

VÁCI SZAKKÉPZÉSI CENTRUM
BORONKAY GYÖRGY MŰSZAKI TECHNIKUMA
ÉS GIMNÁZIUMA

SZAKDOLGOZAT

Bakó Borka
Bali István Gábor
Katona Bálint

2025.

Váci Szakképzési Centrum
Boronkay György Műszaki Technikuma és
Gimnáziuma

VIZSGAREMEK

Arcalite

Konzulens:
Gyombolainé Cserny Zsuzsanna

Készítők:
Bakó Borka
Bali István Gábor
Katona Bálint

Vác
2025

Hallgatói nyilatkozat

Alulírottak, ezúton kijelentjük, hogy a szakdolgozat saját, önálló munkánk, és korábban még sehol nem került publikálásra. Szakdolgozatunk a Váci Szakképzési Centrum Boronkay György Műszaki Technikum és Gimnázium szoftverfejlesztő és -tesztelő technikus képzésén készítettük. Tudomásul vesszük, hogy szakdolgozatunkat a Váci Szakképzési Centrum Boronkay György Műszaki Technikum és Gimnázium tárolja.

.....

Bakó Borka

.....

Bali István Gábor

.....

Katona Bálint

Konzultációs lap

Vizsgázók neve:

- Bakó Borka
- Bali István Gábor
- Katona Bálint

Szakedolgozat címe: **Arcalite**

Program által nyújtott szolgáltatások:

- Platformer játék
- A játékbeli haladás nyomonkövetése a weboldalon
- Tudástár a játékban lévő ellenfelekhez és tárgyakhoz
- E-mailek (visszaigazoló, jelszó módosítás)

Sorszám	A konzultáció időpontja	Konzulens aláírása
1.	2024. október 11.	
2.	2024. november 15.	
3.	2024. december 13.	
4.	2025. január 17.	
5.	2025. február 14.	
6.	2025. március 14.	

A szakdolgozat beadható:

Vác, 2025.

.....

Konzulens

A szakdolgozatot átvettem:

Vác, 2025.

.....

A szakképzést folytató
intézmény felelőse

Tartalomjegyzék

Hallgatói nyilatkozat	3
Konzultációs lap	4
Tartalomjegyzék	5
1. Témaválasztás	7
2. Fejlesztői dokumentáció	8
2.1. Fejlesztői környezet	8
2.1.1. Visual Studio Code	8
2.1.2. XAMPP.....	8
2.1.3. Visual Studio.....	8
2.1.4. GODOT	8
2.2. Használt technológiák, nyelvek	9
2.2.1. HTML	9
2.2.2. CSS	9
2.2.3. JavaScript.....	9
2.2.4. jQuery	9
2.2.5. JSON.....	9
2.2.6. Bootstrap.....	10
2.2.7. PHP	10
2.2.8. HTTP, REST API	10
2.2.9. MySQL	10
2.2.10. C#.....	11
2.2.11. Git, GitHub	11
2.2.12. ChatGPT	11
2.3. Adatszerkezet	12
2.3.1. Alapadatok	12
2.3.2. Kapcsolati ábra	12
2.3.3. Táblák	13
2.4. A weboldal felépítése.....	21
2.4.1. A weboldal szerkezete és formázás	21
2.4.2. Saját felugróablakok	21
2.4.3. Az API és a weboldal közti kommunikáció.....	22
2.4.4. Egyéb funkcionalitás a weboldalon	22
2.5. Az API felépítése	23

2.5.1. A cél.....	23
2.5.2. Hibakezelés és közös metódusok.....	23
2.5.3. A végpontok.....	24
2.6. Az alkalmazás felépítése.....	35
2.6.1. Játék design, logika.....	35
2.6.2. Kód struktúra, architektúra	36
2.6.3. Osztályok és Scriptek.....	51
2.6.4. Függőségek	92
2.7. Tesztelés.....	93
2.7.1. Backend tesztelés	93
2.7.2. Frontend tesztelés	98
2.7.3. A játék tesztelése	99
2.8. Továbbfejlesztési lehetőségek	100
2.8.1. Játék	100
2.8.2. Weboldal	100
3. Felhasználói dokumentáció.....	101
3.1. A program rövid ismertetése.....	101
3.1.1. Játék	101
3.1.2. A weboldal	101
3.2. A program feltelepítése.....	102
3.2.1. Feltételek.....	102
3.2.2. A projekt letöltése.....	102
3.2.3. A játék telepítése.....	102
3.2.4. A weboldal és adatbázis üzembe helyezése.....	103
3.3. Használati útmutató	110
3.3.1. Játék és weboldal csatlakozása	110
3.3.2. A játék irányítása	110
3.3.3. A weboldal használata	116
4. Források és linkek	121
5. Irodalomjegyzék	122

1. Témaválasztás

A feladatunk egy asztali alkalmazás, egy weboldal, és egy ezekkel kommunikáló adatbázis elkészítése volt.

Az asztali alkalmazás témájához sok különféle ötletünk támadt, de egyikük sem ragadta meg az érdeklődésünk. Ám egyszer felmerült egy kérdés: „Lehet-e játékot készíteni az asztali alkalmazásunknak?”. Helyeslő választ kapván már el is dőlt, hogy egy játék lesz a projektünk középpontjában. Ezt a döntést nagyban elősegítette, hogy a csapat tagjai is érdekeltek a játékfejlesztésben, hárunk közül ketten is ebben a szakirányban folytatják tanulmányaikat.

A játék jellege és témája volt a következő kérdés. Az általunk ismert játékok jellegét, kinézetét és játékmódját kezdtük el böngészni ötletekért, mérlegelve, hogy mit lennének képesek határidőre, jó minőségben elkészíteni. Választásunk a játék jellegét illetően a „platformer” stílusra esett. Ilyen jellegű játékokban tipikusan különféle „platformokon” ugrálva kell haladnunk, közben ellenségekkel harcolnunk stb. Kinézetre a kétdimenziós, oldalnézetes stílust választottuk, mivel ezzel már eleinte is rengeteg ötletünk támadt a játék megjelenését illetően. Játékunk főszereplőjeként egy mágust választottunk, akinek a különböző pályákon átkelve, szörnyeket legyőzve kell utat találnia a végső céljához. Ez a cél egyelőre titok, a játékot kijátszva viszont megtudhatjuk!

Már csak a weboldal kérdése maradt hátra. Átgondolva, hogy egy számítógépes játékhoz milyen weboldal illik, két célt adtunk az oldalnak: egy tudástár biztosítása a játékról, illetve felhasználói fiókunk kezelése. A tudástár egy egyedibb megoldást kapott: csak arról tudhatunk meg információkat, amivel már találkoztunk. Épp ezért, illetve játékunk akár más eszközről való folytatása érdekében gondoltuk ideillőnek a felhasználói fiókok implementálását. Az oldalon tudunk regisztrálni, bejelentkezni, kezelni a fiókunk, és a játékba bejelentkezve képesek vagyunk folytatni akár máshol elkezdett vagy folytatott játékunkat is.

2. Fejlesztői dokumentáció

2.1. Fejlesztői környezet

2.1.1. Visual Studio Code

A Visual Studio Code (VSC) egy nyílt forráskódú kódszerkesztő alkalmazás. Nagy mértékben testre szabható, illetve rengeteg kiegészítő tartalom készült hozzá, melyekkel könnyen fejleszthetünk a legtöbb programozási nyelven. Mi a weboldal HTML vázához, formázásához, a funkcionalitást biztosító JavaScript szkriptekhez, illetve az adatbázissal kommunikáló PHP RestAPI fejlesztéséhez használjuk.

2.1.2. XAMPP

A XAMPP egy nyílt forráskódú és platformfüggetlen webservert-szoftver csomag. A csomag fő elemei – amelyek miatt mi is használjuk – az Apache szerver, PHP támogatás, a MariaDB adatbázis-motor, illetve a phpMyAdmin adatbázis-kezelőfelület.

2.1.3. Visual Studio

A Visual Studio egy fejlesztői kódszerkesztő, amit kód szerkesztésére, debug-olására, de akár alkalmazásunk publikálására is használhatunk. Az IDE a standard kódszerkesztésen és debug-oláson felül tartalmaz fordítókat, kódkiegészítő, grafikus tervező, és még sok egyéb hasznos eszközzel, melyek elősegítik a fejlesztési élményünket. Ebben a projektben a játék kódjának megírásához használtuk.

2.1.4. GODOT

A Godot egy játékfejlesztő motor, amellyel egy egységes felületen készíthetünk akár két-, akár háromdimenziós platformfüggetlen játékokat. A motor egy mindent átfogó eszköz-csomagot kínál, hogy a fejlesztők a játék fejlesztésére fókuszálhassanak, és ne kelljen már megoldott problémákat újból megoldani. A program teljesen ingyenes és nyílt forráskódú, többnyire a közösség által fejlesztett.

2.2. Használt technológiák, nyelvek

2.2.1. HTML

A HTML (HyperText Markup Language) egy struktúrált leírónyelv, a weboldalak szerkezetének kialakításához használt szabvány. Általában együtt használatos a CSS és JavaScript nyelvekkel, mely három a webfejlesztés alapjaként szolgál. Míg a HTML a struktúra kialakítására használják, a CSS-t a formázásra, a JavaScript-et pedig a funkcionalitás kialakítására használják.

2.2.2. CSS

A CSS (Cascading Style Sheets) egy stíluslapnyelv, mellyel a weboldalak megjelenítését és formázását alakíthatjuk ki. Lehetővé teszi, hogy a fejlesztők a formázást és a struktúrát elválasszák egymástól, továbbá, hogy a weboldalak különféle környezetekben is egységesen jelenjenek meg.

2.2.3. JavaScript

A JavaScript egy dinamikus szkriptnyelv, amelyet általában weboldalak fejlesztéséhez használnak. Az egyik legelterjedtebb és legjelentősebb programozási nyelv a webfejlesztésben. Lehetővé teszi, hogy a weboldal interaktív legyen és dinamikusan reagáljon a felhasználó bemenetére. Emellett teljes értékű alkalmazásokat, játékokat is lehet benne készíteni, ilyenkor a weboldal egy megjelenítő szerepet vet fel. Mi a JavaScriptet a weboldal interaktív részének kialakításán felül az adatok API-ból történő lekérésére és betöltésére használjuk.

2.2.4. jQuery

A jQuery egy könnyűsúlyú, gyors és keresztplatformos JavaScript könyvtár, amelyet főleg webfejlesztés során használnak. Célja, hogy egyszerűbbé és hatékonyabbá tegye a JavaScript alapú webfejlesztést. Népszerűsége és kényelme miatt sok webfejlesztő használja az általános feladatok megkönnyítésére és az alkalmazások gyors fejlesztésére.

2.2.5. JSON

A JSON (JavaScript Object Notation) egy könnyen olvasható adatsere formátum, amely gépi feldolgozásra alkalmas. A JSON adatokat egy szöveges formátumban ábrázolja, és adatstruktúrákat, például objektumokat és tömböket képes reprezentálni. Mi az API-ban vesszük hasznát, mivel mind a PHP, mind a JavaScript támogatja a formátumot, beépített funkcióival könnyen átalakítható és kezelhető a továbbított adat.

2.2.6. Bootstrap

A Bootstrap egy ingyenes és nyílt forráskódú frontend keretrendszer weboldalak és webalkalmazások fejlesztéséhez. Elsősorban HTML, CSS és JavaScript alapú. A Bootstrap célja, hogy segítsen az egyenletes és gyorsan fejlődő webes projektek létrehozásában, minimalizálva a tervezési időt és növelve a projekt hatékonyságát. A keretrendszer egy nagyon népszerű eszköz a webfejlesztők körében, mivel lehetővé teszi az egyszerű és gyorsan fejlődő webes projektek létrehozását, miközben biztosítja a reszponzív design és a konzisztens megjelenés előnyeit.

2.2.7. PHP

A PHP egy általános szerveroldali szkriptnyelv, amelyet dinamikus weblapok készítésére fejlesztették ki. A hagyományos HTML lapokkal ellentétben a kiszolgáló a PHP-kódot nem küldi el az ügyfélnek, hanem a kiszolgáló oldalán a PHP-értelmező motor dolgozza fel azt. A programokban lévő HTML elemek érintetlenül maradnak, de a PHP kódok lefutnak. A kódok végezhetnek adatbázis-lekérdezéseket, létrehozhatnak képeket, fájlokat olvashatnak és írhatnak, kapcsolatot létesíthetnek távoli kiszolgálókkal. A PHP-kódok kimenete a megadott HTML elemekkel együtt kerül az ügyfélhez.

2.2.8. HTTP, REST API

A REST (Representational State Transfer) egy elterjedt architektúra az alkalmazás-programozási interfészek (API-k) tervezéséhez és megvalósításához. A REST API-k az erőforrásokat (adatokat, szolgáltatásokat) ábrázolják, és az HTTP protokollt használják a kliens és a szerver közötti kommunikációhoz. A REST API-k állapotmentesek, azaz a kliens kérésének tartalmaznia kell az összes szükséges információt. A REST API-k népszerűségét az egyszerűség, az egységes interfész és a skálázhatóság adja, amely lehetővé teszi a nagyobb terhelések kezelését.

2.2.9. MySQL

A MySQL egy ingyenes, nyílt forráskódú relációs adatbázis-kezelő rendszer. A MySQL könnyen használható, széles körben elterjedt és skálázható, ami azt jelenti, hogy nagy adatmennyiségeket is képes hatékonyan kezelni. Az adatok egyszerű kezelése és a megbízhatósága miatt a MySQL-t széles körben használják az üzleti alkalmazásokban, weboldalakban, mobilalkalmazásokban, játékokban és még sok más területen.

2.2.10. C#

A C# (C-sharp) egy modern, objektumorientált programozási nyelv. Elsősorban a .NET platformhoz kapcsolódik, és az egyik leggyakrabban használt nyelv a Windows alkalmazások, webalkalmazások és szolgáltatások fejlesztéséhez. A C#-t széles körben használják üzleti alkalmazások, játékok, webalkalmazások, adatszolgáltatások és még sok más fejlesztésére.

2.2.11. Git, GitHub

A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A GitHub ingyenes internetes szolgáltatás, amely a Git segítségével szoftverfejlesztési verziókövetésszolgáltatást nyújt. A grafikus felület segítségével egyszerűen hozhatunk létre és kezelhetünk Git repository-kat. A projektünk menedzsmentjének nagy része a GitHub beépített Projects felületével lett végezve.

2.2.12. ChatGPT

A ChatGPT (Generative Pre-trained Transformer) az OpenAI mesterséges intelligencia (MI) kutató laboratórium által kifejlesztett chatbot, mely a felhasználókkal való folyamatos kommunikáció automatizálása során értelmezőmodelleket használ, melyek segítségével a bevitt információkat azonnal interaktívan kezeli. Segítségével a sokszor hosszás keresést, interneten kutakodást lerövidíthetjük, egy-egy problémára másodperceken belül választ kaphatunk. Emellett kisebb algoritmusokat, mint tesztadatok generálása, vagy bizonyos adatok, esetleg kódrészletek helyességének ellenőrzése, is könnyebb, és még inkább gyorsabb elvégezni a ChatGPT segítségével.

2.3. Adatszerkezet

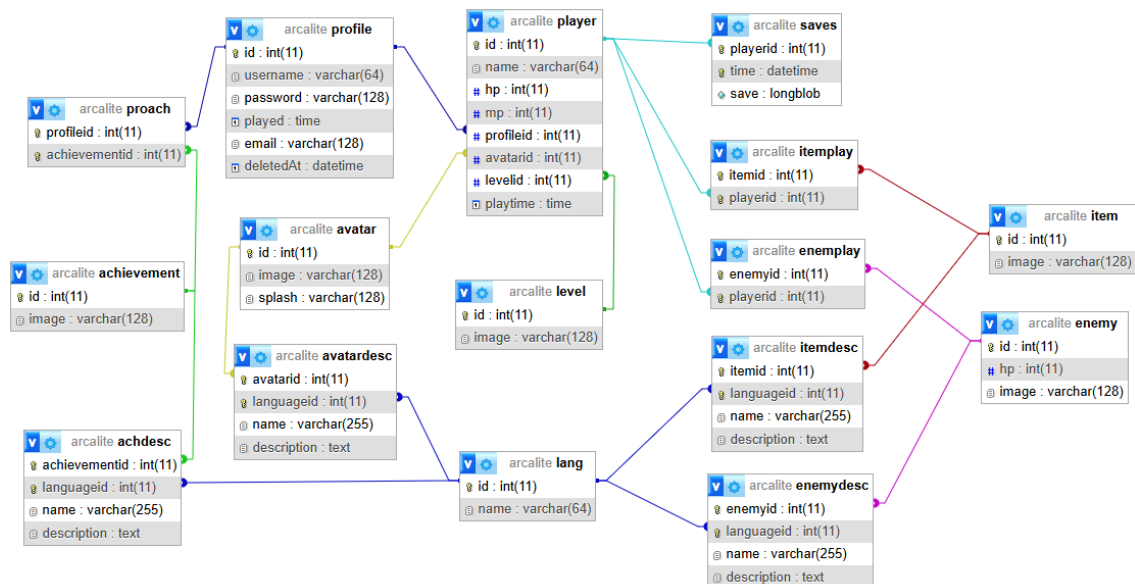
Adatbázisunk célja a felhasználók adatainak, játékbeli állásuknak, illetve a játék általános információinak tárolása

2.3.1. Alapadatok

MySQL kliens verzió	10.4.28 - MariaDB
Adatbázis neve	arcalite
Tárolómotor	InnoDB
Alapértelmezett illesztés	UTF-8

```
CREATE DATABASE arcalite CHARACTER SET utf8;
```

2.3.2. Kapcsolati ábra



2.3.3. Táblák

2.3.3.1. A profile tábla

A felhasználók regisztrációs adatait tárolja, illetve az összes játszott időt.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
username	Szöveg		A fiók felhasználóneve
password	Szöveg		A fiók jelszava (PASSWORD függvénnnyel hash-elve)
email	Szöveg		A fiók e-mail címe
played	Idő		Az összes játszott idő
deletedAt	Dátum és idő		A fiók törlésének időpontja

```
CREATE TABLE profile (
  id INT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(64),
  password VARCHAR(128),
  played TIME DEFAULT 0,
  email VARCHAR(128),
  deletedAt DATETIME DEFAULT NULL
);
```

2.3.3.2. Az achievement tábla

Az elérhető mérföldköveket tartja számon. Egyelőre nincs sokféle adat benne, későbbi fejlesztések miatt lett létrehozva.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
image	Szöveg		A mérföldkőhöz tartozó ikonkép helyét tárolja a szerveren.

```
CREATE TABLE achievement (
  id INT PRIMARY KEY AUTO_INCREMENT,
  image VARCHAR(128)
);
```

2.3.3.3. A *proach* tábla

Az adott felhasználó által elért mérföldköveket tartja számon.

