



# Játékfejlesztés Unity keretrendszerben

## Készítette

Szabó Márk

programtervező informatikus Bsc.

## Témavezető

Troll Ede

tanársegéd

EGER, 2025

# Tartalomjegyzék

<b>Bevezetés</b>	<b>4</b>
<b>1. Technológiai áttekintés</b>	<b>5</b>
1.1. Enginek . . . . .	5
1.1.1. Unreal . . . . .	5
1.1.2. Godot . . . . .	6
1.1.3. Unity . . . . .	6
1.2. Többjátékos technológia a játékfejlesztésben . . . . .	6
1.2.1. Korai megoldások . . . . .	6
1.2.2. Kliens-Szerver . . . . .	7
1.2.3. Peer-to-Peer (P2P) . . . . .	8
<b>2. Unity</b>	<b>9</b>
2.1. Unity6 . . . . .	9
2.2. Felépítése . . . . .	10
2.2.1. GameObject . . . . .	10
2.2.2. Prefab . . . . .	12
2.2.3. Editor . . . . .	12
2.3. Relay . . . . .	13
2.3.1. Relay Server . . . . .	13
2.3.2. Üzenet küldési protokoll . . . . .	14
<b>3. Rendszerterv</b>	<b>15</b>
3.1. Rendszer célja . . . . .	15
3.2. Követelmények . . . . .	15
3.3. Architektúrális terv . . . . .	15
3.4. Használt fejlesztői eszközök . . . . .	15
<b>4. Saját szoftver megvalósítása</b>	<b>16</b>
4.1. Texas Hold’Em . . . . .	16
4.1.1. Szabályok ismertetése . . . . .	16
4.1.2. Használt kézkiértékelő algoritmus . . . . .	19

4.2.	Többjátékos kapcsolat megvalósítása . . . . .	22
4.2.1.	Kapcsolat kezelése a PC oldalon . . . . .	22
4.2.2.	Kapcsolat kezelése a Mobile oldalon . . . . .	22
4.3.	Játékmenet megvalósítása . . . . .	22
4.3.1.	Modul1 . . . . .	22
4.3.2.	Modul2 . . . . .	22
<b>5.</b>	<b>Tesztelés</b>	<b>23</b>
5.1.	Közös DLL tesztelése . . . . .	23
5.2.	Játékok tesztelése . . . . .	23
5.3.	Összefoglalás . . . . .	24
	<b>Összegzés</b>	<b>25</b>
	<b>Irodalomjegyzék</b>	<b>26</b>

# Bevezetés

Magamról, hogy kezdett el érdekelni az infó, majd hogy jutottam a Játékfejlesztésig.

# 1. fejezet

## Technológiai áttekintés

Manapság a játékfejlesztés legelterjedtebb módja a játékmotorok használata. Alternatív megoldásként azonban még előfordul az API-alapú fejlesztés is, amely során különféle API-k – például grafikai vagy multimédiakezelő API-k – segítségével történik a fejlesztés. A szakdolgozatom elkészítéséhez én a játékmotor-alapú fejlesztést választottam, így ki tudtam használni az általa kínált lehetőségeket és eszközöket.

### 1.1. Enginek

A játékmotorok olyan fejlesztői környezetek, amelyek célja, hogy egyszerűsítsék és felgyorsítsák a játékok fejlesztésének folyamatát. Egy tipikus játékmotor különböző komponenseket és funkciókat szolgáltat a fejlesztőnek, mint például: grafikai renderelést, fizikai szimulációt, animációkezelést, hangkezelést. A játékmotorok használata megkönnyíti a fejlesztők dolgát, viszont nem ad annyi rugalmasságot, mint az API alapú fejlesztés. [1]

#### 1.1.1. Unreal

Az Unreal Engine egy professzionális, nagy teljesítményű játékmotor, amelyet az Epic Games fejleszt. Főbb előnyei közé tartozik a kiemelkedő grafikája. Az ingyenes hozzáférése és a vizuális programozást lehetővé tevő Blueprint rendszere miatt kisebb fejlesztők körében is népszerűvé vált. Elsősorban C++-ban történő fejlesztésre optimalizált, de a már említett Blueprint-rendszernek köszönhetően programozói tapasztalat nélkül is lehetséges játékot fejleszteni benne. A játékmotorban sok újrahasználató komponens megtalálható, így a gyakran használt dolgokat pl. karakter mozgást, nem kell felépíteni az alaptól. A játékmotort 3D játékok fejlesztésére találták ki. [2]

### 1.1.2. Godot

A Godot egy nyílt forráskódú, ingyenesen elérhető játékmotor. Nagy előnye a könnyű használhatóság, a kis erőforrás igénye, valamint a nyílt forráskód nyújtotta rugalmasság és közösségi támogatás. Támogatja a 2D és 3D játékfejlesztést egyaránt, natív scriptnyelve a GDScript (Pythonhoz hasonló nyelv). [3]

### 1.1.3. Unity

A Unity az egyik legismertebb és legelterjedtebb játékmotor a piacon, amely támogatja a mobil, asztali, konzolos, valamint webes játékok fejlesztését 2D és 3D játékokhoz egyaránt. Jelentős előnye a könnyen tanulható felhasználói felület és a jól dokumentáltság. Elsősorban a C# nyelvet használja scriptelésre, de támogatja a vizuális programozást is. A játékmotorhoz tartozik több különböző előfizetési szint is, de van ingyenesen használható verziója is. Mivel a C# közel áll hozzám, tapasztalatom is van már ezzel a játékmotorral és maga a játékmotor adottságai pont megfelelték a szakdolgozatom követelményeinek, ezért erre a játékmotorra esett a választásom. Erről a játékmotorról a 2. fejezetben bővebben írok. [4]

## 1.2. Többjátékos technológia a játékfejlesztésben

A mai megjelenő játékok nagy része már tartalmaz valamiféle többjátékos módot. Ez a játékfejlesztés elején még csak helyi megoldásokat jelentett, de ma már hatalmas online terekben játszódó játékokról is beszélhetünk.

