikus változót használom. Ennek előnye, hogy csak a szerver rendelkezik írás joggal, valamint állapotváltozáskor jelzést kapnak a kliensek is, hogy reagálni tudjanak rá.

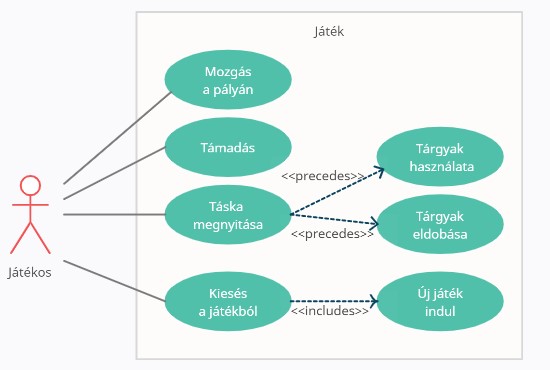
A Netcode további, általam felhasznált részeit lentebb fogom részletezni, ahol egyből a felhasználásukról is írni tudok.

## Felhasználói esetek

A felhasználói esetek diagram két részből áll. A menüből, amivel a felhasználó a játék indításakor találkozik, és a játék eset diagramjából.

  
Főmenü felhasználói diagramja

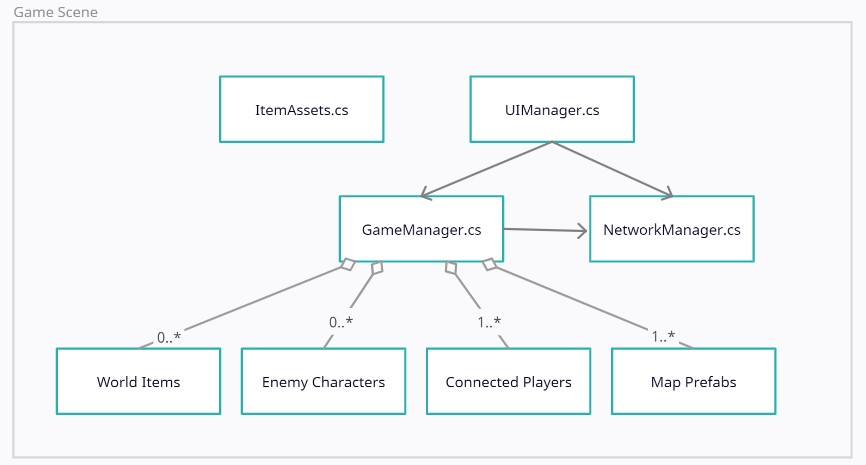
A menüben a felhasználónak két lehetősége van, létrehozhat egy saját játékot a host szerepeként, vagy csatlakozhat egy már létező játékhoz a hálózaton. Saját játék létrehozásakor a felhasználónak be kell állítani egy portot, amin a szerver figyelheti a bejövő üzeneteket. A szerver portja alapértelmezetten 7777. Már meglévő játékhoz csatlakozásnál a program egy ip címet és egy portot kér a felhasználótól. Alapértelmezetten az ip cím a lokális állomány címe és a port megegyezik a fent említettel.

  
Játék felhasználói diagramja

Játék létrehozásakor vagy sikeres csatlakozáskor a felhasználó a játékban találja magát. Itt lehetősége van a karaktere mozgatására a nyilak vagy a w, a, s és d gombokkal. A bal klikk lenyomásával támad a karaktere. A tabulátor lenyomva tartásával megjelentik a táskáért felelős felhasználói felület. Itt lehetősége van a jelenleg nála lévő tárgyak használatára vagy eldobására. Amennyiben a felhasználók mind kiesnek a játékból az újraindul és megpróbálhatják meg egyszer. Több játékos esetén, ha az egyikük kiesik, akkor elveszíti a karaktere fölötti irányítást és a kamera a még bent lévő játékosokat figyeli.