Mezőnév	Típus	Kulcs	Leírás
profileid	Egész	PK, FK	A fiók kódja. Kapcsolat a <i>profile</i> táblával.
achievementid	Egész	PK, FK	A mérföldkő kódja. Kapcsolat az <i>achievement</i> táblával.

```
CREATE TABLE proach (
  profileid INT,
  achievementid INT,
  CONSTRAINT pk_proach PRIMARY KEY (profileid, achievementid),
  CONSTRAINT fk_proach_profile FOREIGN KEY (profileid) REFERENCES
profile(id),
  CONSTRAINT fk_proach_achievement FOREIGN KEY (achievementid)
REFERENCES achievement(id)
);
```

2.3.3.4. A *player* tábla

A felhasználó által készített karakterek adatait tárolja.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
name	Szöveg		A karakter neve
hp	Egész		A karakter életpontjainak maximális értéke
mp	Egész		A karakter manapontjainak maximális értéke
profileid	Egész	FK	A hozzátartozó fiók kódja. Kapcsolat a <i>profile</i> táblával.
avatarid	Egész	FK	A karakter avatárképének kódja. Kapcsolat az <i>avatar</i> táblával.
levelid	Egész	FK	A legutoljára elért szint kódja. Kapcsolat a <i>level</i> táblával.
playtime	Idő		A karakterrel játszott idő.

(a létrehozó kódrészlet a következő oldalon)

```
CREATE TABLE player (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(64),
  hp INT,
  mp INT,
  profileid INT,
  avatarid INT,
  levelid INT,
  playtime TIME DEFAULT 0,
  CONSTRAINT fk_player_avatar FOREIGN KEY (avatarid) REFERENCES
avatar(id),
  CONSTRAINT fk_player_level FOREIGN KEY (levelid) REFERENCES
level(id),
  CONSTRAINT fk_player_profile FOREIGN KEY (profileid) REFERENCES
profile(id)
);
```

2.3.3.5. Az avatar tábla

A karaktereknek választható avatar képek elérési útját tárolja.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
splash	Szöveg		Az avatarhoz tartozó splash art elérési útja a szerveren.
image	Szöveg		Az avatarhoz tartozó kép elérési útja a szerveren.

```
CREATE TABLE avatar (
  id INT PRIMARY KEY AUTO_INCREMENT,
  splash VARCHAR(128),
  image VARCHAR(128)
);
```

2.3.3.6. A level tábla

A szintekhez tartozó képek elérési útját tárolja.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
image	Szöveg		A szinthez tartozó kép elérési útja a szerveren.

```
CREATE TABLE level (
  id INT PRIMARY KEY AUTO_INCREMENT,
  image VARCHAR(128)
);
```

2.3.3.7. Az *enemy* tábla

A játékban lévő ellenfelek adatait tárolja.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
hp	Egész		Az ellenfél életpontjainak maximális értéke.
image	Szöveg		Az ellenfélhez tartozó kép elérési útja a szerveren.

```
CREATE TABLE enemy (
  id INT PRIMARY KEY AUTO_INCREMENT,
  hp INT,
  image VARCHAR(128)
);
```

2.3.3.8. Az *enemplay* tábla

A karakter által már felfedezett ellenfeleket tárolja. Az adott karakter „csak ezeket ismeri”.

Mezőnév	Típus	Kulcs	Leírás
playerid	Egész	PK, FK	A karakter kódja. Kapcsolat a <i>player</i> táblával.
enemyid	Egész	PK, FK	Az ellenfél kódja. Kapcsolat az <i>enemy</i> táblával.

```
CREATE TABLE enemplay (
  enemyid INT,
  playerid INT,
  CONSTRAINT pk_enemplay PRIMARY KEY (enemyid, playerid),
  CONSTRAINT fk_enemplay_enemy FOREIGN KEY (enemyid) REFERENCES
enemy(id),
  CONSTRAINT fk_enemplay_player FOREIGN KEY (playerid) REFERENCES
player(id)
);
```


2.3.3.9. Az *item* tábla

A játékban megszerezhető tárgyak adatait tárolja

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
image	Szöveg		A tárgyhoz tartozó kép elérési útja a szerveren.

```
CREATE TABLE item (
  id INT PRIMARY KEY AUTO_INCREMENT,
  image VARCHAR(128)
);
```

2.3.3.10. Az *itemplay* tábla

A karakter által már megszerzett/felfedezett tárgyakat tárolja. Az adott karakter „csak ezeket ismeri”.

Mezőnév	Típus	Kulcs	Leírás
playerid	Egész	PK, FK	A karakter kódja. Kapcsolat a <i>player</i> táblával.
itemid	Egész	PK, FK	A tárgy kódja. Kapcsolat az <i>item</i> táblával.

```
CREATE TABLE itemplay (
  itemid INT,
  playerid INT,
  CONSTRAINT pk_itemplay PRIMARY KEY (itemid, playerid),
  CONSTRAINT fk_itemplay_item FOREIGN KEY (itemid) REFERENCES
item(id),
  CONSTRAINT fk_itemplay_player FOREIGN KEY (playerid) REFERENCES
player(id)
);
```

2.3.3.11. A lang tábla

Az elérhető nyelveket tárolja.

Mezőnév	Típus	Kulcs	Leírás
id	Egész	PK	
name	Egész		A nyelv megnevezése

```
CREATE TABLE lang (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(64)
);
```

2.3.3.12. Az achdesc tábla

A mérföldkövek nevét és leírását tárolja adott nyelveken.

Mezőnév	Típus	Kulcs	Leírás
achievementid	Egész	PK, FK	A mérföldkő kódja. Kapcsolat az <i>achievement</i> táblával.
languageid	Egész	PK, FK	A nyelv kódja. Kapcsolat a <i>lang</i> táblával.
name	Szöveg		A mérföldkő megnevezése az adott nyelven.
description	Szöveg		A mérföldkő leírása az adott nyelven.

```
CREATE TABLE achdesc (
  achievementid INT,
  languageid INT,
  name VARCHAR(255),
  description TEXT,
  CONSTRAINT pk_achdesc PRIMARY KEY (achievementid, languageid),
  CONSTRAINT fk_achdesc_achievement FOREIGN KEY (achievementid)
REFERENCES achievement(id),
  CONSTRAINT fk_achdesc_languages FOREIGN KEY (languageid) REFERENCES
lang(id)
);
```

2.3.3.13. Az *avatardesc* tábla

Az avatárak nevét és leírását/háttértörténetét tárolja adott nyelveken.

Mezőnév	Típus	Kulcs	Leírás
avatarid	Egész	PK, FK	Az avatár kódja. Kapcsolat az <i>avatar</i> táblával.
languageid	Egész	PK, FK	A nyelv kódja. Kapcsolat a <i>lang</i> táblával.
name	Szöveg		Az avatár megnevezése az adott nyelven.
description	Szöveg		Az avatár leírása az adott nyelven.

```
CREATE TABLE avatardesc (
  avatarid INT,
  languageid INT,
  name VARCHAR(255),
  description TEXT,
  CONSTRAINT pk_avatardesc PRIMARY KEY (avatarid, languageid),
  CONSTRAINT fk_avatardesc_avatar FOREIGN KEY (avatarid) REFERENCES
avatar(id),
  CONSTRAINT fk_avatardesc_languages FOREIGN KEY (languageid)
REFERENCES lang(id)
);
```

2.3.3.14. Az *enemydesc* tábla

Az ellenfelek nevét és leírását tárolja adott nyelveken.

Mezőnév	Típus	Kulcs	Leírás
enemyid	Egész	PK, FK	Az ellenfél kódja. Kapcsolat az <i>enemy</i> táblával.
languageid	Egész	PK, FK	A nyelv kódja. Kapcsolat a <i>lang</i> táblával.
name	Szöveg		Az ellenfél megnevezése az adott nyelven.
description	Szöveg		Az ellenfél leírása az adott nyelven.

```
CREATE TABLE enemydesc (
  enemyid INT,
  languageid INT,
  name VARCHAR(255),
  description TEXT,
  CONSTRAINT pk_enemydesc PRIMARY KEY (enemyid, languageid),
  CONSTRAINT fk_enemydesc_item FOREIGN KEY (enemyid) REFERENCES
enemy(id),
  CONSTRAINT fk_enemydesc_languages FOREIGN KEY (languageid)
REFERENCES lang(id)
);
```

2.3.3.15. Az *itemdesc* tábla

A tárgyak nevét és leírását tárolja adott nyelveken.

Mezőnév	Típus	Kulcs	Leírás
itemid	Egész	PK, FK	A tárgy kódja. Kapcsolat az <i>item</i> táblával.
languageid	Egész	PK, FK	A nyelv kódja. Kapcsolat a <i>lang</i> táblával.
name	Szöveg		A tárgy megnevezése az adott nyelven.
description	Szöveg		A tárgy leírása az adott nyelven.

```
CREATE TABLE itemdesc (
  itemid INT,
  languageid INT,
  name VARCHAR(255),
  description TEXT,
  CONSTRAINT pk_itemdesc PRIMARY KEY (itemid, languageid),
  CONSTRAINT fk_itemdesc_item FOREIGN KEY (itemid) REFERENCES
item(id),
  CONSTRAINT fk_itemdesc_languages FOREIGN KEY (languageid) REFERENCES
lang(id)
);
```

2.3.3.16. A *saves* tábla

Az adott karakter mentési fájljait tárolja el.

Mezőnév	Típus	Kulcs	Leírás
playerid	Egész	PK, FK	A karakter kódja. Kapcsolat az <i>player</i> táblával.
time	Dátum és idő		A mentés időpontja.
save	Longblob		A mentési állomány.

```
CREATE TABLE saves (
  playerid INT,
  time DATETIME DEFAULT CURRENT_TIMESTAMP,
  save LONGBLOB,
  CONSTRAINT pk_saves PRIMARY KEY (playerid, time),
  CONSTRAINT fk_saves_player FOREIGN KEY (playerid) REFERENCES
player(id)
);
```

2.4. A weboldal felépítése

2.4.1. A weboldal szerkezete és formázás

A weboldal 5 darab HTML-fájlból áll, viszont a formázás egy központi CSS fájlban, a *style.css*-ben található, illetve a formázáshoz kapcsolódó JavaScript függvények a *script* mappában a *styleScript.js*-be kerültek.

A weboldalon használt színek többségét CSS-változókkal adjuk meg, és ezek segítségével alakítottunk ki egy sötét és világos témát az oldalnak. Ezt az oldal tetején a menüsávon található kapcsolóval állíthatjuk át. A kapcsoló a háttérben meghívja a *themeChange* függvényt, mely egy cookie-ban eltárolja a jelenleg kiválasztott témát, majd a *setTheme* függvény meghívásával beállítja azt. Az oldal *body* eleme kap egy *bsTheme* adattagot, mely értéke a jelenlegi téma, és amely alapján állítja be a CSS a változóit.

2.4.2. Saját felugróablakok

A weboldalon hibaüzenetek megjelenítésére, illetve a profilműveletekhez szükséges kommunikációra saját készítésű felugróablakokat használunk. Maga a felugróablak egy *div*, benne egy címmel, paragrafussal, szöveges bemeneti mezővel és három gombbal. A közös *common.js* szkriptfájlban találhatjuk a felugróablak működéséhez és használatához szükséges függvényeket:

- **showPopup:** Az oldal tetejére csúsztatja a felugróablakot.
- **closePopup:** Az oldal teteje fölé csúsztatja („elrejt”) a felugróablakot.
- **setupPopup:** A felugróablak mindig megjelenő részeinek beállítását végzi el. Ezek a címszöveg, az üzenet, a „*Mégsem*” gomb beállítása, illetve a többi gomb funkciójának reset-elése.
- **setAlert:** Egy bemenet nélküli felugróablak beállítását végzi el. Paramétereiből beállítja a „*Rendben*” gomb tényleges szövegét, funkcióját, illetve elrejt a nemhasznált részeket.
- **setPrompt:** Egy bemenetet váró felugróablak beállítását végzi el. Paramétereiből beállítja az „*Elküld*” gomb tényleges szövegét, funkcióját, illetve megjeleníti a szükséges elemeket, és elrejt a nemhasználtakat. A paraméterek között szerepel a bemenet helyességét ellenőrző függvény, illetve a helytelen bemenetre visszaadott hibaüzenet is.
- **Alert:** A bemenet nélküli felugróablak használatát lehetővé tevő függvény. Meghívja a megjelenítést, beállítást és az esetleges előző ablak bezárását is.

- **Prompt:** A bemenetet váró felugróablak használatát lehetővé tevő függvény. Meghívja a megjelenítést, beállítást és az esetleges előző ablak bezárását is.

2.4.3. Az API és a weboldal közti kommunikáció

Az API és a kliens közti kommunikációt JavaScript-tel kezeljük, a jQuery *\$.ajax()* függvényének segítségével.

Több végpontnak is szüksége van információkra, mint a felhasználó azonosítója, neve, vagy a megjelenítés nyelvének azonosítója (jelen állapotban mindig 1, azaz magyar). Ezeket az oldal bejelentkezéstől kezdve cookie-kban (sütikben) tárolja, és elérésükhöz a *common.js* állományban áll rendelkezésünkre a *getCookie* függvény, mely a megadott nevű cookie értékét, illetve, ha nincs ilyen nevű cookie, *null* értéket ad vissza.

A tényleges API kérések függvényei a weboldalakhoz tartozó szkriptekben találhatóak.

Fájl	Feladat
index.js	A főoldalon megjelenő karakterek adatait kéri le és tölti be.
login.js	A bejelentkezéshez, regisztrációhoz és új jelszó kéréshez szükséges bemenet-ellenőrzéseket és API-hívásokat intézi.
profile.js	A profil oldalon megjelenő adatokat kéri le és tölti be.
ranking.js	A rangsor táblákba kerülő adatokat kéri le és tölti be.
wiki.js	A lexikon adatait kéri le és tölti be.

2.4.4. Egyéb funkcionalitás a weboldalon

A menüsávon a bejelentkezésfüggő elemek beállítását elvégző függvények a *common.js* fájlban találhatóak. Itt ellenőrzi az oldal, hogy van-e *userid* eltárolva: ha nincs, a *Lexikon* menüpont nem lesz kattintható, illetve a *Bejelentkezés* menüpont fog megjelenni; ha van *userid*, a *Lexikon* elérhető lesz, illetve a *Bejelentkezés* helyett a *Profil* oldalra tudunk eljutni a menüből.

Kijelentkezéskor a már sokszor említett cookie-kat egyszerűen töröljük (felhasználó azonosító és -név), erre a *common.js* fájlban található a *logout* függvény.

2.5. Az API felépítése

2.5.1. A cél

A weboldal, illetve a játék és az adatbázis közötti kommunikációt hivatott az API ellátni. Az API PHP nyelven íródott, a visszatérési adatokat JSON formátumban adja vissza.

2.5.2. Hibakezelés és közös metódusok

Mivel minden végpont meghívásakor az első lépés a bemenő adatok ellenőrzése, az adatbázis kapcsolat felállítása, illetve hibák esetén hasznos visszajelzés küldése, ezért ezek két „közösén használt” PHP-fájlba kerültek.

A *config.php* a bemenő adatok validálására (*hasProperFields* és *checkValidity*) és a visszatérési értékek visszaadására (*ReturnMessage*, *ReturnResult*, *ReturnQuery*) való függvényeket tartalmazza. Utóbbiak mind valamiféle adatot (akár egy üzenetet, akár adatbázisból lekért adathalmazt), mind egy HTTP-válaszkódot is visszaadnak, így hibaüzenetek küldésére is ezeket használják a végpontok.

A *connection.php* az adatbázis kapcsolatot hozza létre a *mysqli* osztály példányosításával. Ezzel központilag, egy helyen lehet megadni a kapcsolathoz szükséges információkat (*\$CONFIG* tömb), mint az adatbázis neve, a szerver címe stb. Kapcsolódási hiba esetén hibát dob vissza, melyben megadja az SQL hibakódot és -üzenetet is.

2.5.2.1. A *config.php* által küldhető hibakódok

Válaszkód	Üzenet	Megjegyzés
500	Hiba az SQL lekérdezésben: (Hibakód: ...) ...	
405	Hiba az API-hívásban.	Hibás metódus esetén
400	Hiba az API-hívásban.	Hibás bemenet esetén

Ezeket a hibakódokat a többi végpont is adhatja, ott nincsenek ezek külön felsorolva.

2.5.3. A végpontok

2.5.3.1. *login.php*

Feladat: Bejelentkeztetés

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
email	szöveg	A bejelentkezéshez használt e-mail cím.
password	szöveg	A fiókhoz tartozó jelszó.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
id	egész	A fiók egyedi azonosítója.
username	szöveg	A fiókhoz tartozó felhasználónév.
email	szöveg	A fiókhoz tartozó e-mail cím.

Hibakódok:

<i>Válaszkód</i>	<i>Üzenet</i>
401	Nincs ehhez az e-mail címhez regisztrált fiók!
401	Hibás jelszó!
401	Több fiók bejelentkezési adatai egyeznek!