### 1.2.1. Korai megoldások

Többjátékoskal játszható játék először stratégia kör alapú játékokban jelent meg, de ez még nem egyidejű játékot jelentett. Ezek után egyidejű többjátékos játék lehetősége először egy 2 játékoskal egyszerre egy helyen játszható játékkal jött létre, ez a játék volt a jól ismert Pong. A Pong után kezdtek egyre jobban elterjedni a helyi többjátékos játékok, ahol az egymás elleni játék helyett, egymással együtt működve is tudtunk játszani. Egy népszerű módszer a helyi többjátékos játékok megvalósításánál az osztott képernyő, ahol általában el van választva a kijelző több egyenlő részre, ahol 1-1 részt 1-1 játékos kezelhet. A 2000-es évek elején pedig már megjelent az online többjátékos játékok lehetősége is, ahol már nem volt szükség egy térben tartózkodni a többi játékoskal. [5]

### 1.2.2. Kliens-Szerver

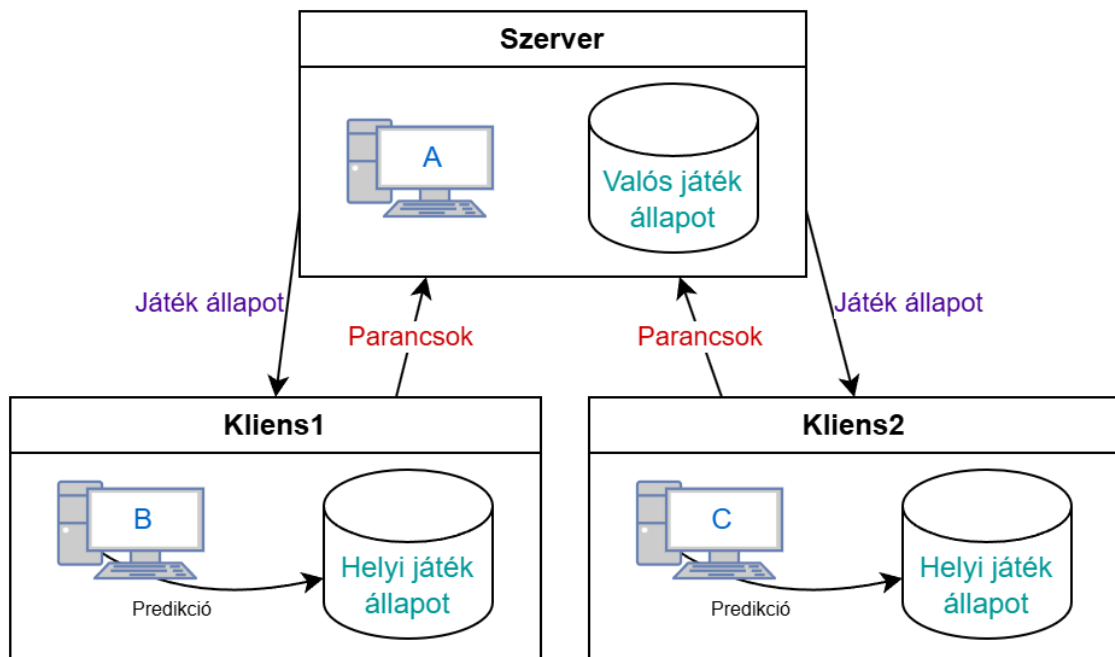
A Kliens-Szerver architektúra az egyik legtöbbet használt módszer az online többjátékos játékok megvalósításához. Az architektúrában 2 szerep van:

– Szerver:

- Felelős a játék logikájának feldolgozásáért, az interakciók kezelésért, az erőforrások kezeléséért és a kliensek közötti információk szinkronizálásáért.
- A szerver lehet egyetlen gépen elhelyezett (dedikált szerver) vagy a játékosok egyik gépén futó (hosztolt szerver).
- A szervernek egyszerre több kliens kéréseit kell kezelnie, ami azt jelenti, hogy skálázhatónak és hatékornynak kell lennie.

– Kliens:

- A játékos játékpéldánya. Bemeneti parancsokat küld a szervernek, és frissítéseket kap a játék állapotáról.
- Felelősek a játékvilág rendereléséért és a helyi interakciók kezeléséért.



1.1. ábra. Kliens-Szerver architektúra

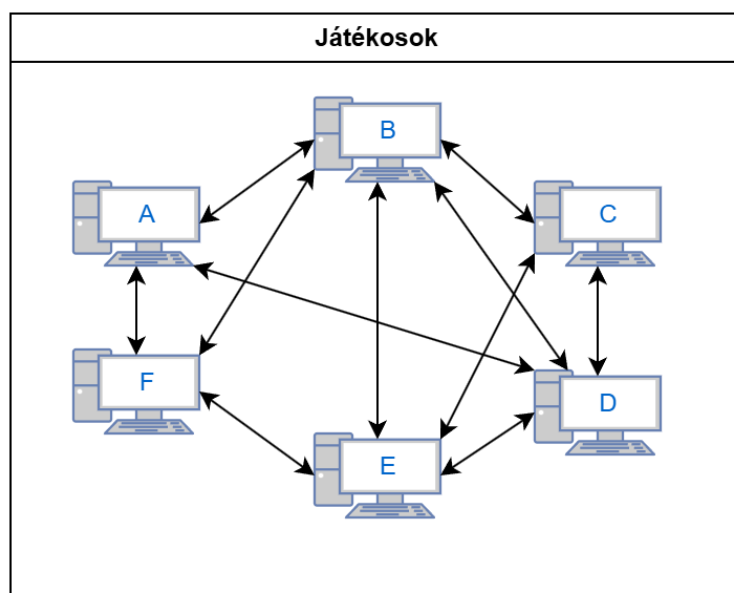
A kliens és a szerver megbízható hálózati protokoll (pl. TCP vagy UDP) segítségével kommunikál, és olyan csomagokat küld, amelyek a játékosok interakcióiról, játékeseeményekről és állapotfrissítésekről tartalmaznak információkat.

Végző sorban szerver ellenőriz minden interakciót és csak ő tud dönteni a változásokról. Ez azért jó, mert így a csalásokkal szemben is védekezik a játék.

Mivel mindig várni kell a szerver által küldött információra egy interakció esetén, így késleltetés léphet fel, főleg lassabb internet kapcsolat esetén. Erre egy módszer amit alkalmazni szoktak a klienseknél az úgynevezett predikció, ahol a helyi interakció alapján a kliens megpróbálja kitalálni mi fog ténylegesen történni. [6]

### 1.2.3. Peer-to-Peer (P2P)

A P2P architektúra lényege, hogy a játékosok közvetlen kapcsolatot létesítenek egymással, így nincs szükség központi szerverekre, ellentétben a Kliens-Szerver architektúrával. Ebben a felépítésben, minden játékosnak helyi szinten kezeli a rendszer a játéklógikát.