## Program szerkezete

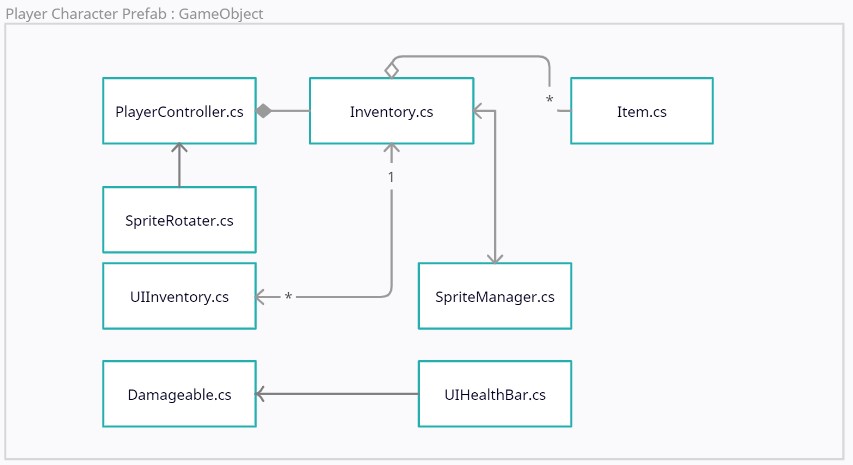
Az alábbi részben átfogóan bemutatom a játék felépítését, a különálló részeit és azok kapcsolatát egymással.

  
A játék jelenetének felépítése.

Korábban említettem, hogy a Unity által kínált jelenetek funkcionalitását nem fogom kihasználni, ugyanis én egyetlen jelenetet használok és az általam írt kód felel ennek a jelenetnek a váltakozásáért. A fenti ábrán a jelenet felépítését ábrázolom. A „.cs” végződésű elemek C# kódok (scriptek) komponensként a saját GameObjecteiken. Ezek az osztályok jelen vannak a programban az indítástól kezdve a bezárásig. Elérhetőek bárhonnan a jelenetből, ezért információt juttathatnak más osztályoknak. Erre példa az ItemAsset osztály, aminek feladata indításkor betölteni szükséges Prefabeket és a játékban később felhasználandó grafikai elemeket.

A játék indulását követően a felhasználót eligazító egyszerű menü az UIManager osztály felelőssége. Kommunikál a NetworkManager osztállyal a hálózati beállításokról és tájékoztatja a GameManager osztályt a játék indulásáról. Az utóbbi osztály főképp szerveroldali kódot tartalmaz. Figyelemmel kíséri a játék állapotát, létrehozza az ellenfeleket, legenerálja a pályát és szamon tartja a csatlakozott játékosokat. Amikor egy felhasználó sikeresen csatlakozik a játékhoz a GameManager létrehoz neki egy karaktert az előkonfigurált Player Prefab alapján és átadja a tulajdonjogot az adott kliensnek.

A NetworkBehaviour egyik adottsága, hogy a létrehozó fél tulajdonjoggal rendelkezik felette, amit később átadhat más feleknek is. Egy NetworkBehaviour leszármazott osztály tulajdonjogával rendelkező kliens kaphat írásjogot a saját hálózati változóira, egyedül ő hívhat meg RPC-ket az adott scriptből és ellenőrizhetjük éppen melyik állományon fut a kód az örökölt *IsOwner* és *IsLocalPlayer* változókkal.

  
A játékos karakterén lévő komponensek és azok viszonyai egymással.

## Modell

### Item és csomagoló osztályok (collider, rigidbody, spawn, InetworkSerializable, NetworkBehaviour, NetworkObject, NetworkTransform)

### Inventory (események, delegációs függvények)

### Chest

### Enemy (logika, navmeshagent, prefab)

### Bullet Movement (spawn stratégia)

### Damageable (komponens)

### Player Controller (input, iflocalclient, rotation, kamera, egyedi collision detection, prefab, NetworkClientTransform)

### UiHealthBar, UiExperienceBar

### UiInventory (drag and drop)

### ItemAssets (Spriteok, felbontások)

### Pálya felépítése (tilemap, collider, layers)

### Game Manager

### Connection Manager

### UI Manager

## Játék tesztelése

## Játék bővíthetősége

# Hivatkozások

# Köszönetnyilvánítás