2.5.3.2. *register.php*

Feladat: Az újonnan regisztráló felhasználó adatainak rögzítése.

Metódus: **POST**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
username	szöveg	A fiókhoz tartozó felhasználónév
email	szöveg	A fiókhoz tartozó e-mail cím.
password	szöveg	A fiókhoz tartozó jelszó.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
code	egész	HTTP válaszkód
message	szöveg	A válaszüzenet

Hibakódok:

<i>Válaszkód</i>	<i>Üzenet</i>
409	Már van fiók regisztrálva ehhez az e-mail címhez!
409	Már van fiók ilyen felhasználónévvel!
500	Hiba az adatok feltöltése közben.

2.5.3.3. *email_exists.php*

Feladat: Ellenőrzi, hogy egy adott e-mail címhez tartozik-e aktív felhasználó.

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
email	szöveg	A fiókhoz tartozó e-mail cím.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
exists	logikai	Létezik-e a megadott e-mail címhez tartozó fiók.
username	szöveg	A fiókhoz tartozó felhasználónév. Ha az <i>exists</i> hamis, NULL.
email	szöveg	A fiókhoz tartozó e-mail cím. Ha az <i>exists</i> hamis, NULL.

Hibakódok:

A *config.php* által adhatók.

2.5.3.4. *new_password.php*

Feladat: Új jelszó beállítása.

Metódus: **POST**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
email	szöveg	A fiókhoz tartozó e-mail cím.
password	szöveg	A beállítandó jelszó.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
code	egész	HTTP válaszkód
message	szöveg	A válaszüzenet

Hibakódok:

<i>Válaszkód</i>	<i>Üzenet</i>
500	Hiba a frissítés közben! Az új jelszó nem került beállításra!

2.5.3.5. *profile.php*

Feladat: A bejelentkezett felhasználó karakteradatainak lekérése.

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
userid	egész	A lekérni kívánt fiók egyedi azonosítója.
langid	egész	A lekért adatok nyelve.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
-	tömb[karakteradat]	Egy tömb a lekért karakteradatokkal.

A karakteradatok felépítése:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
id	egész	A karakter egyedi azonosítója.
name	szöveg	A karakter neve.
hp	egész	A karakter maximum életpontja.
mp	egész	A karakter maximum manapontja.
level	egész	A karakter legmagasabb elért szintje.
playtime	idő	A karakterrel játszott idő.
image	szöveg	A karakterhez tartozó splashart.

Hibakódok:

A *config.php* által adhatók.

2.5.3.6. *update.php*

Feladat: E-mail cím vagy felhasználónév változtatás.

Metódus: **POST**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
userid	egész	A fiókhoz tartozó egyedi azonosító
username	szöveg	Az új felhasználónév. Ha üres szöveg, ignorálja a végpont.
email	szöveg	Az új e-mail cím. Ha üres szöveg, ignorálja a végpont.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
username	szöveg	A frissített rekordhoz tartozó felhasználónév.
email	szöveg	A frissített rekordhoz tartozó e-mail cím.

Hibakódok:

A *config.php* által adhatók.

2.5.3.7. delete_profile.php

Feladat: Fióktörlés

Metódus: **POST**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
userid	egész	A fiókhoz tartozó egyedi azonosító

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
code	egész	HTTP válaszkód
message	szöveg	A válaszüzenet

Hibakódok:

<i>Válaszkód</i>	<i>Üzenet</i>
500	Hiba a törlés során! A profil nem került törlésre!

2.5.3.8. *index.php*

Feladat: A játékban játszható karakterek adatainak lekérése.

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
langid	egész	A lekért adatok nyelve

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
-	tömb[karakteradat]	Egy tömb a lekért karakteradatokkal.

A karakteradatok felépítése:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
id	egész	A karakter egyedi azonosítója.
image	szöveg	A karakterhez tartozó avatárkép elérési útja a szerveren.
name	szöveg	A karakter neve.
description	szöveg	A karakterről szóló leírás.

Hibakódok:

A *config.php* által adhatók.

2.5.3.9. *ranking.php*

Feladat: A ranglistákhoz szükséges adatok lekérése.

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
type	szöveg	A ranglista szempontja.
langid	egész	A lekért adatok nyelve.

Kimenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
-	tömb[adat]	Egy tömb a lekért adatokkal.

Az adatok felépítése, ha a type „Profile”:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
Felhasználónév	szöveg	A fiókhoz tartozó felhasználónév.
Játékidő	idő	A játékos összegzett játékidője.
Elért_mérföldkövek	egész	A játékos által elért mérföldkövek száma.
Végigjátszások	egész	A játékos végigvitt játékainak száma.

Az adatok felépítése, ha a type „GameThrough”:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
Karakter	szöveg	A karakter neve.
Profil	szöveg	A karakterhez tartozó fiók neve.
Avatár	szöveg	A karakterhez tartozó avatár neve.
Játékidő	idő	A karakterrel játszott idő.
Elért_szint	egész	A karakterrel elért legmagasabb szint.
Felfedezett_ellenfelek	egész	A karakterrel felfedezett ellenfelek száma.
Felfedezett_tárgyak	egész	A karakterrel felfedezett tárgyak száma.

Hibakódok:

A *config.php* által adhatók.

2.5.3.10. wiki.php

Feladat: A lexikonhoz szükséges adatok lekérése.

Metódus: **GET**

Bemenet:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
userid	egész	A lekérni kívánt fiók egyedi azonosítója.
langid	egész	A lekért adatok nyelve.
request_type	szöveg	A lekérés típusa.

Kimenet:

Ha a *request_type* „STATISTICS”:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
enemyFound	egész	A játékos által már felfedezett ellenfelek száma.
enemyAll	egész	Az ellenfelek összzdarabszáma.
itemFound	egész	A játékos által már felfedezett tárgyak száma.
itemAll	egész	A tárgyak összzdarabszáma.

Egyébként:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
-	tömb[adat]	Egy tömb a lekért karakteradatokkal.

Az adatok felépítése, ha a *request_type* „ENEMY”:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
id	egész	Az ellenfél egyedi azonosítója.
name	szöveg	Az ellenfél neve.
hp	egész	Az ellenfél maximum életpontja.
desc	idő	Az ellenfél leírása.
image	szöveg	Az ellenfélhez tartozó kép.

Az adatok felépítése, ha a request_type „ITEM”:

<i>Megnevezés</i>	<i>Típus</i>	<i>Leírás</i>
id	egész	A tárgy egyedi azonosítója.
name	szöveg	A tárgy neve.
desc	idő	A tárgy leírása.
image	szöveg	A tárgyhoz tartozó kép.

Hibakódok:

<i>Válaszkód</i>	<i>Üzenet</i>
400	Hiba az API-hívásban.

2.6. Az alkalmazás felépítése

A játék forráskódjának általunk írt fájljai a játék mappáján belül a *Script* mappában találhatóak. A feladat megszabásai és tanulmányaink miatt C# nyelven írtuk a forráskódot. A .cs fájlok a *Script*-en belül feladatuk szerint csoportosítva vannak. A mappák feladatait az alábbi táblázat szemlélteti.

Mappa	Benne lévő fájlok feladata
Game	A játék elemeinek, szereplőinek működését biztosító szkriptek.
Globals	Minden más által elérhető funkciók szkriptjei. Ilyen pl. az adatbázis-kapcsolatot létesítő fájl vagy a mentést és betöltést végző fájl.
Maps	A pályák váltásának kezelését végző szkriptek.
Mechanics	A játékban található mechanikák és logikák kezelését végző szkriptek.
Menus	A különféle menük és egyéb GUI elemek kezelését végző szkriptek.

A szkriptfájlok mellett egy *.cs.uid* fájl is található, mely a GODOT játékmotor miatt kell, mivel az ezalapján ismeri fel és használja a .cs fájlokat.

2.6.1. Játék design, logika

2.6.1.1. Pályatervezés és logika

A pálya alapvetően lineáris kialakítású. A játékos csak vízszintesen tud haladni. Ez a jövőbeli pályákban továbbfejleszthető a függőleges haladás bevezetésével, vagyis a játékos fel- és lefelé is elhagyhat szobákat, mivel a háttérrendszer már készen áll erre.

A pályák kézzel készültek, több TileMapLayer motor elem felhasználásával különböző dekorációs rétegekhez, valamint egy fizikai réteghez, amellyel a játékos kölcsönhatásba léphet. Minden réteg ugyanazt a közös TileSet erőforrást használja a konzisztencia és optimalizáció érdekében.

Egy szoba kétféle mechanizmussal rendelkezhet: egy ellenőrzőponttal (checkpoint), vagy egy ellenség vezérlővel (enemy controller).

Az ellenség vezérlők a szoba abszolút középpontjában helyezkednek el, és az adott szoba összes ellenségéért felelősek. Minden vezérlőhöz egyedi számú Spawner elem van csatolva, amelyek meghatározzák az ellenség típusát és megjelenési helyét. Ezek funkciói a következők:

- Ellenségek megjelenítése, amikor a játékos egy szomszédos szobába lép (ellenségek alaphelyzetben idle mozgásba kerülnek)

- Ellenségek eltüntetése, ha a játékos elhagyja a közelséget. Ez rövid időzítőhöz kötött, és megszakad, ha a játékos gyorsan visszatér. Ez azért szükséges, hogy a játék ne pazarolja az erőforrásokat, ha a játékos gyorsan oda-vissza mozog a szobák között.
- Értesíti az ellenségeket, amikor a játékos belép a szobába, és lezárja a szobát, hogy a játékos ne tudjon távozni, amíg le nem győzi az összes ellenséget.
- A szoba feloldása, amikor érzékeli, hogy az összes hozzá tartozó ellenség elpusztult.
- Egy logikai változó (boolean) tárolja, hogy a szoba ki lett-e tisztítva – ez kerül mentésre a játékmenet file-jában, hogy a játékba visszalépéskor a kitisztított szobákban ne éledjenek újra az ellenségek.

A másik típusú mechanizmus az ellenőrzőpont (checkpoint). Az ellenőrzőpontok egyfajta megpihenési lehetőséget nyújtanak a játékosnak: 20 egységnyi életpontot gyógyítanak, valamint növelik az életpont és manapont regenerációs sebességet az aktiválás után. Az ellenőrzőpont ütközési területét (trigger hitbox) manuálisan lehet szerkeszteni az editorban, így meg lehet növelni a méretét, hogy a játékos véletlenül se hagyja ki. Ez azért fontos, hogy a játékos ne tudja kihasználni az ellenőrzőpont gyógyító és mentési funkcióját.

Az ellenőrzőpont aktiválása elindítja a mentési folyamatot is, amely rögzíti a fontos adatokat és egy szövegfájlba exportálja azokat.

Lehetnek olyan szobák is, amelyek egyik mechanizmussal sem rendelkeznek. Ezek általában valamilyen típusú logikai feladványt vagy más környezeti kihívást, jelenetet tartalmaznak.

2.6.2. Kód struktúra, architektúra

2.6.2.1. Játékmotor

A projekthez a Godot motort választottuk, mivel könnyű, hordozható, és jól alkalmazható 2D-s fejlesztéshez. Továbbá a GDScript helyett a C# nyelv használata mellett döntöttünk, a követelményeknek megfelelően, illetve ismertebb számunkra ez a nyelv. Emellett a C# egy stabilabb nyelv a GDScript-hez képest.

2.6.2.2. Fájl struktúra

A játék három fő mappából áll: Assets, Nodes és Scripts.

Az Assets mappában található minden asset amit használ a játék. Itt találhatóak:

- Sprites: minden entitás képkockái
- Tilesheet: az elemek amikből felépül a pálya, illetve pár kisebb elem, például a kurzor
- UI: felhasználói felület elemei
- Resources: egy tároló minden más .tres és .res fájl számára, amelyek egyébként önállóak
- Shaders

Az utolsó almappa a Placeholder assets, amely az összes letöltött asset-et tartalmazza egy központi könyvtárként.

A Nodes mappa a .tscn fájlok tárolására szolgál, amelyek minden egyedi Node-struktúrát, más néven példányt képviselnek.

A Nodes mappának négy almappája van:

A „Game” mappa tartalmazza az összes entitást, például a játékost, az ellenségeket és az összes lövedéket.

- A „Maps” mappa tartalmazza a pályákat.
- A „Mechanics” mappában található azok a Node-ok, amelyek nem láthatók a játékos számára (backend nodes), például a checkpoint Node-ok és a kameravezérlő.
- A „Menus” mappa pedig az összes „Control” típusú Node-ot tartalmazza, amelyek a játék menürendszerének elemei.

Két további példány található közvetlenül a „Nodes” mappában: a játék fő jelenete, amelybe a játék induláskor betölt, valamint egy „animációtároló” jelenet, amely a letöltött effektek közül a hasznos animációk rendszerezésére szolgált.

A Scripts mappa hasonló almappa-struktúrát követ, mint a Nodes mappa, mivel minden példányhoz tartozik egy script, ezért az áttekinthetőség érdekében lett így tükrözve a felépítés. Az egyetlen különbség a „Globals” almappa, amely az összes singleton/autoload scriptet tartalmazza, amelyeket a játék használ.

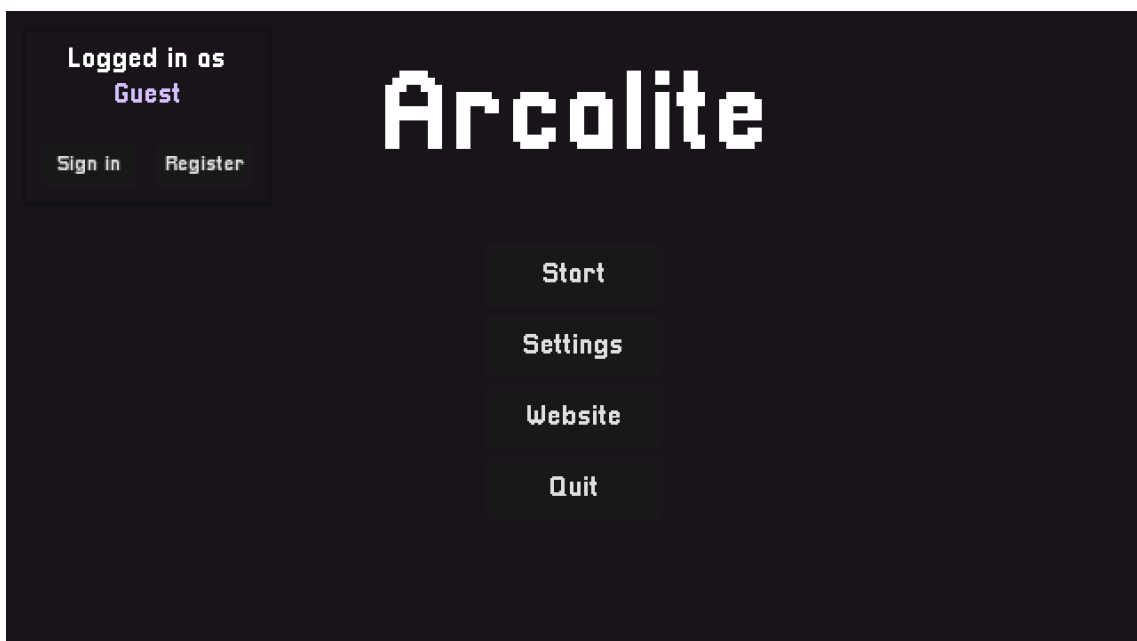
2.6.2.3. Node és Scene hierarchia

A program elindításakor a játék elsőként a töltőképernyőt tölti be. Ezt követően betöltődik az összes singleton, köztük a PreloadRegistry.cs, amely felelős a kódból elérhető összes erőforrás indexeléséért. Miután a PreloadRegistry betöltötte az összes erőforrást, jelet küld vissza a töltőképernyőnek, amely ezután átvált a main.tscn jelenetre. Ez a jelenet a tényleges játék kezdőpontja.

A játék alapvetően két jelenet között váltogat: main.tscn és gameScene.tscn között. A main jelenet tartalmazza a főmenüt, míg a gameScene aktív a tényleges játékmenet közben.

Amikor a main jelenetre váltunk, az létrehozza a mainMenu.tscn példányát, mint gyermekelemet.