1.2. ábra. Peer-to-Peer architektúra

A P2P architektúra előnyei a Kliens-Szerver architektúrával szemben:

- Mivel a játékosok közvetlen kapcsolatot alakítanak ki egymással, így általában gyorsabb a válaszidő.
- Sokkal költség hatékonyabb megoldás, mivel nem kell egy központi szerver üzemeltetésére költeni.

Mivel ebben a felépítésben nincs egy központi szerver, ami ellenőrző szerepet is tölt be, így a csalások kiszűrése több gondot okozhat. Mint minden felépítésnek, ennek is vannak előnyei és hátrányai is. Ezt az architektúrát kisebb játékos létszám igényű játékoknál célszerű használni. A szakdolgozatomban során egy a Unity Technologies által létrehozott P2P architektúrán alapuló technológiát alkalmazok, amiről a 2.3. szekcióban bővebben írok. [7] [8]



## 2. fejezet

# Unity

A Unity egy játékmotor, amit aktívan a Unity Technologies fejleszt. A játékmotor segítségével számos platformra fejleszthetünk 2D és 3D játékokat is. Ezek a platformok közé tartoznak a virtuális valóságot szolgáltató platformok is. A megszokott asztali számítógépek mellett, mobil és web támogatást is nyújt a játékmotor. [9]

### 2.1. Unity6

Mai legfrissebb verziója a játékmotornak a Unity6, amit 2024 végén adtak ki. A frissítés számos nagy fejlesztéssel állt elő, amik közül nekem az egyik legfontosabb a tesztelés szempontjából, a *Multiplayer play mode* volt. Fontosabb fejlesztések:

- Renderelés teljesítményének növelése
- Magával ragadóbb vizuális megjelenítés
- Többjátékos játékok fejlesztésének egyszerűsítése
  - Multiplayer Center: kiválaszthatjuk, hogy milyen igényeink vannak a játékkal kapcsolatban, majd a Unity felsorolja nekünk a megfelelő technológiákat és csomagokat, így nem nekünk kell keresgelnünk és kutatnunk a csomagok után.
  - Multiplayer Play Mode: talán ez volt számomra a legfontosabb, hiszen exponenciális mértékben felgyorsította a tesztelés folyamatát. A Multiplay Play Mode lehetővé teszi, hogy egyszerre több példányban is futhasson a játék a gépemen buildelés nélkül.
- Web alapú fejlesztés teljesítményének növelése
- Gyorsított UI fejlesztés

Ezek mellett még számos fejlesztést és javítást hozott a frissítés. [10]

## 2.2. Felépítése

A játékok Unity-ben különböző jelenetekre (scene) vannak felbontva. A különböző jelenetekkel játékbeli tereket választhatunk el, például 1-1 szintet a játékon belül. A jelenetekben belül a játék fő építő eleme a *GameObject*, amit egy fa hierarchiában tudunk a jelenetekbe pakolni. [11]

### 2.2.1. GameObject

Unity-ben minden objektum egy *GameObject*. Ez azt jelenti, hogy mindennek, ami a játékban szerepelhet, *GameObject*-nek kell lennie. Önmagában egy *GameObject* nem csinál semmit, ezért tulajdonságokkal kell felruháznunk. A *GameObject* egy konténer, amihez darabokat adhatunk, amiktől hang, effekt, karakter vagy akármi lehet. Ezeket a darabokat komponenseknek nevezzük. [12]

#### Komponensek

A komponensek lehetnek beépítettek, amivel a Unity szolgál, vagy akár írhatunk saját komponenseket C# script-ek segítségével. Komponensek segítségével tudjuk életre kelteni a *GameObject*-eket. A saját script-ek írása lehetővé teszi a *GameObject*-en szereplő többi komponens módosítását és kezelését is. Komponenseket tudunk törölni és hozzáadni egy objektumhoz, ezeket a műveletek a 2.2.3. szekcióban említett inspector ablakon keresztül tudjuk kezelni. Ebben a szekcióban bemutatom részletesebben a leggyakrabban használt komponenseket. [13]

**Transform** ♦ Az egyik legfontosabb komponens, amivel minden objektumnak rendelkeznie kell. Ezt a komponenset nem is lehet törölni az objektumokról. Információt szolgáltat az objektum: pozíciójáról, méretéről, forgásáról és a hierarchiában lévő tulajdonságairól is. [14]

**Rigidbody** ♦ Azok az objektumok amelyeken van ilyen komponens, azok használják a beépített fizikai motort a gravitáció és többi fizikai hatás szimulálásához. A Transform komponens használata helyett, a fizikai motor segítségével módosítható az objektum pozíciója. Különböző fizikai hatásokat szimulálhatunk a komponens segítségével. Számos tulajdonságot is beállíthatunk az objektumnak, mint például a tömegét, a pontosabb szimulációk érdekében. [15]

**Camera** ♦ A játékos a kamerán keresztül fogja látni a világot. Egy projekt létrehozásakor alaphoz létrejön egy objektum, amire a Camera komponens már hozzá van adva. A komponensen különböző beállításokat találhatunk a felbontással, effektusokkal és hasonlókkal kapcsolatban. [16]

**Collider** ♦ Ez a komponens lehetőséget ad két objektum találkozásának vizsgálatára. Ha mozgatni is szeretnénk az objektumokat, akkor Rigidbody használatával tudjuk csak megtenni. Több különböző formában tudjuk az objektum köré tenni: [17]

- Box (3D)
- Square (2D)
- Sphere (3D)
- Circle (2D)
- Mesh (3D)
- Polygon (2D)

**Sprite Renderer** ♦ Ez a komponens lehetővé teszi a képek megjelenítését 2D illetve 3D projektekben is. Egy sprite-ot kell megadnunk neki, amit különböző képformátumokból a Unity-be importálva tudunk létrehozni. [18]

**Light** ♦ Ennek a komponensnek a segítségével tudjuk kezelni a fényeket a játékban. 2D és 3D esetén is több különböző light komponenst szolgáltat a Unity. A fényeknek lehetőségünk van állítani a fényét, színét, távolságát és még sok hasonló tulajdonságot. [19]

**Audio Source** ♦ Ennek a komponensnek egy Audio Clip-et szolgáltatva tudjuk lejátszani az adott hangunkat. Van lehetőségünk a hangerőt és hasonló tulajdonságokat módosítanunk. Használatához szükségünk van egy Audio Listener komponensre is, aminek segítségével akár térben is különbséget tehetünk a hangok között. [20]

**Script** ♦ Az egyik legfontosabb komponens, amivel személyre tudjuk szabni a játékmenetet. Mindegyik script megírása C#-ban történik. Az olyan script-ek amiket egy objektumhoz akarunk kötni a MonoBehaviour beépített osztályból kell örököltetnünk. Ez az osztály sok különböző metódust és függvényt ad, de ezek közül a 2 legfontosabb a start és az update metódusok.