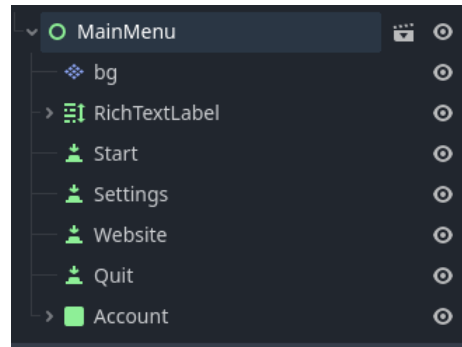
A mainMenu jelenet négy kattintható opciót tartalmaz, valamint egy fiókkezelő panelt a bal felső sarokban. A fiókkezelő panel a következőkből áll:



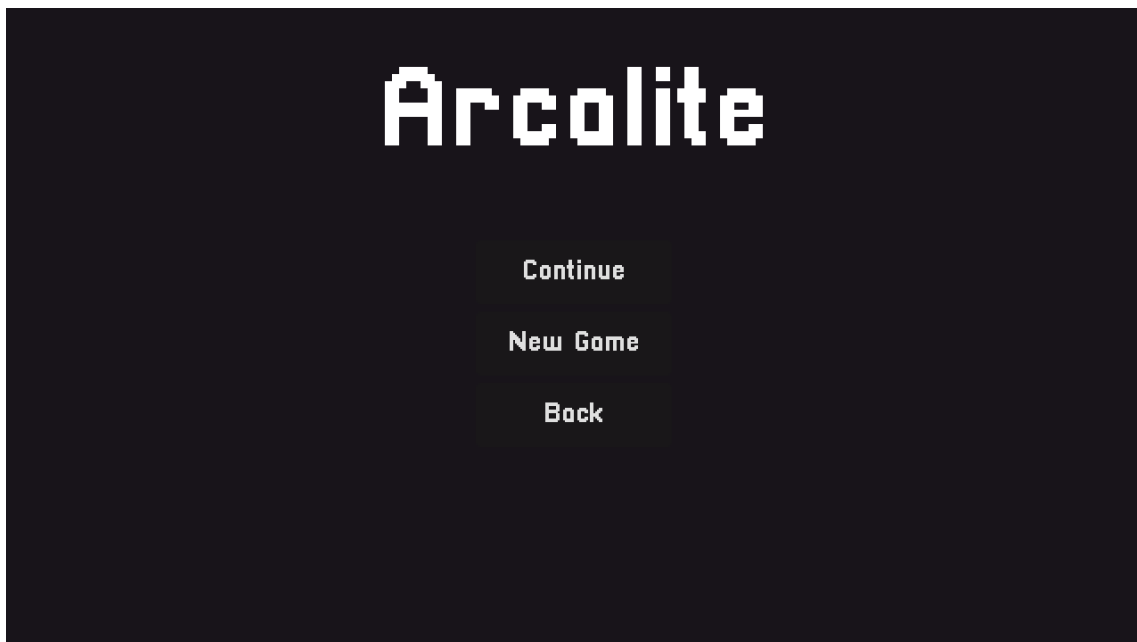
- Egy szövegmező, amely a fiók nevét jeleníti meg, vagy "Vendég" (Guest) feliratot, ha a felhasználó nincs bejelentkezve.
- Egy bejelentkezés/kijelentkezés gomb (a felhasználó aktuális státuszától függően változik).
- Egy gomb, amely a weboldal regisztrációs oldalára irányítja a felhasználót; ez csak kijelentkezett állapotban látható.

A képernyő közepén elhelyezkedő fő gombok a következők:

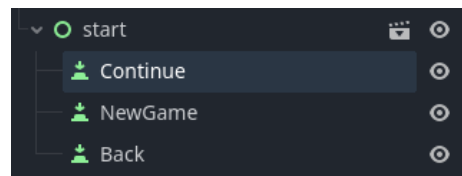
- Start: Három opciót tartalmaz: Folytatás, Új játék, és Vissza.
- Beállítások (Settings): Megnyitja a beállítási menüt.
- Weboldal (Website): Megnyitja a játék weboldalának kezdőlapját a felhasználó alapértelmezett böngészőjében.
- Kilépés (Quit): Kilépteti a játékot.



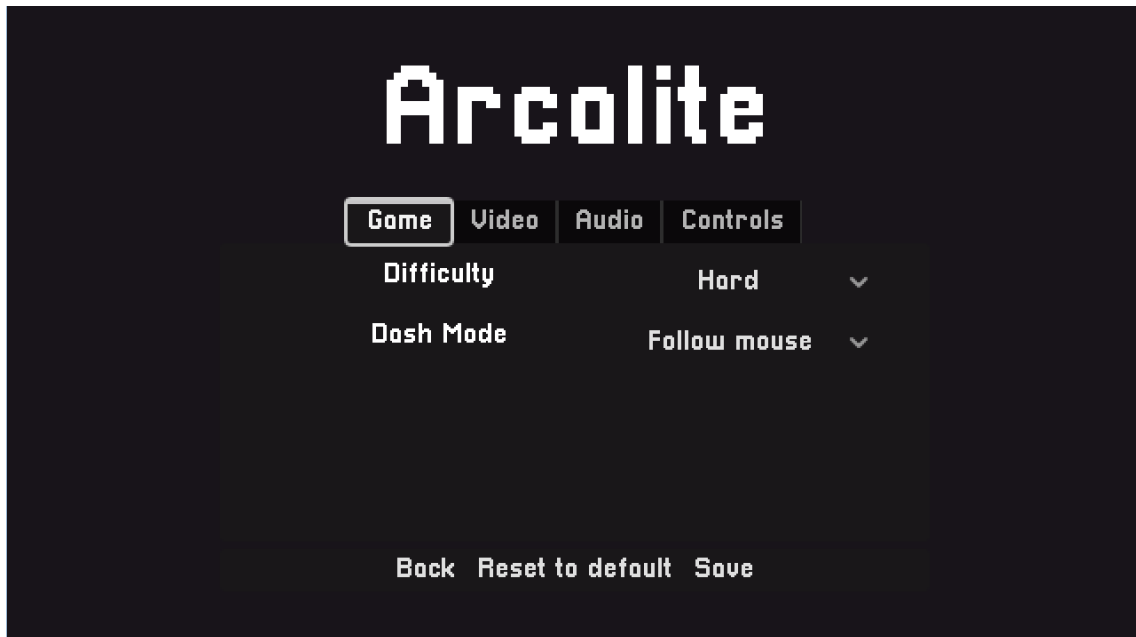
A Start almenüben további három lehetőség található:



- Folytatás: Lehetővé teszi meglévő mentések folytatását
- Új játék: Felszólítja a felhasználót a mentés nevének megadására, majd új játékot indít.
- Vissza: Visszalép az előző menübe.

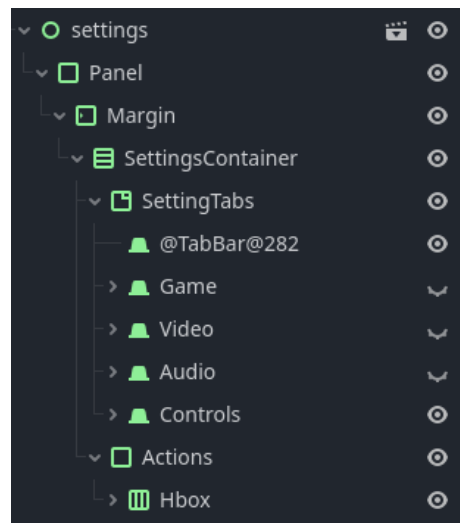


A Beállítások almenü két fő részből áll, amelyeket több rétegnyi formázó konténer Node rendez el. Ez a két rész az opciók területe és a műveleti sáv (actions bar).



Az opciók területe egy TabContainer Node-ot használ, amely lehetővé teszi különböző lapok (tabs) közötti váltást. A játék négy lapot használ:

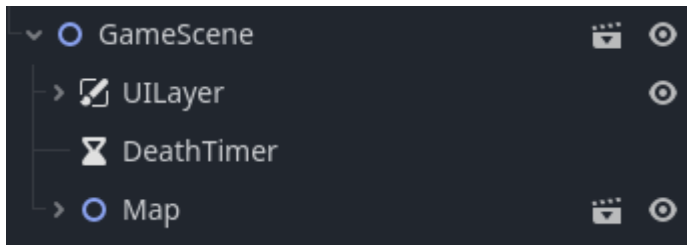
- Játék (Game): A játékmenetre vonatkozó beállításokat tartalmazza, például a nehézségi szintet és a sprint módot.
- Videó (Video): Felbontás, ablakmód, vsync.
- Hang (Audio): Fő hangerő, effektek (SFX) és zene külön-külön. Jelenleg nincsen hangja a játéknak, de van rá opció.
- Irányítás (Controls): Egyedi Node-okat használ, amelyek lehetővé teszik a játékos számára, hogy minden játékon belüli műveletet bármelyik billentyűre vagy egérgombra átállítson. Lehetőség van elsődleges és másodlagos billentyű hozzárendelésére is.



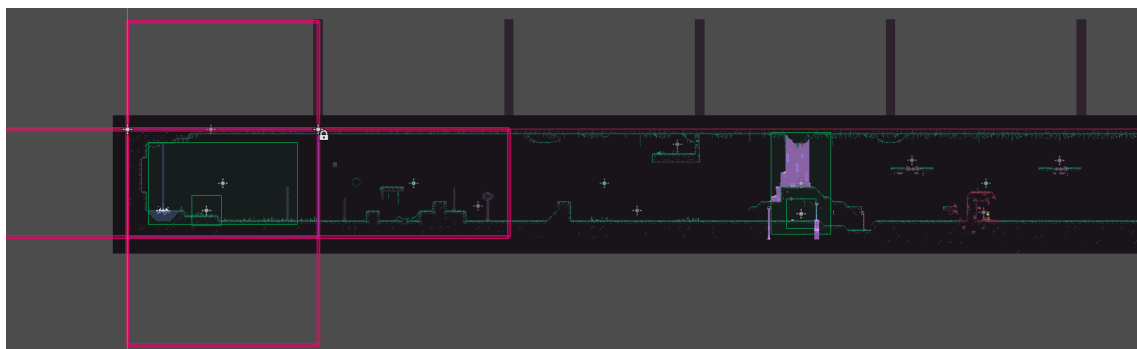
Az Irányítás lap elemei HotKeyRebindItem nevű egyedi Node-példányokból állnak. Ezek vízszintesen rendezett konténerekből állnak, amelyek egy szövegmezőt és két gombot tartalmaznak. Alapértelmezetten a szövegmező és a gomb "action" és "none" felirattal jelenik meg. Inicializáláskor ezek az elemek megkapják az általuk reprezentált művelet nevét, és ennek megfelelően frissítik a helykitöltő szövegeket.

Amikor új játékot indítunk vagy meglévő mentést töltünk be, a játék átvált a `GameScene.tscn` jelenetre.

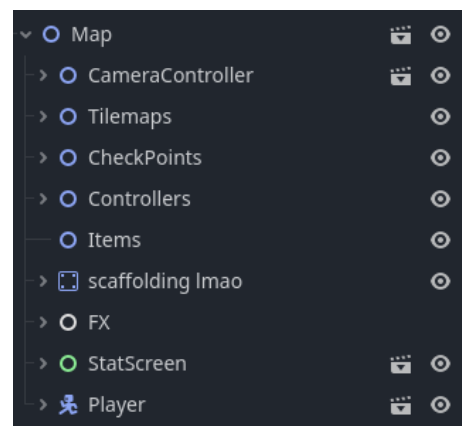
Ez a jelenet példányosítja a megfelelő pályát (map) és a UI réteget. A további játékmene-
tet a Map Node, annak scriptje, illetve a gyerekelemek és azok scriptek irányítják.



A Map Node a következő Node-csoportokat tartalmazza:

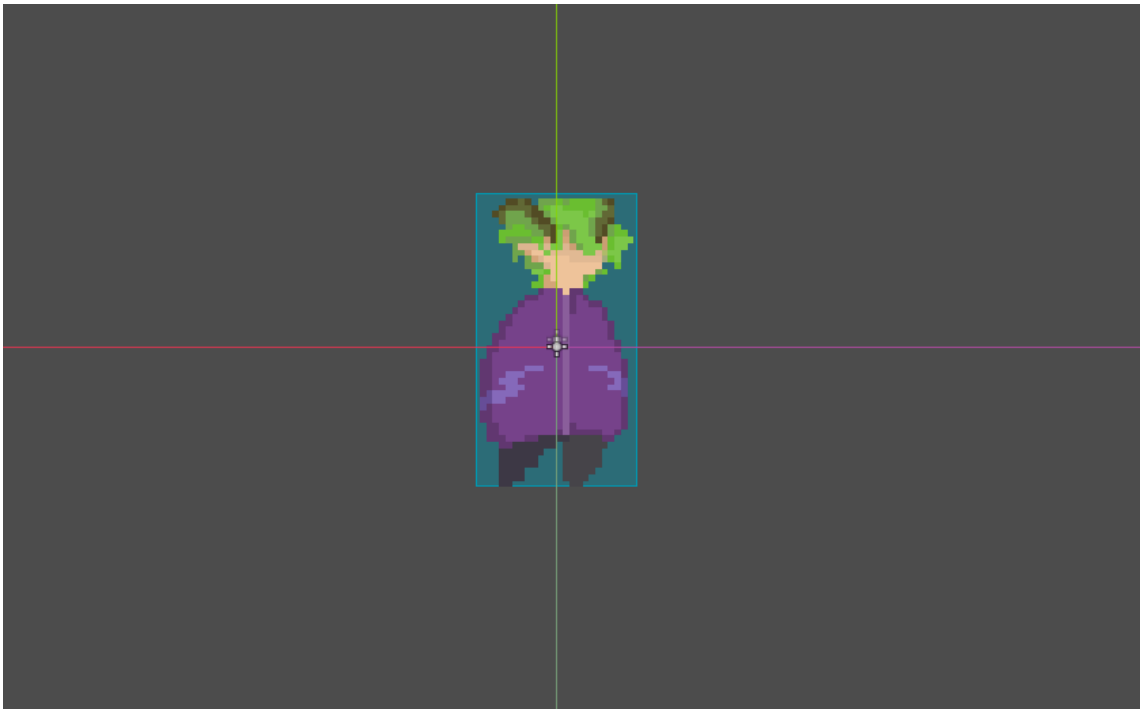


- **CameraController:** Ez az elem tartalmazza a kamerát, valamint egy jelző területet, amely visszajelzést ad arról, hogy a játékos éppen melyik szobában tartózkodik. Tartalmaz továbbá olyan határolókat is, amelyek megakadályozzák, hogy az ellenségek elhagyják a szobát, és harc közben bezárják a játékost.
- **Tilemaps:** Egy szervező Node, amely az összes környezeti réteget (`TileMapLayer`) összefogja.
- **CheckPoints:** Tartalmazza az összes `CheckPoint` Node-ot.
- **Controllers:** Az összes `EnemyControl` Node tárolója.
 - Az `EnemyControl` Node-ok alatt több `EnemySpawn` Node található, amelyek meghatározzák, hogy milyen típusú és hol megjelenő ellenségek jönnek létre.
- **Items:** Tárolja az összes elejtett tárgyat.



- Scaffolding: Nem látható segédstruktúra, amely a szobák méretét jelzi pályakészítés során.
- FX: Tárolja az összes vizuális effektekkel kapcsolatos Node-ot, például esőt.
- StatScreen: Egy "Control" típusú elem, amely a futam végén megjelenő statisztikákért felelős.

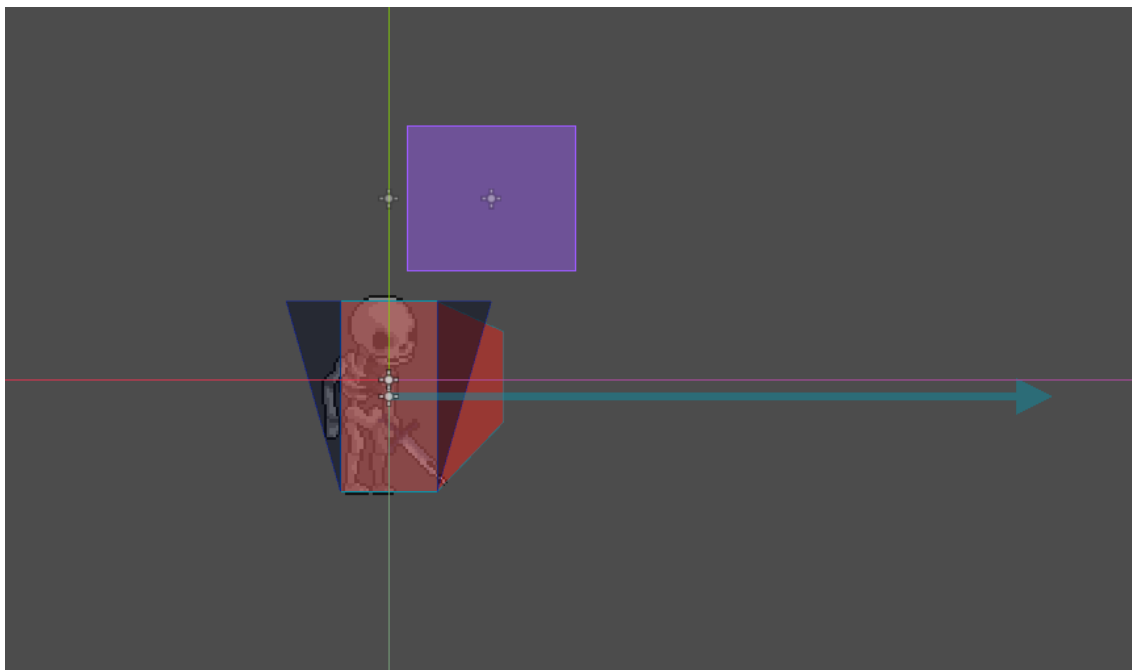
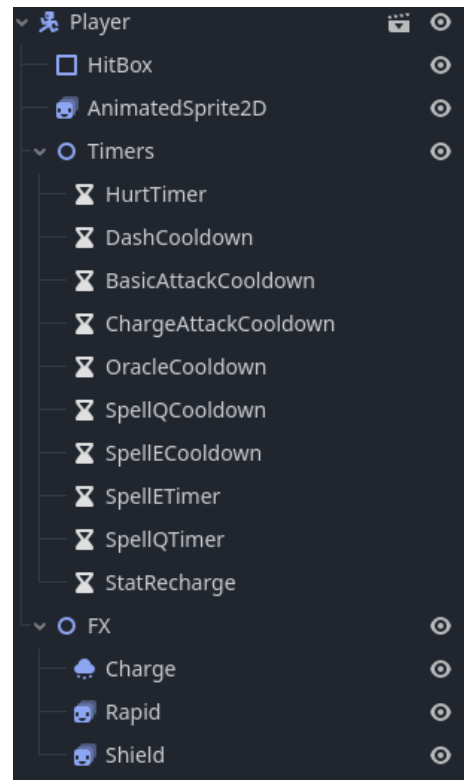
A játékos (Player) dinamikusan jelenik meg az egyik ellenőrzőpontnál, és a Map Node gyermekelemeként jön létre.



A Player Node négy alapelemmel rendelkezik:

- HitBox: CollisionShape2D típus, a fizikai ütközésekért felelős.
- AnimatedSprite2D: A vizuális megjelenítés.
- Timers: Egy szervező Node, amely a játékos scriptje által használt időzítőket tartalmazza.
- FX: Egy szervező Node, amely a játékos által használt vizuális effekteket foglalja magában.

Az ellenségek az EnemyControl Node-ok alatt található EnemySpawn Node-ok által jönnek létre az EnemyControl gyermekelemeiként.



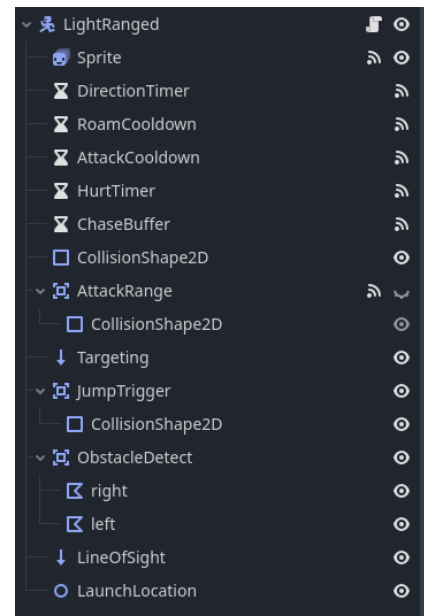
Nincs egy központi "Enemy" Node; az ellenségek scriptfájlja egy alap osztályként működik, amelyből az egyes ellenség típusok öröklík a saját scriptjeiket. Alapvetően egy ellenség az alábbi elemekből áll:

- HitBox
- Sprite
- Timers (nincs külön szervező Node, típustól függően változó mennyiségben)
- AttackRange: Area2D típus, amely meghatározza, hogy a játékos az ellenség támadási tartományában van-e.
- JumpTrigger: Area2D típus, amely ellenőrzi, hogy van-e akadály az ellenség előtt.
- ObstacleDetect: Area2D típus, amely ellenőrzi, hogy elég hely van-e egy sikeres ugráshoz.
- LineOfSight: RayCast2D típus, amely csak a tereppel és a játékosal lép kölcsönhatásba; ez az ellenség "szeme".

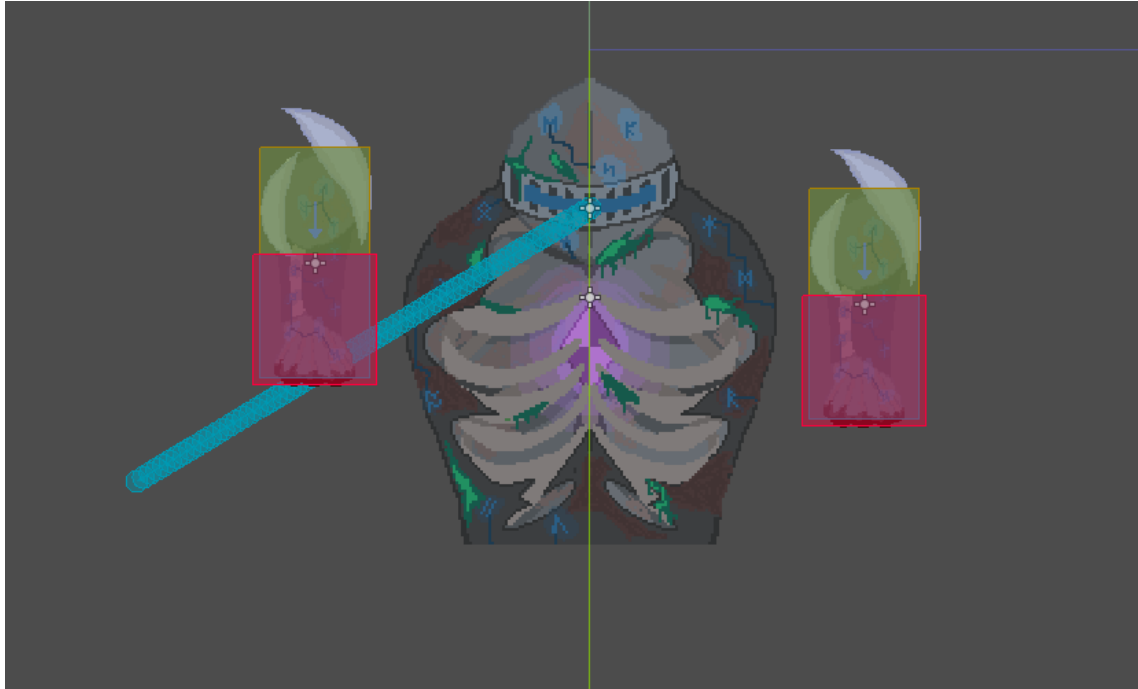


Ezen kívül az ellenség osztályától függően rendelkezhet egy célzó RayCast-tal és egy kilövési pozíciót jelző csomóponttal, ha távolsági támadásokat használ, vagy extra támadási hatótávval, időzítővel és animációs sprite-tal, ha speciális típusú támadásai vannak.

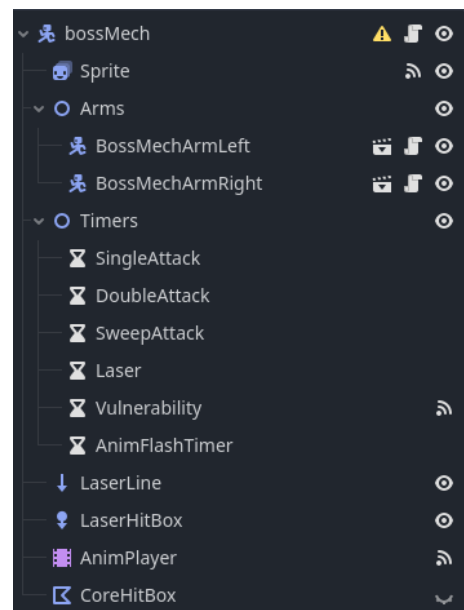
- Sprite (animált)
- CoreHitBox: fizikai ütköződoboz, nem fedi le a főellenfél teljes testét, csak a magját (vizuálisan jelölve), amely az egyetlen sérülékeny pont.
- Arms: szervezőcsomópont, tartalmazza a gólem két karját. Mivel a karok fontos részei a főellenfél kialakításának, külön példányként jelennek meg.
- Timers: szervezőcsomópont, időzítőket tartalmaz.
- LaserLine: RayCast2D, a lézertámadás háttérbeli (logikai) reprezentációja.
- LaserHitBox: ShapeCast2D, a lézertámadás fizikai megjelenítése.
- AnimPlayer: AnimationPlayer csomópont, a főellenfél bevezető és halálanimációjához használva.



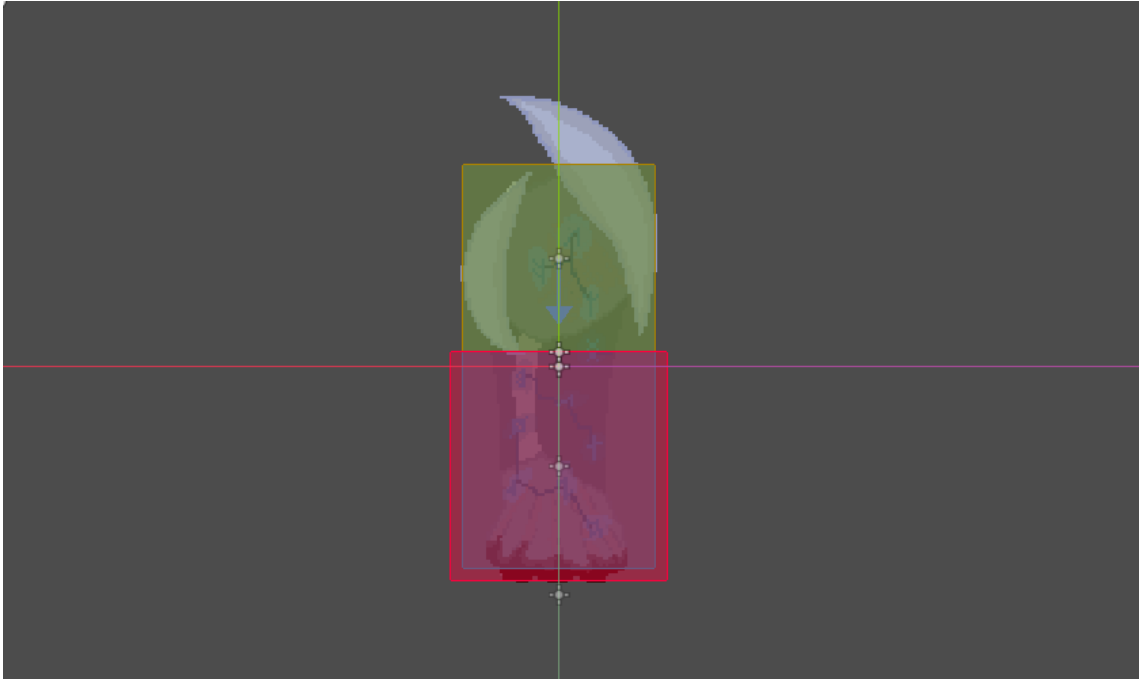
Az utolsó jelentős entitás a végső főellenfél, a mehaboss/gólem/CALYPSOS (még nem tudom, hogyan nevezzem el). A jelentős eltérések miatt ez az ellenség nem használja az alap ellenségosztályt, hanem saját osztálya van. A főellenség a következő elemekből áll:



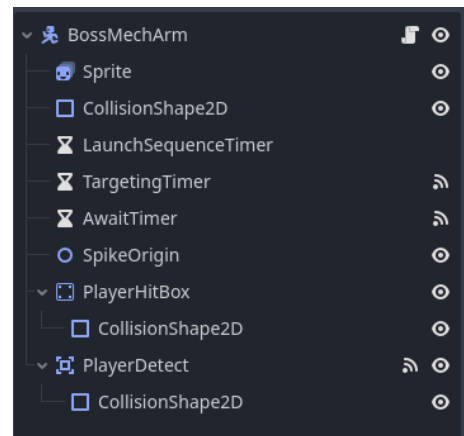
- Sprite (animált)
- CoreHitBox: Fizikai hitbox, csak a főellen-ség magját fedi, csak itt sebezhető.
- Arms: Szervező Node, amely a két kart tartalmazza (külön példányok).
- Timers: Szervező Node időzítőkkal.
- LaserLine: RayCast2D, a lézertámadás hát-térreprezentációja.
- LaserHitBox: ShapeCast2D, a lézertámadás fizikai reprezentációja.
- AnimPlayer: AnimationPlayer Node, az intro és a halál animáció vezérlésére.



A karjai az alábbi elemekből állnak:

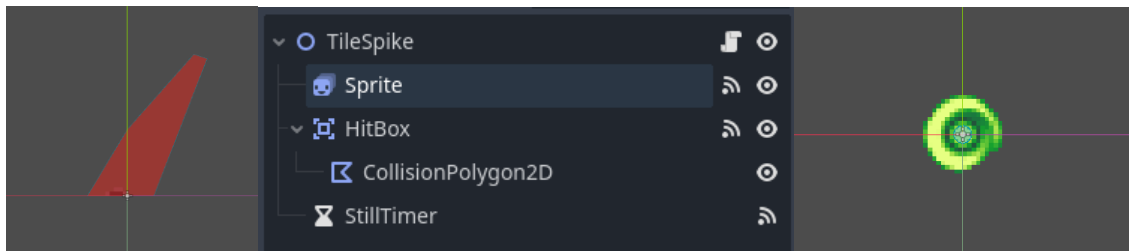


- Sprite
- HitBox (jelenleg nincs használatban, ha jól emlékszem)
- Timers (nincs szervező Node, mivel csak három időzítő van)
- SpikeOrigin: Egy pointer node, amely meghatározza a tüskék megjelenési helyét támadások során, így elkerülhető a pozíciók hardkódolása (a dinamikusság érdekében).
- PlayerHitBox: StaticBody2D típus, szilárd objektumként szolgál, amelyre a játékos felállhat (terepként van osztályozva). Ez egy egyirányú ütközés, tehát a játékos csak az "alulról" nézve felső részén tud állni (a karok tetején).
- PlayerDetect: Area2D típus, amely meghatározza, hogy a játékos támadási távolságon belül van-e, közvetlen támadásokhoz használatos.



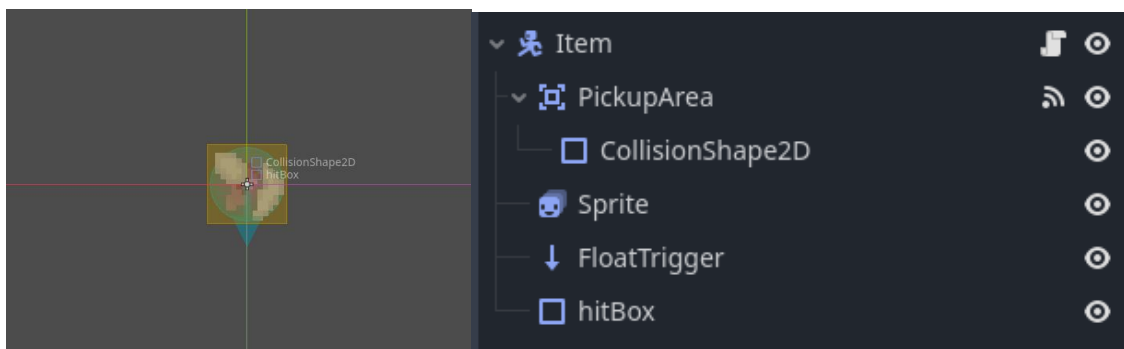
Az összes lövedék a játékban – függetlenül attól, hogy a játékos vagy az ellenség hozza létre – ugyanazokat az alapvető elemeket osztja meg: Sprite és HitBox.

A lövedékek típusa alapján további elemek is lehetnek bennük, például egy időzítő (Timer). Például az Oracle képesség (időlassítás) rendelkezik egy időzítővel, amely meghatározza, hogy mennyi ideig marad a pályán.



Egyedi lövedék a végső főellenség lézere, amely nem rendelkezik saját ütközési területtel (hitbox), mivel ez magán a főellenségen található, és háromféle animált sprite-ot használ.

Az itemek (tárgyak) számára csak egy közös jelenet létezik, és a típusuk példányosításkor kerül meghatározásra.



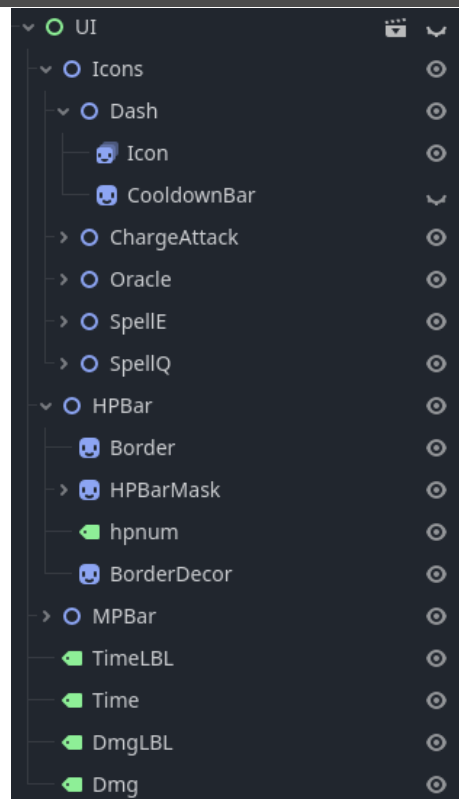
Az item főbb elemei:

- Sprite: Több fül (tab) van az összes tárgy típushoz, a helyes sprite-ot választja ki a példányosításkor megadott típus alapján.
- HitBox: Szilárd ütközési terület, amely megakadályozza, hogy a tárgyak átmenjenek a tereptárgyakon.
- PickupArea: Area2D típus, amely felelős azért, hogy a játékos érzékelje a tárgyat, és hogy az érintkezéskor eltűnjön.
- FloatTrigger: RayCast2D típus, amely a tárgy lebegő animációját vezérli, amíg az a pályán van.



A *gameScene* node alatt található egy *Viewport* típusú node, amelyhez a UI példány lesz létrehozva. Ez azért van így, hogy a UI el legyen választva a játéktérrel, és statikus maradjon a képernyőn.

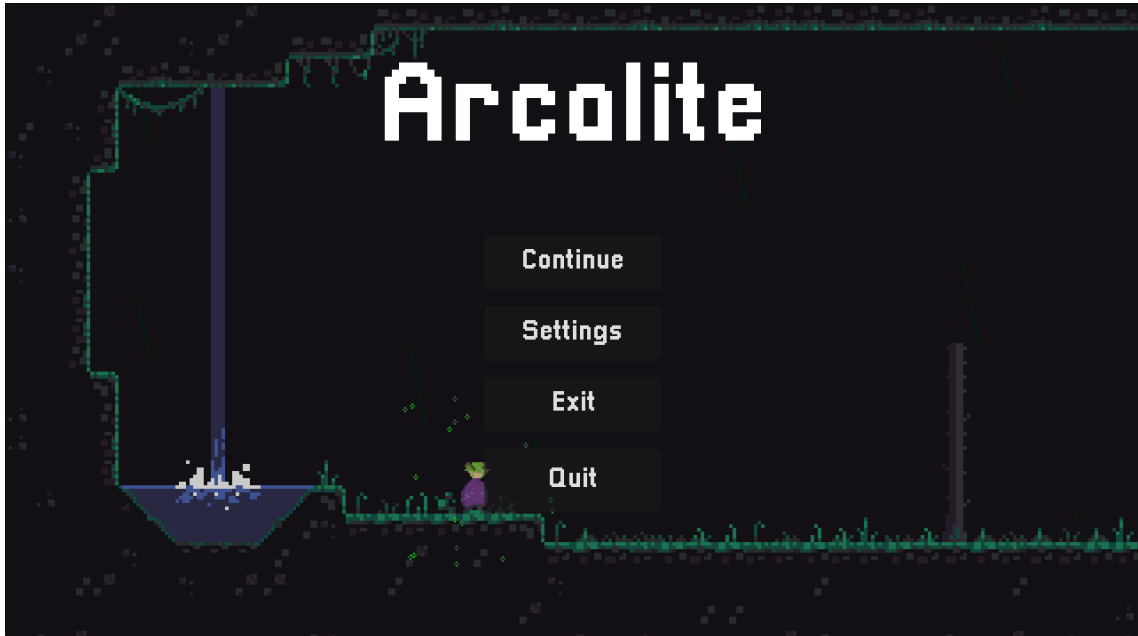
A UI elemek a képernyő két sarkában vannak elhelyezve. A statisztikák a bal felső sarokban találhatók, míg a visszatöltési idők (cooldown) a jobb felső sarokban. Az Életerő (Health) és Mana statisztikák sávjai sprite elemek kombinációjából állnak, valamint opcionálisan tartalmaznak egy szövegmezőt is a számértékek megjelenítésére. Emellett van egy harmadik statisztika, a sebzés (damage), amely egy szövegmezőben, számként jelenik meg.