- Start: a start metódus egyszer fut le, amikor az objektum betölt a játékba.
- Update: az update metódus minden frame-ben lefut, amikor az objektum aktív.

A megírt kódban különböző objektumokra és komponensekre is tudunk hivatkozni, amit a 2.2.3. szekcióban említett inspector ablakban kell be referálnunk. Lehetőségünk van másik script-re is hivatkozni. A játékfejlesztésben sok tervezési minta megtalálható az optimális kód írására, de a leggyakrabban használnak mondható az egyke (singleton) tervezési minta. [21]

## 2.2.2. Prefab

[22]

## 2.2.3. Editor

Minden ami nem kód írás egy játék fejlesztése során, az az editor-ban fog történni. A Unity editor több különböző ablakból tevődik össze, amik segítik a fejlesztési folyamatot. Ebben a szekcióban bemutatom részletesebben a legfontosabb ablakokat.



2.1. ábra. Unity editor

**Project Window** ♦ Ebben az ablakban tudjuk kezelni a fájlrendszerét a projektünknek. Itt hozhatunk létre rendszerezés céljából különböző mappákat, és helyezhetjük el a felhasználni kívánt fájlokat is. [23]

**Scene View** ♦ Ez az egyik legfontosabb ablak. Itt tudjuk kezelni a jeleneten belüli játék elemeinket. Több különböző módon tudjuk manipulálni az objektumokat, beleértve a pozíciójuk módosítását, és az átméretezésüket is. [24]

**Hierarchy Window** ♦ Ez az ablak tartalmazza a jelenlegi jeleneten belüli összes játék objektumot. Itt tudunk létrehozni új objektumokat, és a faszerű hierarchia nézetben rendezni őket. [25]

**Inspector Window** ♦ Ha kiválasztunk egy objektumot a hierarchiában, vagy a jelenet nézetben, akkor azt az objektumot ezen az ablakon keresztül tudjuk kezelni. Itt

tudunk új komponenseket adni hozzá és a meglévőeket pedig módosítani. Itt tudjuk például a saját kódjainkat is az objektumokra helyezni. [26]

**Game View** ♦ Ezen az ablakon látható a játékban használt kamerák által rögzített renderelt kép, amely a végső játékot mutatja be. [27]

## 2.3. Relay

A Relay szolgáltatás lehetőséget ad arra, hogy többjátékos játékokat fejlesszünk drága dedikált szerverek nélkül. A kapcsolódás folyamat hozzáférési kódokon keresztül történik a játékosok között. A Unity Technologies elmondása szerint ezt a szolgáltatást kis méretű casual játékokhoz és kör alapú kártya játékokhoz a legérdekesebb használni. Az alábbiak a fő előnyei ennek a szolgáltatásnak. [28] [29]

- Költséghatékony megoldás a játékosok összekapcsolásra: A Peer-to-Peer alapú megoldás lehetővé teszi, hogy a játékosok a saját eszközeiken hosztolják a játékot és így nincs szükség drága szerverekre.
- Automatikus és egyszerű skálázás: a Relay kezeli nekünk a skálázási igényeket, akár 100 játékost is képes összekapcsolni egy játékmenetbe.
- Biztonságos kapcsolat: a Relay kezeli helyettünk a biztonságos kapcsolatot és biztosítja az információk titkosított cseréjét a játékosok között.

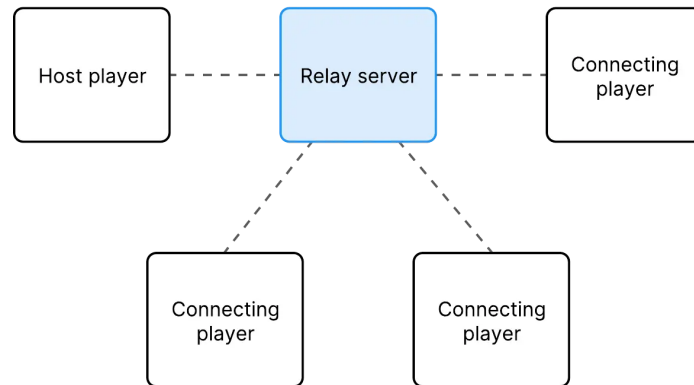
Két fő komponensből épül fel a Relay.

- Relay Server: az alacsony szintű Unity Transport réteghez kapcsolódik, hogy bájtokat küldjön az ügyfelek között.
- Relay Allocation Service: backenden működik, hogy a játékosok a hozzáférési kódok megosztásával játékmeneteket hozhassanak létre és csatlakozhassanak hozzájuk.

### 2.3.1. Relay Server

A játékosok nem közvetlen egymáshoz csatlakozva, hanem Relay szervereken keresztül kommunikálnak egymással. A Relay szerverek továbbítják az üzeneteket a kliensek között kis késleltetésű datagram cserével, ez biztosítja azt, hogy két játékos sose kapcsolódik közvetlen egymáshoz. Ezek a szerverek nyilvános végpontként viselkednek, ami elérhető az összes játékos számára. Ez a megközelítés megoldást ad több különböző felmerülő problémára is mint például a játékosok közötti tűzfalakra. Minden játékos ugyanarra az ip címre tud csatlakozni, amit biztosra vehetnek, hogy a játékmenet során

nem fog változni, így a játékosoknak nem kell tudniuk egymás ip címéről, ez is erősíti a biztonságot a rendszerben. [30]



2.2. ábra. Relay szerverek felépítése

### 2.3.2. Üzenet küldési protokoll

Üzenet típusok

## **3. fejezet**

# **Rendszerterv**

Azok az elemek, amiket tanultatok RFT-n, azok kerülnek ide

### **3.1. Rendszer célja**

### **3.2. Követelmények**

### **3.3. Architektúrális terv**

PC unity, Mobile unity, SharedDLL

### **3.4. Használt fejlesztői eszközök**

## 4. fejezet

# Saját szoftver megvalósítása

### 4.1. Texas Hold’Em

A póker a világ egyik legismertebb kártyajátéka, amelyben a játékosok célja, hogy a saját lapjaikból a lehető legjobb kombinációt kialakítva megszerezzék az asztalon lévő kasszát. A póker számos különböző szabályrendszerrel rendelkező változatban létezik.

A *Texas Hold’Em* a közösségi pókerjátékok legnépszerűbb formája, amelyet jellemzően 2 és 10 játékos között játszanak. Ez egy viszonylag zárt struktúrájú játék, ahol a licitálás menete állandó szabályok szerint zajlik.

#### 4.1.1. Szabályok ismertetése

Ebben a fejezetben a póker legelterjedtebb, hivatalos versenyeken is alkalmazott szabályrendszere kerül bemutatásra. Ami közös az összes variációban, hogy a játékot 52 lapos francia kártyával játsszák dzsókerek nélkül.

#### Játék menete

A játék során három fontos szerep forog körbe a játékosok között, amit „gombokkal” jelölünk. Ezek a szerepek az osztó, kis vak és nagy vak. Az osztótól balra ülő játékos lesz a kis vak, a kis vaktól balra ülő pedig a nagy vak. Az osztót pedig több különböző módon választhatjuk meg a játék elején.

Az osztó keveri és osztja ki a lapokat a szabályok szerint. A vakok pedig még osztás előtt kötelesek betenni a vak téteket, ahol a kis vak tét általában a nagy vak tét fele. Egy játszma az alábbiak szerint játszódhat le: [33]

##### 1. Osztás

- Az osztó először megkeveri a paklit. A kiosztás előtt a kis vak és a nagy vak beteszik a kötelező téteket. Ezt követően az osztó balról kezdve, két körben, egyesével oszt minden játékosnak egy-egy zárt lapot.



## 2. *Pre-Flop (első licitkör)*

- A licitálás a nagy vaktól balra ülő első játékoskal kezdődik, aki az alábbi lehetőségek közül választhat:
  - Tartás – megadja az aktuális tétet.
  - Emelés – növeli a tétet a limitszabályok szerint.
  - Dobás – eldobja a lapjait, ezzel kiszáll a játékból.
- A licitálás az óramutató járásával megegyező irányban halad tovább.

## 3. *Flop (második licitkör)*

- Az osztó egy lapot félretesz égető lapként, majd három közös lapot felfordítva az asztal közepére helyez.
- A licitálást az osztógombtól balra ülő első aktív játékos kezdi, és az alábbi lehetőségek közül választhat:
  - Passzolás – nem emel, de marad a játékban.
  - Nyitás – tétet tesz be a limitszabályok szerint.
  - Dobás – eldobja a lapjait és kiszáll a körből.
- Ha valaki nyit, a többiek dönthetnek:
  - Tartás – megadják a tétet.
  - Emelés – növelik a tétet.
  - Dobás – kiszállnak a körből.

## 4. *Turn (harmadik licitkör)*

- Az osztó ismét éget egy lapot, majd egy negyedik közös lapot felfordítva az asztalra helyez.
- A harmadik licitkör a második licitkörhöz hasonlóan zajlik.

## 5. *River (negyedik licitkör)*

- Az osztó még egy égető lapot félretesz, majd kiosztja az utolsó, ötödik közös lapot.
- Minden játékos hét lapból próbálja a lehető legjobb ötlapos kombinációt kialakítani.
- Az utolsó licitkör a második és a harmadik licitkörhöz hasonlóan zajlik.

## 6. *Showdown (lapok felfedése)*

- Ha az utolsó licitkör után egynél több játékos marad, akkor a játékosok megmutatják a lapjaikat választásuk szerint.
- A kasszát a legerősebb pókerkezet birtokló játékos nyeri el.

## Pókerkezek

Az alábbi felsorolás a lehetséges pókerkezeket mutatja be, amelyeket erősségük szerint rendeztem el, a legerősebbtől a leggyengébbig. A lista tetején található kéz a pókerben elérhető legmagasabb értékű kombináció, és innen lefelé haladva egyre gyengébb kezek következnek. Az alábbi sorrend megtekinthető a 4.1. ábrán is. [35, 9. oldal]

### 1. *Royal flös (royal flush)*

- A legerősebb lapkombináció. Egyszínű 10-es, bubi, dáma, király, ász lapokból áll. Ha két ilyen találkozik, akkor döntetlen<sup>1</sup> van.

### 2. *Színsor (straight flush)*

- Öt egyszínű sorba rendezhető lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, akkor döntetlen van.

### 3. *Póker (four of a kind)*

- Négy ugyanolyan számozású vagy jelű lapból és egy akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb póker nyer.

### 4. *Full (full house)*

- Három ugyanolyan számozású vagy jelű lapból és két másik ugyanolyan számozású vagy jelű lapból áll. Ha két ilyen találkozik, a magasabb drill nyer. Ha egyforma, a magasabb pár nyer.

### 5. *Szín (flush)*

- Öt ugyanolyan színű lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, a második legmagasabb dönt, és így tovább...

### 6. *Sor (straight)*

- Öt sorba rendezhető lapból áll. Ha két ilyen találkozik, a legmagasabb lap dönt. Ha egyforma, a színerősség dönt.

### 7. *Drill (three of a kind)*

- Három ugyanolyan számozású vagy jelű lapból és két akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb drill nyer. Ha egyforma, a magasabb semleges lap, majd az alacsonyabb dönt.

### 8. *Két pár (two pairs)*

---

<sup>1</sup> Osztozás történik a nyeremény között.

- Kétszer két ugyanolyan számozású vagy jelű lapból áll. Ha két ilyen találkozik, a magasabb pár, majd az alacsonyabb, majd a semleges lap erőssége dönt.

#### 9. Egy pár (one pair)

- Két ugyanolyan számozású vagy jelű lapból és három akármilyen másik lapból áll. Ha két ilyen találkozik, a magasabb pár nyer. Ha egyforma, a semleges lapok döntenek.

#### 10. Magas lap (high card)

- Bármilyen lap, abból is a legmagasabb értékkel rendelkező.