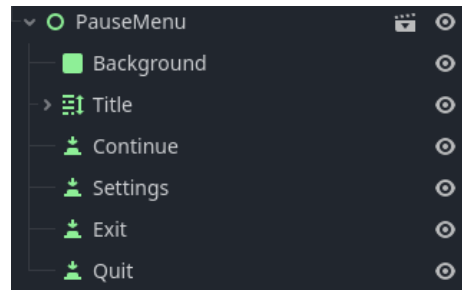
A visszatöltések két sprite elemből állnak: az egyik a cooldown-t mutatja, ami egy téglalapként jelenik meg, amelyet a kód alakít át töltősávvá; a másik sprite az összes képesség ikonját tartalmazza. A megjelenített képesség a kódon keresztül kerül továbbításra a cooldown modulnak, amely ezután az ikon állapotát "Készenléti" (Ready) módba állítja, miközben elrejtí a töltősávot. A képességek két vagy három állapottal rendelkeznek: *Készenléti* (Ready), *Visszatöltés alatt* (Cooldown), és *Aktív* (Active) – az utóbbi csak akkor létezik, ha a képességnek van aktív hatóideje.

A játék kezdetén a felhasználó három cooldown ikont lát: a *Dash*, *ChargeAttack* és *Oracle* képességeket. Egy speciális tárgy felvétele után egy új cooldown ikon jelenik meg, amely szinkronizálva van a tárgy képességével. Ez a két extra ikon dinamikus; abban a sorrendben jelennek meg, ahogyan a játékos megszerzi őket.

A UILayer tartalmazza a PauseMenu node-ot is, amely akkor jelenik meg, amikor a felhasználó megnyomja az Escape billentyűt. A PauseMenu node nagyon hasonlít a főmenüben található menühöz a gombok tekintetében, azzal a kiegészítéssel, hogy tartalmaz egy szövegmezőt is, amely megjeleníti az aktuális mentés játékidejét. A gombok elrendezése a következő:



- Folytatás (Resume): Bezárja a PauseMenu-t.
- Beállítások (Settings): Megnyitja a beállítási menüt, amely ugyanazt a node-ot használja, mint a főmenüben lévő beállítási menü.
- Kilépés (Exit): Kilép a főmenübe.
- Kilépés az asztalra (Quit): Kilép az asztalra



2.6.3. Osztályok és Scriptek

2.6.3.1. Autoload-ok

A program indításakor a globális script-ek töltenek be először, más néven autoload-ok vagy singleton-ok. Az alábbiakban látható a betöltésük sorrendje és az, hogy mit csinál egy-egy script:

PreloadRegistry: Ez a script felelős minden olyan erőforrás tárolásáért és rendszerezéséért, amelyre bármely másik script hivatkozik. Ezek az erőforrások olyan node-okat tartalmaznak, amelyeket kódból példányosítunk, például lövedékeket és tárgyakat, de ide tartozik a játékos is, akit a pálya node példányosít, valamint olyan asset források, mint az egyedi kurzor és a tileset, amelyet a tilemap rétegek használnak. Az erőforrások egymásba ágyazott osztályokba vannak szervezve az egyszerűbb olvashatóság érdekében. Tartalmaz továbbá egy aszinkron feladatot is, amely figyeli a nagyobb erőforrások betöltési állapotát, és jelet küld a betöltőképernyőnek, ha azok sikeresen betöltődtek.

ConfigFájlHandler: Ez a script a beállítások háttérkezeléséért felel. Egy egyedi Godot-változót, a ConfigFájl-t használja tárolóként, és egy .ini fájlt az adatok tárolására. Példányosításkor az alábbiakat hajtja végre:

- Ellenőrzi, hogy létezik-e már mentett fájl, és ha nem, létrehoz egy újat az alapértelmezett beállításokkal.
- Lekéri a szükséges információkat, például az ablak méretét és azt, melyik képernyőn fut a játék.
- Lefuttatja a reset függvényt, amely szinkronizálja a settingChanges szótárat a beállításokat tartalmazó fájlal.
- Lefuttatja az applySettings függvényt, amely betölti a beállítási opciókat a játékba, és alkalmazza azokat.

A script az alábbi statikus változókkal rendelkezik:

- config: ConfigFájl típusú változó, amelyet arra használnak, hogy a beállításokat egy exportálható formátumba állítsák össze.
- SETTINGS_FÁJL_PATH: string típusú konstans, amely a beállításfájl elérési útját határozza meg.
- settingChanges: Egymásba ágyazott szótár (dictionary). Az első szint egy karakterláncot tartalmaz, amely a beállítás adott szekcióját jelöli, és egy másik

szótárhoz van párosítva. A szekciók a beállítási felület különböző lapjai, például: Játék, Videó, Hang és Irányítás. A belső szótár egy újabb karakterláncot tartalmaz, amely az egyes beállítások nevét jelöli, és egy Variant típust, amely egy dinamikus változótípus a Godot osztálykönyvtárából. A Variant tárolja a beállítás tényleges értékét.

- `resXValues`: Egész számokat tartalmazó tömb, amely a vízszintes felbontási értékeket tárolja. Erre azért van szükség, mert a beállításfájl csak a felbontás indexszámát menti el ezen értékek alapján.
- `resYValues`: Egész számokat tartalmazó tömb, amely a függőleges felbontási értékeket tárolja.
- `screenId`: Egész szám, annak a fizikai képernyőnek az indexe, amelyen a játék fut.
- `Window`: Ablak típus, amelyet a videóbeállítások (például felbontás és ablakmód) lekérésére és alkalmazására használnak.

Térjünk át a függvényekre:

`DefaultSettings()` – nincs visszatérési érték:

Ez a függvény egy alapértelmezett beállításokat tartalmazó fájlt hoz létre. Meghívja a program, ha a `Ready` függvény nem talál meglévő konfigurációs fájlt, ha hiba történik a konfiguráció alkalmazása során, vagy ha a felhasználó kézzel meghívja a beállítások menüfelületén található gombbal. Miután létrehozta az alapértékeket és kiírta a konfigurációs fájlt, a függvény hamisra állítja a globális logikai módosítót, amely egy érvénytelen konfigurációs fájl jelzéséért felelős.

`SaveSettings()` – nincs visszatérési érték:

Ez a függvény egy többszintű ciklust használ, amely végigmegy a `settingChanges` szótár minden egyes beállításán, és alkalmazza azokat a `config` változóra. Ezután kiírja a konfigurációs fájlt, majd meghívja az `ApplySettings` és a `ResetSTChanges` függvényeket.

`LoadSetting(section: string)` – `Dictionary<string, Variant>` típusú visszatérés:

Egy konkrét szekciót vár bemenetként, majd beolvassa a konfigurációs fájlból az adott szekció összes beállítását. Végül az adatokat egy szótárba formázza, amelynek felépítése megegyezik a `settingChanges` belső szótárával, és ezt a szótárat adja vissza.

`checkSettings()` – boolean visszatérési érték:

Végigmegy a konfigurációs fájlban szereplő minden beállításon, és összeveti a `settingChanges` változóval. Ha bármelyik beállítás eltér, az azt jelzi, hogy történt változás, így

false értéket ad vissza. Ez értesíti a program többi részét, hogy a beállításokat el kell menteni vagy vissza kell állítani. Ha nincs eltérés, a függvény true értékkel tér vissza.

ApplySettings() – nincs visszatérési érték:

Ez a függvény felelős a settingChanges változóban szereplő beállítások alkalmazásáért a játék kliensben. Először egy try ciklust indít. Ez a ciklus a hibakezelésért felelős – ha hiba történik, leállítja a folyamatot, és az invalidSettings logikai változót true értékre állítja.

ResetSTChanges() – nincs visszatérési érték:

Ez a függvény végigmegy az összes szekción, és a settingChanges változót szinkronizálja a konfigurációs fájl tartalmával, azaz visszaállítja a változásokat az aktuális beállításokra.

SaveLoadHandler:

Ez a script a mentésfájl exportálásáért és importálásáért felelős. Egyetlen statikus változója van: savepath – string típusú, a mentésfájl elérési útját határozza meg.

A függvénye a következő:

Save(több érték) – nincs visszatérési érték:

Több értéket vesz be, amelyeket új sorral elválasztva karakterláncba ír.

Az értékei a következők:

- Map: az aktuálisan aktív térkép node neve.
- Checkpoint: az aktuálisan aktív ellenőrzőpont node neve.
- Rooms Cleared: boolean lista, amely az összes ellenségvezérlő tisztítási állapotát tartalmazza. Ez szükséges ahhoz, hogy a már megtisztított szobák ne idézzenek újra ellenségeket új játékindításkor.
- MaxHP: a játékos maximális életerejé.
- MaxMP: a játékos maximális mana-ja.
- CurrentHP: a játékos aktuális életerejé.
- CurrentMP: a játékos aktuális mana-ja.
- Damage: a játékos sebzési értéke.
- Items: int lista, amely a két tárgy helyét jelöli. A tárgyak számokhoz vannak rendelve egy Enum segítségével.

- RunID: a mentésfájl azonosító száma. Ha ez helyi mentés, akkor -1, egyébként az adatbázis által rendelt szám.
- RunName: a mentésfájl neve, amit a játékos ad meg egy új játék indításakor.
- PlayTime: float, amely azt jelzi, hány másodpercig futott aktívan a játék ezen mentés során.

Miután létrejött a mentési karakterlánc, a függvény ellenőrzi, hogy a játékpéldányhoz tartozik-e bejelentkezett felhasználó. Ha igen, akkor megnézi, hogy a jelenlegi játékfutás (run) már létezik-e az adatbázisban – ha igen, frissíti azt; ha nem, új mentést hoz létre. Ha új mentést észlel, akkor a -1 értékű runID-t lecseréli az adatbázis által kiosztott azonosítóra, és frissíti vele a mentési karakterláncot is.

Végül a függvény kiírja a mentési adatokat egy .txt fájlba, és elmenti a savepath által meghatározott helyre.

CheckSave() – bool visszatérési érték:

Ellenőrzi, hogy létezik-e mentésfájl a savepath helyen. Ha igen a visszatérési érték true, ha nem false,

Load(save: string, alapértelmezés szerint üres string) – nincs visszatérési érték:

Ez a függvény beolvassa a mentésfájlból a változókat a játékba, amely alapján a program vissza tudja állítani a játékállapotot arra a pontra, ahol el lett mentve.

Ha bemeneti karakterláncot kap (azaz a save paraméter nem üres), akkor az azt jelenti, hogy az adatbázisból tölt be mentést, és a megadott karakterláncból állítja vissza a változókat.

Ha nincs bemeneti karakterlánc, akkor a helyi mentésfájlt olvassa be, és annak alapján tölti be az értékeket.

2.6.3.2. DBConnector

A fájlnak alapvetően három feladata van: Az első az adatbázissal való kapcsolat létrehozása. Az ehhez szükséges adatok (szervercím, port, adatbázisnév stb.) az osztály változóiban vannak eltárolva, melyek segítségével épül fel a connection string. Ez alapján az osztály a MySqlConnection osztály példányosításával létrehozza a kapcsolatot. A másik két funkció is ezt használja.

A második feladat a bejelentkezéskor a felhasználó adatainak lekérése. Lekérésre kerül a játékos azonosítója és felhasználóneve, illetve a külön játékmeneteinek adatai (név,

játszott idő, stb.) és a hozzátartozó legutolsó mentés. A lekért adatok eltárolására a fájlban található két struct segítségével történik. A UserData a játékos adatainak, illetve a CharacterData a játékos mentéseinek tárolására szolgál.

Az utolsó feladat az újonnan mentett játék feltöltése.

A funkciók a következők:

GetLastPlayerEntry(connstate: logikai érték, alapértelmezetten hamis) – egész számot ad vissza: Ez a függvény lekérdezést hajt végre az adatbázisban, hogy lekérje az utolsó mentésfájl-bejegyzést. Ez arra szolgál, hogy új mentés létrehozásakor lekérjük a saveId-t.

PrepareSave(playerID: egész szám, save: szöveg) – SaveState enumot ad vissza: Ezt a függvényt a SaveLoadHandler hívja meg, ha egy felhasználó be van jelentkezve. A függvény célja annak meghatározása, hogy az adott mentésfájl milyen állapotban van az adatbázisban. Ha a mentésfájl megtalálható az adatbázisban, frissíti annak játékidéjét, és „Existing” értékkel tér vissza. Ha nem található, létrehoz egy új mentésrekordot az adatbázisban, és „Created” értéket ad vissza. Ha kivétel történik, „None” értékkel tér vissza.

UploadSave(PlayerID: egész szám, save: szöveg) – logikai értéket ad vissza: Ez a függvény felelős a tényleges mentésfájl adatbázisba történő feltöltéséért. Először beilleszti a PlayerID-t a mentésfájlba, hozzárendeli azt az eddig lokális mentéshez kapcsolódó adatbázisrekordhoz. Ezután lekérdezést nyit, és új bejegyzést ad hozzá az adatbázishoz.

2.6.3.3. *Globals*

Ez a script olyan változókat tartalmaz, amelyeket több másik script-nek is el kell érnie.

- PlayerControl: boolean, azt jelzi, hogy a játékos irányíthatja-e a karakterét.
- hasSaveFájl: boolean, a SaveLoadHandler CheckSave függvényével együtt használják annak megállapítására, hogy van-e helyi mentésfájl.
- currentSave: string lista, az aktív mentésfájl sorait tartalmazza. A SaveLoadHandler ebbe a változóba tölti be az értékeket, és a többi script innen kéri le a számára szükséges adatokat.
- runName: az aktív mentésfájl neve, külön tárolva a könnyebb hozzáférés érdekében.
- runID: az aktív mentésfájl adatbázis-azonosítója, külön tárolva a könnyebb kezelés érdekében. Alapértelmezés szerint -1.

- `gameActive`: boolean, azt jelzi, hogy a játék aktív-e. A játék „nem aktív”, ha a szünetmenü meg van nyitva.
- `GRAVITY`: float konstans, az összes entitás által használt gravitációs érték.
- `spawnPoint`: `CheckPoint` típus, az aktív ellenőrzőpont teljes példányát tárolja, a mentésfájl és az újraéledési folyamat használja.
- `activeMap`: string, az aktív pálya node-nevét tárolja, a mentésfájl használja.
- `statScreen`: `statScreen` típus, a játékban található statisztikai képernyő node-jának hivatkozása.
- `playTime`: float, nyomon követi, hogy hány másodperc telt el az aktuális játékfutás során.
- `gameBeaten`: boolean típus, akkor lesz true, ha a végső főellenséget legyőzték – ezzel jelzi, hogy az aktuális mentés befejezettnek számít.
- `player`: `Player` típus, a játékos példányának hivatkozása. Azért létezik globálisan, mert sok scriptnek szüksége van a játékos elérésére.
- `Difficulty`: int, a beállítások menü által beállított érték, a `diffMultipliers` indexelésére szolgál.
- `diffMultipliers`: float három nehézségi szinthez tartozó módosító értékeket tartalmaz. Az ellenségek statisztikáinak skálázására használják.
- `PopupResult`: boolean, a felugró ablakok eredményét jelzi.
- `PopupOpen`: boolean, akkor lesz true, ha aktív felugró ablak van a képernyőn.
- `invalidSettings`: boolean, a konfigurációs fájl sérülését jelzi.
- `buttontoggle`: int, a `HotkeyRebindButtons` által használt érték, megadja, hogy az elsődleges vagy a másodlagos billentyű kerül-e újrakötésre. Ez szükséges annak biztosítására, hogy ne lehessen egyszerre két újrakötést elindítani.
- `user`: `UserData` típus, a jelenleg bejelentkezett felhasználó adatait tárolja.

A változókon kívül a `Globals` egyetlen függvényt tartalmaz:

`GameBeaten()` – nincs visszatérési érték:

Ezt a függvényt akkor hívja meg a program, amikor a játék befejeződik. Azért létezik, hogy ha a `gameBeaten` logikai értéken kívül más változást is végre kell hajtani ebben a pillanatban, akkor azt itt lehessen megtenni.

2.6.3.3.1. Enums

Ez a script tartalmazza a különböző enumokat amit használ a játék.

ItemType

Meghatározza a tárgy típusát, példányosításnál és fájlba mentésnél hívja meg a program, a felvett tárgyakra hivatkozik.

Elemei:

- üres – 0: null állapot, sérült tárgyak kezelésére szolgál (az ilyen típusú tárgyak azonnal eltűnnek), valamint a játékos üres tárgyhelyeinek jelzésére.
- nyaklánc – 1: Nyaklánc típusú tárgy. Gyorslövés képesség.
- pajzs – 2: Pajzs típusú tárgy, a játékban „orichalcum core” néven ismert. Pajzs képesség.
- szilánk – 3: Szilánk típusú tárgy, véletlenszerűen hullik bármelyik ellenségtől, a játékos statjainak fejlesztésére szolgál.