4.1. ábra. Lehetséges pókerkezek

### 4.1.2. Használt kézkiértékelő algoritmus

A legelterjedtebb kézkiértékelő algoritmusok azon alapulnak, hogy a kezek értékeit egy előre kiszámított kéz értékeket tároló táblából keressük ki. Ez az egyik leggyorsabb módszer, viszont a módszer egy nagy hátránya a nagy méretű tábla tárolása, amely tárhely igényes. A játékomban egy bit matematikán alapuló algoritmust használtam, aminek alapjait a [34] blog adta. Ez a módszer lassabb, mint a táblás, viszont a tárhely igényes probléma itt megszűnik.

#### Algoritmus

Az alábbi lépéseket kell végrehajtanunk a kézkiértékeléshez:

1. Minden kártyáról el kell tárolnunk a rangját és a színét. Az inputunk 5 darab ilyen kártyából fog állni.

2. 2 különböző bitmező létrehozása a kártyák alapján

- Az első mező a kártyák rangjának előfordulásáról tartalmaz 15 biten információt. Az egyes biteken azt fogjuk jelölni, hogy az adott rangból van-e az 5 adott kártya között (1 ha van, 0 ha nincs). A 4.1 táblázatban és a 4.2 táblázatban látható 1-1 példa az elő állított bitmezőről.

0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
A	K	Q	J	T	9	8	7	6	5	4	3	2		

4.1. táblázat. [9, 10, J, Q, K] kártyákhoz tartozó első bitmező

0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
A	K	Q	J	T	9	8	7	6	5	4	3	2		

4.2. táblázat. [9, 9, 9, K, K] kártyákhoz tartozó első bitmező

- A második mező a rangok előfordulásának számáról tárol információt 60 biten. Minden ranghoz rendelünk 4 bitet, ahol annyit jelölünk 1-el, amennyi darabunk van az adott rangból. A 4.3 táblázatban és a 4.4 táblázatban látható 1-1 példa az elő állított bitmezőről.

0000	0001	0001	0001	0001	0001	0000	0000	0000	0000	0000	0000	0000	0000	0000
A	K	Q	J	T	9	8	7	6	5	4	3	2		

4.3. táblázat. [9, 10, J, Q, K] kártyákhoz tartozó második bitmező

0000	0011	0000	0000	0000	0111	0000	0000	0000	0000	0000	0000	0000	0000	0000
A	K	Q	J	T	9	8	7	6	5	4	3	2		

4.4. táblázat. [9, 9, 9, K, K] kártyákhoz tartozó második bitmező

3. kéz értékelés a második bitmező segítségével

- A második bitmező 15-el való maradékos osztásának segítségével kitudunk értékelni 6 különböző pókerkezet. Először átszámoljuk a második bitmezőt 10-es számrendszerbe, majd utána végezzük el a 15-el való maradékos osztást. Pár példa erre:

- Póker (four of a kind): [9, 9, 9, 9, K] kártyákhoz tartozó második bináris mező 10-es számrendszerbe átszámolva: 4504630419521536. Ez az érték maradékosan osztva 15-el egyenlő lesz 1-el. Minden lehetséges ilyen pókerkézhez tartozó érték 1-et fog visszaadni erre az osztásra.

- Full (full house): Mindig 10-et fog visszaadni az osztás eredményeképpen.
- Drill (three of a kind): Mindig 9-et fog visszaadni az osztás eredményeképpen.
- Két pár (two pairs): Mindig 7-et fog visszaadni az osztás eredményeképpen.
- Egy pár (one pair): Mindig 6-ot fog visszaadni az osztás eredményeképpen.
- Magas lap (high card): Mindig 5-öt fog visszaadni az osztás eredményeképpen.

A varázslat az, hogy mind a hat pókerkéz a fenti módszerrel történő számolás mindig ugyanazt az eredményt adja, függetlenül attól, hogy mi a tényleges öt lap. Amit a maradékos osztás 15-el csinál, az egyenértékű az egyes 4 bites kártyák értékének összegzésével. Tehát a  $[2, 2, 2, 3, 3]$  kártyák esetében ez pl.  $0111 + 0011$  amit binárisan összeadva, majd átszámolva 10-et kapunk. És ez mind a 6 pókerkéznél így fog működni.

#### 4. sorok ellenőrzése

- A *sorok* ellenőrzéséhez az első bitmezőt fogjuk felhasználni. Az első bitmezőn bitenkénti ÉS műveletet alkalmazunk önmagának negatívjával. Ennek eredményeképpen megkapjuk az LSB-t, ami a bináris szám ábrázolásának legjobb oldali bitjére utal. Majd ezután, ha elosztjuk az eredeti bitmezőt az LSB-vel, akkor megtudjuk állapítani, hogy a pókerkéz *sor*-t tartalmaz-e. Ha az osztás eredménye pontosan egyenlő 11111-el, azaz decimálisan 31-el, akkor *sorunk* van, ellenkező esetben pedig nincs *sorunk*. Ez a módszer minden *sorra* működni fog, kivéve az *ász-alacsony sorra*, amit egy külön ellenőrzéssel megtudunk állapítani. Ha az első bitmezőnk pontosan egyenlő az alábbi bitmezővel: 100000000111100, akkor *ász-alacsony sorunk* van.

#### 5. flush ellenőrzése

- A *flush* ellenőrzésénél nincs szükségünk bit matematikára, szimplán csak azt kell megnéznünk, hogy a kapott 5 input kártya közül, mindnek meggyezik a színe. Ha mindegyik ugyanolyan színű, akkor a pókerkéz *flush*, ellenkező esetben pedig nem. Ha *flush* kezünk van és az előző ellenőrzés szerint *sorunk* is van, akkor *straight flush* kezünk van. Az első bitmező segítségével pedig tudjuk ellenőrizni a *royal flush*-t is. Hasonló módon mint az *ász-alacsony sornál* itt is az első bitmezőt kell ellenőriznünk. Ha az első bitmezőnk pontosan egyenlő az alábbi bitmezővel: 111110000000000, akkor *royal flush* pókerkezünk van.

## 6. döntetlenek eldöntése

- Abban az esetben, ha két pókerkéz azonos típusú, a döntetleneket úgy döntjük el, hogy az 5 kártyát először az előfordulás sorrendje, majd a rangja szerint rendezzük, és bitek eltolásával összehasonlítható pontszámot hozunk létre az alábbiak szerint:

- Az első kártya értékét kettes számrendszerben eltoljuk balra 16-al
- A második kártya értékét kettes számrendszerben eltoljuk balra 12-vel
- A harmadik kártya értékét kettes számrendszerben eltoljuk balra 8-cel
- A negyedik kártya értékét kettes számrendszerben eltoljuk balra 4-el
- Az ötödik kártya értékét nem toljuk el

Ezután kombináljuk mind az öt létrejött számot bitenkénti VAGY művelettel és ennek az eredménye lesz a döntetleneket eldöntő pontszáma a pókerkéznek, ahol minél magasabb ez a pontszám, annál jobb kezünk van.

A teljes algoritmus tehát röviden így működik:

1. Eltároljuk az 5 kártyánkról a szükséges szín és rang adatokat.
2. A második bitmezőn maradékos osztást használva ellenőrizzük, hogy póker, full, drill, két pár, egy pár vagy magas lap pókerkezünk van-e.
3. Az első bitmező LSB-vel történő osztásával ellenőrizzük a sorokat, majd elvégezzük az ász-alacsony sorok extra ellenőrzését.
4. Ellenőrizzük, hogy van-e öt azonos színű lap a flush-höz, beleértve a royal flush extra ellenőrzését is.
5. A döntetleneket felbontjuk a pókerkezekhez kiszámolt bizonyos pontszámok összehasonlításával.

## 4.2. Többjátékos kapcsolat megvalósítása

Messagek küldése stringként JSON formában, amiket C#-ban osztályként kezelünk

### 4.2.1. Kapcsolat kezelése a PC oldalon

Joincode létrehozása majd lobby panelen megmutatása lobby, kick sendmessage to clients or to client

### 4.2.2. Kapcsolat kezelése a Mobile oldalon

Csatlakozás join code alapján chat, ready, disconnect sendmessage to host

## **4.3. Játékmenet megvalósítása**

### **4.3.1. Modul1**

### **4.3.2. Modul2**

## 5. fejezet

# Tesztelés

A tesztelés nagyon fontos része a fejlesztésnek, hiszen ennek a folyamatnak a során tárjuk fel a lehetséges hibákat a programunkban. A tesztek nagy szerepet játszanak az egyszerűen bővíthető és fejleszthető kódbázisokban, hiszen így minden módosítás és bővítés után letudjuk ellenőrizni, hogy van-e hiba a kódban. A legelterjedtebb tesztelési forma az *egység tesztek* írása, ahol 1-1 metódusnak vagy függvénynek több különböző bemenet alapján vizsgáljuk a várt kimenetét. Az egység tesztek során le kezeljük a helyes bemenetre való működést és az akár hibát kiváltó bemenetre való működést is. Az egyik legfontosabb tesztelési módszer játékfejlesztés szempontjából a játék teszt (playtest), ahol a játékot minél körütekintőbben játszva tárjuk fel az esetleges hibákat. A tesztelés folyamatát az is segítheti egy ekkora projektnél, hogy megpróbálunk minél tisztább és rendezettebb kódot írni, így az egység teszteket is egyszerűbb dolgunk lesz megírni.

### 5.1. Közös DLL tesztelése

A közös DLL tesztelése során a már fentebb említett egység teszteket alkalmaztam. A kódban lévő összes metódusra és függvényre írtam egység teszteteket, minden lehetséges eset tesztelésére. Ezeknek a teszteknek a segítségével könnyen tudtam bővíteni és ellenőrizni a kódbázist. Mint, ahogy korábban a 3.3. szekcióban említettem a közös DLL projektnél az egyik legfontosabb része a kézkiértékelést kezelő algoritmus. A kézkiértékeléshez írt egység teszteknél különösen figyeltem arra, hogy minden lehetséges pókerkéz kombinációt használjak bemenetként a teszteknél.

### 5.2. Játékok tesztelése

A játékok tesztelésében legnagyobb szerepet a playtest játszott. A playtest kétféleképpen nézett ki:



- Barátokkal: a ki buildelt Unity játékokat élesben teszteltük közösen.
- Egyedül: Unity editor által adott lehetőségeket kihasználva teszteltem. Ez azt jelentette, hogy a PC játék 1 példányban futott a gépemen, a mobil játék pedig a Unity által szolgáltatott *Multiplayer play mode*-nak köszönhetően egyszerre 2-4 példányban futott a gépemen. [37]

A playtest-ek során mindig odafigyeltem, hogy minden lehetőséget kipróbáljak a játékban. Ezek a playtest-ek nem csak hibakeresésre de játékélménybeli javításába is sokat segítettek, hiszen a barátaim sok dolgot észrevettek és jeleztek felém, amik a játékélményt segíthetik. Későbbiekben készítettem egy naplózó rendszert is a játékokhoz, ami megkönnyítette a hibák feltárását.

Mindkét játékhoz készítettem a már korábbiakban említett egység teszteket is, amikhez az NUnit könyvtárat használtam. Játékfejlesztés során körülményes lehet az egység tesztek használata, hiszen nagyon sokszor egy adott kódrészlet akár több másik komponenstől is függhet, ennek megoldására úgynevezett mock komponenseket hozunk létre. [38]

### 5.3. Összefoglalás

A tesztelés folyamata lehet körülményes és néhány helyen érződhet úgy hogy csak idő pazarlás, de mindenképpen megéri időt tölteni vele. A játékfejlesztésben pedig a hibakeresésen kívül másban is fontos szerepet játszik, hiszen akár a játékélmény fejlesztésében is segíthet. Vannak olyan játékok is, ahol a játékmenet kiegyensúlyozásának érdekében is rákényszerülünk a tesztelésre és elhagyhatatlan eleme a fejlesztése folyamatnak.