EnemyClass

Az EnemySpawner-ek által használt enum, meghatározza, milyen típusú ellenséget idézzen meg.

Elemek:

- None: Nullállapot – az ilyen típusú spawner nem idéz meg semmit.
- LightMelee
- HeavyMelee
- LightRanged
- HeavyRanged
- Elite

BossClass

Szintén az EnemySpawner használja, meghatározza, milyen típusú főellenséget idézzen meg. Akkor érvényes, ha a spawner “BossSpawner” értéke true. Elemek:

- None: Nullállapot – az ilyen típusú spawner nem idéz meg semmit.
- Mech

MechAttackType

A MechBoss használja, amikor támadási jelet küld a karjainak. Meghatározza, hogy milyen típusú támadást hajtson végre az egyik vagy mindkét kar.

Elemek:

- Single
- Double
- Sweep
- Laser

SaveState

A DBConnector használja a mentési állapot jelzésére.

Elemek:

- Existing: a mentés már létezik az adatbázisban, csak frissítés szükséges.
- Created: a mentés még nem létezett az adatbázisban, új bejegyzés jött létre.
- None: nullállapot, hiba esetén használatos.
- Math: Ez a script más scriptek által használt matematikai függvényeket tartalmaz.

RNG(percentage: integer) – logikai értékkel tér vissza:
Ezt a függvényt annak eldöntésére használják, hogy egy ellenség elejt-e egy szilánkot. A függvény egy egész számot vár bemenetként, amit 0 és 100 közé korlátoz, majd összehasonlít egy 0 és 100 közötti véletlenszámmal. Ha a megadott százaléérték nagyobb, mint a véletlenszám, akkor true értékkel tér vissza, különben false.

2.6.3.4. Node osztályok

A GODOT-ban a játék mindegyik eleme egy úgy nevezett Node. Ez gyakorlatban egy Node2D nevű osztályból való származottságot jelenti. A MainNode a játék indításakor hozza létre a fő Node-ot, mely alá a játék többi eleme tartozni fog. Emellett a töltőképnyő és a főmenü Node-jait is legenerálja, illetve elindítja az adatok betöltését.

2.6.3.4.1. Menü mappa

Ezek az osztályok Control típusú node-okhoz tartoznak, amelyek a felhasználói felület (UI) kezeléséért felelnek. Mindegyik tartalmaz változókat a gyermekelemeikhez, amelyeket a _Ready függvényben csatlakoztatnak, hogy a kódból elérhetők legyenek.

Továbbá tartalmaznak jel (signal) függvényeket is, amelyeket kifejezetten a node elemekkel való kapcsolódásra terveztek, és akkor futnak le, amikor a node kibocsátja a megfelelő jelet.

MainNode

Ez a script kezeli a betöltőképernyőt, a főmenü betöltését és az érvénytelen beállítások ellenőrzését. Példányosításkor betölti a loadingScreen node-ot gyermekként, és eltávolít minden egyedi egérkurzort.

Függvények:

- LoadingFinished() – void értékkel tér vissza:
Ezt a függvényt a gyermek loadingScreen hívja meg, amikor a PreloadRegistry async feladata megerősíti, hogy minden betöltődött. Ezután törli a betöltőképernyőt és betölti a főmenüt gyermek node-ként.
- Process(delta: double) – void értékkel tér vissza:
A Godot beépített függvénye, amely minden játékkockában (tick) meghívásra kerül. Ellenőrzi, hogy az invalidSettings értéke igaz-e. Ha igen, értesíti a játékost a hibás konfigurációról, és visszaállítja az alapértelmezett beállításokat.

MainMenu

Függvények:

- Ready: Példányosítási függvény, összeköti a hivatkozásokat, és beállítja a fiókol-dalt a bejelentkezési állapot szerint.
- Register_Pressed: Signal függvény, átirányít a webes regisztrációs oldalra.
- SignIn_Pressed: Bejelentkezési állapottól függően vagy törli a bejelentkezési ada-tokat, vagy megnyitja a bejelentkezési popupot.
- SignInPopup_Login: Signal függvény, a login popup eredményét kezeli: sikeres bejelentkezés esetén megjeleníti a felhasználónevet, elrejtja a regisztráció gombot, átnevezi a bejelentkezés gombot „Kijelentkezés”-re és középre igazítja; sikertelen bejelentkezés esetén „Vendég”-re változtatja a nevet, megjeleníti a regisztráció gombot és visszanevezi a gombot „Bejelentkezés”-re.
- StartPressed: Signal függvény, betölti a „Start” almenüt, és az submenuOpen lo-gikai értéket true-ra állítja.

- `SettingsPressed`: Signal függvény, betölti a „Beállítások” almenüt, és az `submenuOpen` logikai értéket `true`-ra állítja.
- `QuitPressed`: Signal függvény, bezárja az ablakot.
- `ButtonControls`: Megjeleníti vagy elrejtí az UI elemeket egy logikai bemenet alapján.
- `Process`: delta függvény, ellenőrzi, hogy betöltődött-e a tileset erőforrás, és ha igen, háttérként használja; a `ButtonControls` függvényt hívja a `submenuOpen` állapot alapján.

SubmenuStart

Függvények:

- `Ready`: Példányosítási függvény, összeköti a node hivatkozásokat, és beállítja a „Folytatás” gomb elérhetőségét annak alapján, hogy van-e mentés.
- `ButtonControl`: Beállítja az UI elemek láthatóságát a bemeneti logikai érték szerint.
- `SubmenuContinueNode_MenuClosed`: Signal függvény, akkor hívja meg a program, amikor a „submenuContinue” törlésre kerül, és `ButtonControl(true)`-t hív.
- `ContinuePressed`: Signal függvény, betölti a „Continue” almenüt, és ha a felhasználó be van jelentkezve, `ButtonControl(false)`-t hív, ha nincs, akkor elindítja a játékot a helyi mentésből.
- `NewGamePressed`: Signal függvény, példányosítja az `newGameLaunch` node-ot, és meghívja a `ButtonGroup(false)`-t.
- `NewGameLaunchNode_Cancel`: Signal függvény, ha a `newGameLaunch` megszakításra kerül, `ButtonGroup(true)`-t hív.
- `BackPressed`: Signal függvény, önmagát törli, és a főmenü `submenuOpen` változóját `false` értékre állítja.
- `Process`: delta függvény, az `Escape` gomb lenyomását figyeli, és ha igaz, meghívja a `BackPressed`-et.

SubmenuSettings

Változók (a node hivatkozásokon kívül):

- `popupOpen`: boolean, jelzi, hogy van-e aktív popup az almenü fölött.

- `isSaved`: boolean, jelzi, hogy a változások el lettek-e mentve a konfigurációs fájlba.
- `Rebinding`: boolean, megakadályozza, hogy az escape gomb kilépjen a beállításokból, miközben épp egy billentyű hozzárendelést törölünk (amely szintén escape-et használ).
- `Rebindtimer`: int, puffer az escape gomb zárolásához.

Függvények:

- `Ready`: Példányosítási függvény, node hivatkozások összekötése, próbálja frissíteni a beállításválasztókat, visszaállítja az alapbeállításokat és értesíti a felhasználót, ha fájl sérülést észlel.
- `Process`: delta függvény, figyeli az Escape gombot, és ha lenyomják, hívja a `BackPressed`-et, ha a `rebindtimer` aktív, visszaszámol, és ha az exit popup igazat ad vissza, meghívja az `Exit` függvényt.
- `UpdateSelectors`: Frissíti az összes beállítási lehetőség értékét a konfigurációs fájlban szereplő értékek alapján.
- `DiffSelect`: Jelszignál függvény, szinkronizálja a felhasználó által kiválasztott nehézségi szintet a `settingChanges`-szel.
- `DashModeSelect`: Jelszignál függvény, szinkronizálja a dash mód változását a `settingChanges`-szel.
- `WindowSelect`: Jelszignál függvény, szinkronizálja az ablakmód változását a `settingChanges`-szel.
- `ResSelect`: Jelszignál függvény, szinkronizálja a felbontás változását a `settingChanges`-szel.
- `VsyncToggle`: Jelszignál függvény, szinkronizálja a Vsync kapcsolását a `settingChanges`-szel.
- `MasterVolumeSlideEnded`: Jelszignál függvény, szinkronizálja a fő hangerő csúszka módosítását a `settingChanges`-szel.
- `MasterVolumeChanged`: Jelszignál függvény, szinkronizálja a fő hangerő számértékének változását a `settingChanges`-szel és a hozzá tartozó csúszkával.
- `MusicVolumeSlideEnded`: Jelszignál függvény, szinkronizálja a zene hangerő csúszka változását a `settingChanges`-szel.
- `MusicVolumeChanged`: Jelszignál függvény, szinkronizálja a zene hangerő számértékének változását a `settingChanges`-szel és a csúszkával.

- SFXVolumeSlideEnded: Jelszignál függvény, szinkronizálja az effekt hangerő csúszka változását a settingChanges-szel.
- SFXVolumeChanged: Jelszignál függvény, szinkronizálja az effekt hangerő számértékének változását a settingChanges-szel és a csúszkával.
- MasterVolumeSliding: Jelszignál függvény, beállítja a hangerő számértékét a csúszka aktuális értékére.
- MusicVolumeSliding: Ugyanaz, mint fent, de a zene hangerőre.
- SFXVolumeSliding: Ugyanaz, de az effekt hangerőre.
- BackPressed: Ellenőrzi, hogy a beállítások mentve lettek-e. Ha nem, felugró ablakot jelenít meg, amely a felhasználót választás elé állítja: megszakítja a kilépési folyamatot a mentéshez, vagy mentés nélkül kilép. Ha a beállítások mentve lettek, meghívja az Exit függvényt.
- ResetPressed: Jelszignál függvény, meghívja a ConfigFájlHandler DefaultSettings függvényét, majd az Update függvényt.
- SavePressed: Meghívja a ConfigFájlHandler SaveSettings függvényét, az isSaved logikai értéket true-ra állítja, majd meghívja az UpdateSelectors-t és végül a BackPressed-et.
- Exit: Meghívja a ConfigFájlHandler ResetSTChanges függvényét, majd elindítja a kilépési folyamatot annak függvényében, hogy mi a szülő node. Ha a szülő a főmenü, a submenuOpen logikai értéket false-ra állítja, majd megszűnik. Ha a szülő a szünet menü, akkor csak megszűnik. Ha egyik sem, kivételt dob és bezárja a programot. A függvény végén a popupResult értéket visszaállítja false-ra, az alapértelmezett értékre.

HotkeyRebindButton: A node referencia változókon kívül ez az osztály két InputEventKey típusú és két InputEventMouseButton típusú változót tartalmaz. Ezek tárolják a gomb újrakötéséhez tartozó művelet elsődleges és másodlagos billentyűit.

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat. Lekéri az általa reprezentált művelet elsődleges és másodlagos billentyűit, majd beállítja a művelet nevét és a kulcs szövegeit.

- Process: delta függvény, ha a script éppen bemeneteket figyel, letiltja a billentyű gombokat a kattintások elkerülése érdekében, és újra engedélyezi őket, amikor a figyelés leáll.
- listenMouseInput: Egérbemenetek figyelésére szolgál, és ezekből inputEventMouseButton típusú változókat hoz létre, hogy azokat a játékbeállításokhoz lehessen rendelni. Tartalmaz egy függvényt, amely beállítja, melyik gomb aktív, miközben a másikat letiltja.
- _UnhandledKeyInput: Billentyűzetesemények figyelésére szolgál. Ha eseményt észlel, meghívja a rebindActionKey-et az adott eseménnyel. Tartalmaz egy függvényt, amely kezeli az aktív gombot.
- SetActionName: Beállítja a művelet szövegét a hozzárendelt input map érték alapján.
- SetKeyText: Lekéri az adott bemenethez rendelt billentyűt a konfigurációs fájlból, és megjeleníti a gombon. Ha az adott billentyű egy egér oldalgomb, a "Xbutton" szöveget "Side"-ra cseréli.
- rebindActionKey: Két külön útvonallal rendelkezik az elsődleges és másodlagos billentyűk újrakötésére. A billentyűeseményt hozzáadja az input maphez és a settingChanges szótárhoz. Ha a billentyű az Escape, a művelet hozzárendelése törlésre kerül.
- Button1Toggled: Jelszignál, az elsődleges gomb aktiválásakor hívódik meg. Ha az érték true, elkezdi figyelni a billentyűleütéseket, miközben a többi HotkeyRebindButton-t letiltja. Ha false, újra engedélyezi őket és leállítja a figyelést.
- Button2Toggled: Ugyanaz, mint az elsődleges, de a másodlagos gombhoz.
- OnExiting: Jelszignál függvény, amely a node megszűnésekor hívódik meg. Ha mentés történik kilépés nélkül, az input map műveletét visszaállítja a példányosításkor elmentett kulcsra.

SubmenuContinue

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat, ellenőrzi, hogy van-e helyi mentésfájl, és hogy kapcsolódik-e fiókhoz. Ha nem, akkor a lista tetejére kerül, "local" megjelöléssel. Ezután lekéri a bejelentkezett fiókhoz tartozó

mentéseket és listázza őket. Ha ezek után nincs elérhető mentés, figyelmeztető üzenetet jelenít meg a játékos számára.

- OnBackPressed: (Ez itt megszakadt – ha kéred, folytatom a fordítást innen.)
- Process: delta függvény, ellenőrzi, hogy az Escape billentyű le van-e nyomva, és ha igen, meghívja az OnBackPressed függvényt.

SaveItem

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat, lekéri a mentés azonosítóját, nevét és játékidéjét – ha adatbázisból származik, akkor a felhasználói adatokból, ha helyi mentés, akkor a mentésfájlból. Ezeket az adatokat használja a szövegmezők konfigurálásához.
- OnMouseEntered: Jelszignál függvény, amikor az egér fölé viszik, megváltoztatja a háttérszínt a kijelölés jelzésére.
- OnMouseExited: Jelszignál függvény, visszaállítja a háttérszínt, amikor az egér elhagyja a területet.
- GuiInput: Node függvény, az egérek kattintás észlelésére szolgál. Kattintáskor jelenetet vált a GameScene-re, és betölti a kiválasztott mentést.

NewGameLaunch

Függvények:

Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat.

- StartPressed: Jelszignál függvény, beolvassa a szövegmezőben megadott nevet. Ha a szöveg hossza nagyobb, mint 0, az új mentés a megadott nevet kapja. Ha nem, akkor az alapértelmezett “Új mentés” nevet használja. Jelenetet vált a GameScene-re, majd a mentés után beállítja a playerControl-t true-ra.
- Process: delta függvény, ellenőrzi, hogy le van-e nyomva az Escape billentyű, és ha igen, megszünteti önmagát.

LoadingScreen

Csak egy függvénye van:

- Ready: Példányosító függvény, elindítja a PreloadRegistry aszinkron folyamatot. A feladat befejeződésekor jelez a MainNode-nak.

Popup

Függvények:

- SetMessageType: A node példányosításakor használatos, beállítja az üzenet típusát. Meghatározza a felugró ablak funkcióját.
- OnCancelPressed: Jelszignál függvény, bezárja a felugró ablakot false eredménnyel.
- OnConfirmPressed: Jelszignál függvény, bezárja a felugró ablakot true eredménnyel.
- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat, és beállítja a felugró ablakot a típusának megfelelően.

SigninPopup

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat.
- Cancel_Pressed: Jelszignál függvény, a szülő submenuOpen logikai értékét false-ra állítja, bejelentkezési jelet küld, majd megszűnik.
- SignIn_Pressed: A megadott e-mail és jelszó alapján megpróbál bejelentkezni az adatbázisba. Siker esetén a submenuOpen érték false lesz, bejelentkezési jelet küld és megszűnik. Sikertelenség esetén hibát jelenít meg, amely leírja a hiba okát.

PauseMenu

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat. Fontos megjegyzés, hogy a ProcessMode változót Always-ra állítja – így a PauseMenu akkor is működik, amikor a játék szünetel, míg maga a játék megáll.
- Process: delta függvény, ellenőrzi, hogy lenyomták-e az Escape billentyűt, és ha igen, meghívja a ContinuePressed függvényt.
- ContinuePressed: Folytatja a játék feldolgozását és megszünteti önmagát.
- SettingsPressed: Példányosítja a beállítások node-ját gyermekként, és elrejtja a többi UI elemet.
- ExitPressed: Visszaállítja a ProcessMode állapotát, majd jelenetet vált a főmenüre (mainScene).

- QuitPressed: Kilépteti a programot.
- settingsClosed: Visszaállítja az UI elemeket.

RespawnScreen

Függvények:

- Ready: Példányosító függvény, csatlakoztatja a node hivatkozásokat.
- Quit_Pressed: Kilépteti a programot.
- MainMenu_Pressed: Jelenetet vált a MainScene-re.
- Respawn_Pressed: Meghívja a Map script respawn függvényét, majd megszünteti önmagát.

2.6.3.4.2. *Globals mappa*

GameScene

Függvények:

- Ready: Példányosító függvény, amely csatlakoztatja a node hivatkozásokat, majd betölti a pályát a mentésállapot szerint. Ha egy mentésből történik a betöltés, akkor beállítja a runID, runName, az aktív pálya nevét, valamint a játékidőt. Új játék indításakor a pályát map_0-ra, a játékidőt pedig 0-ra állítja. Ezután betölti a pálya nevét a globális változókba (Globals), átnevezi a pálya node-ot „Map”-re, hogy ne okozzon konfliktust más scriptekkel, beállít egy egyedi célkereszt kurzort, és a gameActive változót true-ra állítja.
- DeathTimer_Timeout: Jelszignál függvény, amely meghívásakor példányosítja az újraéledési képernyőt (respawn screen).
- Process: Delta függvény, amely nyomon követi és kezeli a játékidőt, aktiválja a halálszámlálót, amikor a játékos meghal, valamint figyeli az Escape billentyű lenyomását. Ha ez megtörténik, példányosítja a szünet menüt.