# Összegzés

# Irodalomjegyzék

- [1] WIKIPEDIA: *Game engine*, Wikipedia, az online enciklopédia, elérhető: [https://en.wikipedia.org/wiki/Game\\_engine](https://en.wikipedia.org/wiki/Game_engine) [Letöltve: 2025-02-28]
- [2] WIKIPEDIA: *Unreal Engine*, Wikipedia, az online enciklopédia, elérhető: [https://en.wikipedia.org/wiki/Unreal\\_Engine](https://en.wikipedia.org/wiki/Unreal_Engine) [Letöltve: 2025-02-28]
- [3] GITHUB: *Godot*: <https://github.com/godotengine/godot>) [Letöltve: 2025-02-28]
- [4] WIKIPEDIA: *Unity (game engine)*, Wikipedia, az online enciklopédia, elérhető: [https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine)) [Letöltve: 2025-02-28]
- [5] JAMAL ALADDIN: *The Evolution of Multiplayer Gaming: A Journey Through Time*, Jamal Aladdin blogja, elérhető: [https://medium.com/@Jamal\\_Aladdin/the-evolution-of-multiplayer-gaming-a-journey-through-time-e34ef59294c2](https://medium.com/@Jamal_Aladdin/the-evolution-of-multiplayer-gaming-a-journey-through-time-e34ef59294c2) [Letöltve: 2025-02-28]
- [6] LEM APPERSON: *Beginning Game Development: Client-Server Architecture*, Lem Apperson blogja, elérhető: <https://medium.com/@lemapp09/beginning-game-development-client-server-architecture-1b7676d80dea> [Letöltve: 2025-02-28]
- [7] WIKIPEDIA: *Peer-to-peer*, Wikipedia, az online enciklopédia, elérhető: <https://hu.wikipedia.org/wiki/Peer-to-peer> [Letöltve: 2025-03-02]
- [8] TASHI PROTOCOL: *Peer-to-Peer Gaming*, Tashi Protocol blogja, elérhető: <https://medium.com/tashi-gg/peer-to-peer-gaming-9991600c6707> [Letöltve: 2025-03-02]
- [9] UNITY TECHNOLOGIES: *Platform development*, elérhető: <https://docs.unity3d.com/Manual/PlatformSpecific.html> [Letöltve: 2025-03-03]
- [10] MARTIN BEST: *Unity 6 is here: See what's new*, elérhető: <https://unity.com/blog/unity-6-features-announcement> [Letöltve: 2025-03-03]

- [11] UNITY TECHNOLOGIES: *Scenes*, elérhető: <https://docs.unity3d.com/560/Documentation/Manual/CreatingScenes.html> [Letöltve: 2025-03-03]
- [12] UNITY TECHNOLOGIES: *GameObjects*, elérhető: <https://docs.unity3d.com/560/Documentation/Manual/GameObjects.html> [Letöltve: 2025-03-03]
- [13] UNITY TECHNOLOGIES: *Using Components*, elérhető: <https://docs.unity3d.com/560/Documentation/Manual/UsingComponents.html> [Letöltve: 2025-03-03]
- [14] UNITY TECHNOLOGIES: *Transform*, elérhető: <https://docs.unity3d.com/560/Documentation/Manual/class-Transform.html> [Letöltve: 2025-03-03]
- [15] UNITY TECHNOLOGIES: *RigidBody*, elérhető: <https://docs.unity3d.com/Manual/class-Rigidbody.html> [Letöltve: 2025-03-03]
- [16] UNITY TECHNOLOGIES: *Camera*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Camera.html> [Letöltve: 2025-03-03]
- [17] UNITY TECHNOLOGIES: *Collider*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Collider.html> [Letöltve: 2025-03-03]
- [18] UNITY TECHNOLOGIES: *Sprite Renderer*, elérhető: <https://docs.unity3d.com/540/Documentation/Manual/class-SpriteRenderer.html> [Letöltve: 2025-03-03]
- [19] UNITY TECHNOLOGIES: *Light*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Light.html> [Letöltve: 2025-03-03]
- [20] UNITY TECHNOLOGIES: *Audio Source*, elérhető: <https://docs.unity3d.com/Manual/class-AudioSource.html> [Letöltve: 2025-03-03]
- [21] UNITY TECHNOLOGIES: *Scripting*, elérhető: <https://docs.unity3d.com/Manual/scripting.html> [Letöltve: 2025-03-03]
- [22] UNITY TECHNOLOGIES: *Prefabs*, elérhető: <https://docs.unity3d.com/Manual/Prefabs.html> [Letöltve: 2025-03-03]
- [23] UNITY TECHNOLOGIES: *The Project Window*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/ProjectView.html> [Letöltve: 2025-03-03]
- [24] UNITY TECHNOLOGIES: *The Scene View*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/UsingTheSceneView.html> [Letöltve: 2025-03-03]
- [25] UNITY TECHNOLOGIES: *The Hierarchy Window*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/Hierarchy.html> [Letöltve: 2025-03-03]

- [26] UNITY TECHNOLOGIES: *The Inspector Window*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/UsingTheInspector.html> [Letöltve: 2025-03-03]
- [27] UNITY TECHNOLOGIES: *The Game View*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/GameView.html> [Letöltve: 2025-03-03]
- [28] UNITY TECHNOLOGIES: *Relay: Free P2P Networking & Connection Solution*, elérhető: <https://unity.com/products/relay> [Letöltve: 2025-03-04]
- [29] UNITY TECHNOLOGIES: *Unity Relay*, elérhető: <https://docs.unity.com/ugs/manual/relay/manual/introduction> [Letöltve: 2025-03-04]
- [30] UNITY TECHNOLOGIES: *Relay Servers*, elérhető: <https://docs.unity.com/ugs/en-us/manual/relay/manual/relay-servers> [Letöltve: 2025-03-04]
- [31] UNITY TECHNOLOGIES: *Allocations Service*, elérhető: <https://docs.unity.com/ugs/en-us/manual/relay/manual/allocations-service> [Letöltve: 2025-03-04]
- [32] WIKIPEDIA: *Póker*, Wikipedia, az online enciklopédia, elérhető: <https://hu.wikipedia.org/wiki/P%C3%B3ker> [Letöltve: 2024-11-12]
- [33] WIKIPEDIA: *Texas Hold'Em*, Wikipedia, az online enciklopédia, elérhető: [https://hu.wikipedia.org/wiki/Texas\\_Hold%E2%80%99Em](https://hu.wikipedia.org/wiki/Texas_Hold%E2%80%99Em) [Letöltve: 2024-11-13]
- [34] JONATHAN HSIAO: *Evaluating Poker Hands with Bit Math*, Jonathan Hsiao blogja, elérhető: <https://jonathanhsiao.com/blog/evaluating-poker-hands-with-bit-math> [Letöltve: 2025-02-25]
- [35] SZURDI ANDRÁS: *Pókerkönyv. Kezdőknek és haladóknak*, Ciceró, Budapest, 1995.
- [36] VARGA ERVIN: *Póker alapkönyv*, Vagabund, Kecskemét, 2008.
- [37] UNITY TECHNOLOGIES: *Multiplayer Play Mode*, elérhető: <https://docs.unity3d.com/6000.0/Documentation/Manual/com.unity.multiplayer.playmode.html> [Letöltve: 2025-03-03]
- [38] NUNIT: a unit-testing framework for all .Net languages, elérhető: <https://docs.nunit.org/> [Letöltve: 2025-03-02]