2.6.3.4.3. Map mappa

Függvények:

- Ready: Példányosító függvény, amely csatlakoztatja a node hivatkozásokat. Ha a játék mentésből töltődik, minden olyan checkpointot töröl, amely a mentésben tárolt checkpoint előtt található. Ezután a megadott checkpointot csatlakoztatja a Globals referenciához. Új játék esetén az első checkpoint kerül csatlakoztatásra. Ellenőrzi, hogy a csatlakoztatott checkpoint finalCheckpoint értéke true-e, és ha

igen, meghívja a GameBeaten függvényt. Végül meghívja a SetRoomStatus függvényt, majd a játékos node-ot példányosítja gyermekként.

- SetRoomStatus: Végigmegy a pálya összes ellenségvezérlőjén (EnemyControllers), és beállítja azok roomCleared értékét. Ha mentésből tölt, a mentés adatai alapján határozza meg az értéket, egyébként minden ellenségnél false-t állít be.
- Respawn: Elindítja az újraéledési folyamatot. Meghívja a kamera RespawnMove függvényét, eltávolítja az összes ellenséget, majd meghívja a SetRoomStatus és a játékos SetStats függvényét.
- roomStatus: Egy logikai értékeket tartalmazó listát ad vissza, amely az összes ellenségvezérlő roomCleared értékét tartalmazza.
- OnCameraMoved: Jelszignál függvény, amely a kamera vízszintes mozgása alapján mozgatja az esőeffektet, hogy az kövesse a játékost. Az esőeffekt nem globális, így optimalizálja az erőforrás-használatot.

2.6.3.4.4. *Game mappa*

Player

A *Player* script a játék egyik legfontosabb és leghosszabb scripte, mivel ez felelős a játékos és a játékkörnyezet közötti interakcióért.

Változók:

- defBH_X: int, a játékos találati dobozának X mérete.
- defBH_Y: int, a találati doboz Y mérete.
- isCrouching: boolean, jelzi, hogy a játékos guggol-e.
- crouchDown: boolean, a guggolás aktiválódását jelöli.
- crouchUp: boolean, a guggolás befejeződését jelöli.
- maxSpeed: int, a maximális mozgási sebesség.
- vel: float, a játékos aktuális sebessége.
- jumpStrength: int, az ugráskor a függőleges sebességhez hozzáadott érték.
- GRAVITY: float konstans, a Globals gravitációs értékére hivatkozik.
- prevDir: float, a játékos előző mozgásirányának meghatározásához.
- dashed: boolean, dash aktiválására szolgál.
- dashSpeed: float, a dash során fellépő gyorsulás.
- dashDecayRate: float, amely meghatározza a dash sebességcsökkenésének ütemét.

- `currentDashSpeed`: A dash aktuális sebessége.
- `isDashing`: boolean, jelzi, hogy a játékos éppen dash-el-e.
- `dashVector`: `Vector2` típus, a dash irányát tartalmazza.
- `CAisCharging`: boolean, jelzi, hogy a charge támadás töltés alatt áll-e.
- `CACharge`: float, a charge töltési szintje.
- `chargelevel`: int, a charge támadás szintje.
- `speedmodifier`: float, amely a mozgási sebességet módosítja töltés közben.
- `BADispersion`: int, az alap támadás szóródását módosítja.
- `spellItemE`: `itemType` típusú felsorolás, az első képességtárgyat jelöli.
- `spellItemQ`: `itemType` típusú felsorolás, a második tárgyképességet jelöli.
- `rapidFire`: boolean, a gyorslövés képesség aktiválási állapotát jelöli.
- `shielded`: boolean, a pajzs képesség aktiválási állapotát jelöli.
- `isHurt`: boolean, jelzi, hogy a játékos sérült állapotban van-e.
- `isDead`: boolean, jelzi, hogy a játékos halott állapotban van-e.
- `Resting`: boolean, a játékos pihenési állapotát jelöli.
- `HPRechargeAmount`: float, azt az életerőmennyiséget jelöli, amit a játékos regenerál pihenés közben.

Függvények:

`Ready()` – nincs visszatérési érték

Példányosításkor fut le. Csatlakoztatja önmagát a `Globals` referenciához, valamint a gyerekelemeket a referenciaváltozókhoz. Meghívja a `setStats` függvényt, és a játékost az aktív checkpoint pozíciójába helyezi.

`getInputs()` – `Vector2`

Ez a függvény minden játéktick során rögzíti a játékos bemeneteit. Először kiszámítja a vízszintes irányt a lenyomott gombok alapján, és hozzáadja egy vektorhoz. Ezután a játékos sprite-ját a megfelelő irányba fordítja. Ellenőrzi, hogy az ugrás gomb le lett-e nyomva, és ha igen, hozzáadja a megfelelő értéket a vektorhoz. Ha nem, és nincs lefelé ütközés (`isOnFloor == false`), lefelé mutató vektort ad hozzá. Miután összeállította az irányvektort, ellenőrzi a guggolás gombot, és ennek megfelelően állítja be az `isCrouching` változót. Végül ellenőrzi a dash gomb lenyomását. Ha megtörtént, és a `DashMode` „8 direction” módban van, akkor az aktuálisan lenyomott mozgásirányokból

épít egy dash vektort, és jelez a dash elindítására.

Visszatér az összeállított irányvektorral.

Movement(delta: double) – nincs visszatérési érték

Ez a függvény vezérli a játékos mozgását. Először, ha a játékos dash állapotban van, figyelmen kívül hagyja a felhasználói bemeneteket, és a dash vektor alapján mozgatja a karaktert. Ez a rész felel a dash befejezéséért is, ha a sebesség eléri a küszöbértéket.

Ezután ellenőrzi, hogy minden feltétel adott-e a dash indításához, és ha igen, meghívja a `Dash()` függvényt.

Normál mozgás során meghívja a `getInputs()` függvényt, majd annak eredményét használva módosítja a játékos sebességét. A sebesség kiszámítása után meghívja a `CrouchApply()` és `Fall()` függvényeket.

Dash() – nincs visszatérési érték

Ez a függvény előkészíti a játékost a dash mozdulathoz. Ha a `DashMode` beállítás „Follow Mouse”, lekéri az egér pozícióját, és abból számítja dash irányt a játékos pozíciójához képest. Ezután beállítja a dash sebességet, `isDashing` értékét `true`-ra, `dashed` értékét pedig `false`-ra állítja.

Fall(delta: double) – nincs visszatérési érték

Ez a függvény lefelé irányuló gyorsulást alkalmaz a játékosra a zuhanás szimulálásához.

CrouchApply() – nincs visszatérési érték

Ez a függvény módosítja a játékos találati dobozát a guggolt állapothoz, és a sebességét is beállítja. Guggolás közben a vízszintes mozgás csökken, míg a lefelé irányuló gyorsulás nő.

BasicAttack() – nincs visszatérési érték

Ez a függvény felelős a `basicProjectile` node példányosításáért és konfigurálásáért. Először létrehozza a lövedék node-ot, majd a `GameScene` node alá rendeli. Ezután beállítja az irányát, forgását (a szóródás alapján), valamint a sebzés értékét.

ChargeAttack() – nincs visszatérési érték

Ez a függvény példányosítja és konfigurálja a `chargeProjectile` lövedéket. Először létrehozza a node-ot, majd a `GameScene` node alá rendeli. Ezután beállítja a pozícióját, irányát, továbbítja a `chargeLevel` értéket, és beállítja a sebzését. Végül jelez a felhasználói felületnek, hogy elindítsa a töltéses támadás visszaszámlálásának vizuális effektjét.

`SpellOracle()` – nincs visszatérési érték

Ez a függvény példányosítja és konfigurálja a `spellOracle` típusú node-ot. Először létrehozza, majd a `Map` node alá rendeli. Ezután beállítja a pozícióját és szintjét. Végül jelez az UI felé, hogy indítsa el a varázslat visszaszámlálási animációját.

`Spelle()` – nincs visszatérési érték

Ez a függvény felelős a játékos első tárgyhelyén lévő képesség aktiválásáért. Ha az első helyen egy nyaklánc típusú tárgy található, akkor meghívja a `RapidFireAbility` függvényt. Ha pajzs típusú tárgy van, akkor a `ShieldAbility` függvényt hívja meg.

`SpellQ()` – nincs visszatérési érték

Ez a függvény felelős a játékos második tárgyhelyén lévő képesség aktiválásáért. Ha ezen a helyen nyaklánc típusú tárgy van, a `RapidFireAbility` függvényt hívja meg, ha pajzs típusú, akkor a `ShieldAbility` függvényt.

`SpelleTimerTimeout()` – nincs visszatérési érték Ez a jelzéshez kapcsolt függvény felelős az első tárgyhelyhez tartozó képesség visszaszámlálási fázisának elindításáért az UI-ban, és az adott tárgy típusának megfelelő értékek visszaállításáért. Ha a tárgy „nyaklánc”, akkor `rapidFire` értékét `false`-ra állítja. Ha „pajzs”, akkor `shielded` értékét állítja `false`-ra, és visszaállítja a pajzshatás node-ját az alapállapotba.

`SpellQTimerTimeout()` – nincs visszatérési érték

Ez a jelzéshez kapcsolt függvény a második tárgyhelyhez tartozó képesség visszaszámlálásának indításáért felelős, valamint visszaállítja az értékeket az adott helyen lévő tárgy típusa alapján. „Nyaklánc” esetén `rapidFire` lesz `false`, „pajzs” esetén `shielded` lesz `false`, és visszaállítja a pajzshatás node-ját az alapértelmezettre.

`RapidFireAbility(slot: string)` – nincs visszatérési érték

Ez a függvény kezeli a gyorslövés (`rapidFire`) képesség hatásait. Először `rapidFire` értékét `true`-ra állítja, aktiválja a vizuális effektet, és elhasznál egy meghatározott mennyiségű manát. Ezután a megadott slot érték alapján jelez az UI felé, hogy állítsa az adott hely ikonját „Aktív” állapotra.

`ShieldAbility(slot: string)` – nincs visszatérési érték

Ez a függvény kezeli a pajzs (`shield`) képesség hatásait. Először `shielded` értékét `true`-ra állítja, aktiválja a vizuális effektet, és elhasznál egy meghatározott mennyiségű manát.

Ezután a megadott slot érték alapján jelez az UI-nak, hogy állítsa az adott slot ikonját „Aktív” állapotra.

PickupItem(itemtype: itemType enum, cooldown: float)

Ez a függvény akkor fut le, amikor a játékos felvesz egy tárgyat. A megadott tárgy típus alapján a következőket hajtja végre:

Nyaklánc vagy Pajzs: Ellenőrzi, hogy a játékos már rendelkezik-e ilyen típusú tárggyal, hogy elkerülje a duplikációt. Ha van üres hely, oda felszereli, majd beállítja az adott slothoz tartozó cooldown értéket.

Shard: Megnöveli a maximális és aktuális életerőt tízzel, valamint a sebzést kettővel.

Ha a tárgy nem „shard” volt, akkor jelez az UI-nak, hogy frissítse az új képességhez tartozó visszaszámlálást.

Hit(damage: float, hitVector: Vector2) – nincs visszatérési érték

Ez a függvény akkor hívódik meg, amikor a játékos bármilyen forrásból sebzést kap.

Először ellenőrzi, hogy a játékos sebezhető-e épp. Ha igen, csökkenti az életerejét a megadott sebzés mértékével, majd megszakítja a dash sorozatot, hogy ne induljon el az ütés animáció után. Ezután ellenőrzi, hogy az életerő nullára csökkent-e, ha igen, isDead értékét true-ra állítja, és a függvény befejeződik. Ha nem, isHurt értéket true-ra állítja, és ha a hitVector nem null, akkor a játékos sebességét az alapján módosítja. Elindít egy hurtTimer-t, ami ideiglenesen letiltja a mozgást a visszalökődés idejére.

HitTick(damage: float) – nincs visszatérési érték

Alternatív sebzéskezelés, például folyamatos sebzést okozó források (mint a Laser node) esetén. Csökkenti az életerőt a megadott értékkel, majd ellenőrzi, hogy nullára csökkent-e. Ha igen, isDead lesz true, és kilép.

OnHurtTimerTimeout() – nincs visszatérési érték

Ez egy jelzéssel hívott függvény, amely isHurt értékét false-ra állítja, amikor lejár az időzítő.

Animate() – nincs visszatérési érték

Ez a függvény vezérli az AnimatedSprite node animációit. A játékos aktuális állapota alapján állítja be, hogy melyik animáció játsszon le.

`Rest(state: Boolean, amount: float – alapértelmezett érték: 0)` – nincs visszatérési érték
 Ez a függvény beállítja a játékos `Resting` állapotát a `state` bemenet alapján, valamint az `HPRechargeAmount` értékét az `amount` alapján.

`Cooldowns()` – nincs visszatérési érték

Ez a függvény a játékos statisztikáinak újratöltéséért felelős. Egy `Timer node`-dal működik együtt, amely minden hívásnál elindul, és időzítő lejártával újra meghívja magát.

Alapértelmezés szerint minden alkalommal növeli a `mana` értékét eggyel. Ha a játékos `Resting` állapotban van, akkor minden ciklusban növeli az életerőt eggyel és csökkenti az `HPRechargeAmount` értékét eggyel.^x

Ha a `Resting` értéke `true`, akkor a párhuzamos időzítő (`tandem timer`) várakozási ideje 0,05 másodpercre áll be. Ellenkező esetben az időzítő egy másodpercre van állítva.

`SetStats()` – nincs visszatérési érték

Ez a függvény felelős a játékos statisztikáinak, visszaszámlálási értékeinek és tulajdonságainak beállításáért. Ha mentett fájlból tölt be, akkor a mentett adatok alapján beállítja a játékos maximális és aktuális életerejét és manáját, sebzését, valamint a tárgyállapotokat. Ha nem mentésből tölt, akkor az alapértékeket állítja be. Ezt követően beállítja a visszaszámlálási időzítők várakozási idejét, visszaállítja a mozgásváltozókat (pl. sebesség), hogy ne öröklődjön mozgás újraéledés után.

Továbbá visszaállít minden logikai (`Boolean`) változót (mint például `isDead`, `isHurt`, `isDashing` stb.) az alapértékeikre. Végül a játékost az aktív ellenőrzőpont pozíciójába helyezi, és jelet küld az UI-nak, hogy frissítse a visszaszámlálásokat.

`Update(delta: double)` – nincs visszatérési érték

Ez a függvény a fő `delta` ciklus szerepét tölti be a scriptben.

A következőkért felel:

- Figyeli a támadási és képességbillentyűk lenyomását, és meghívja a megfelelő függvényeket
- Kezeli az ütésből származó visszalökődés fokozatos leépülését
- Meghívja a Movement és Cooldown függvényeket
- Elindítja a vereség szekvenciát, ha azt érzékeli, hogy a játékos meghalt.

PhysicsProcess(delta: double)

A beépített delta ciklus. Meghívja az Update függvényt, ha a playerControl értéke true.

A játékos scriptje számos „Get” függvényt is tartalmaz, amelyek visszaadják a nevükben szereplő változók aktuális értékét.

Item

Függvények:

- Ready: Példányosítási függvény. Összeköti a node hivatkozásokat, és az adott típus alapján konfigurálja az item node-ot. Elindítja az alap lebegő animációt.
- Float: Felelős az item lebegő, mozgó animációjának folyamatos végrehajtásáért.
- OnTriggerAreaEntered
Jelzés alapú függvény. Akkor aktiválódik, amikor a belépő objektum a játékos. Ekkor meghívja a játékos PickupItem függvényét a megadott item adatokkal, majd az item eltűnik a pályáról.
- PhysicsProcess: Delta ciklus, meghívja a Float függvényt, és biztosítja, hogy az item ne ütközzön be a tereptárgyakba.

Enemy

Ez az osztály az alapvető ellenségosztály, amelyet különböző specifikus ellenség típusok örökölnek.

Változók:

- maxHP: float, maximális életerő
- currentHP: float, aktuális életerő
- Damage: float, az okozott sebzés értéke
- shardDropRate: int, annak esélye, hogy az ellenség halálakor shard-ot dob
- isSlowed : boolean, lassítva van-e az ellenség

- slowFactor: float, a lassítás mértéke
- Jumped : boolean, true, ha az ellenség megpróbált ugrani (dupla ugrás elkerülése)
- Attacked : boolean, igaz, ha az ellenséget megtámadták
- isDead : boolean, halott-e az ellenség
- playerInAtkRange : boolean, a játékos támadási hatótávon belül van-e
- isAttacking : boolean, támadás állapotban van-e
- isHurt : boolean, sebesült állapotban van-e
- lostVision : boolean, az ellenség elvesztette-e a játékost a látómezejéből
- bufferRan : boolean, időzítő visszaszámlálás után visszatért-e az ellenség idle állapotba
- attackFrame: int, az a frame, amikor a támadás sebzést okoz
- atkCD: float, a támadási visszaszámlálás ideje
- isChasing : boolean, üldözi-e a játékost
- isRoaming : boolean, szabadon mozog-e az ellenség (nem üldöz, nem támad)
- Direction: Vector2D típusú, az ellenség mozgási iránya
- roamSpeed: float, a maximális elmozdulás sebessége idle állapotban
- chaseSpeed: float, Az ellenség maximális sebessége üldözési állapotban
- Speed: float, Az ellenség aktuális sebessége
- jumpStrength: int, Az ugráskor a függőleges sebességhez hozzáadott elmozdulási érték
- prevDir: float, Az előző vízszintes mozgási irány
- hitVector: Vector2D típusú, Az ellenség támadásakor a játékosra ható vektor

Függvények:

- Ready: Létrehozási (inicializáló) függvény. Összeköti a node hivatkozásokat, és beállítja az chaseBuffer időzítő várakozási idejét.
- Move: Az ellenség alapvető mozgási függvénye, három fő részre bontható:
- Ugrás: Beállítja az ugrási sebességet, és jumped értékét true-ra állítja, ha teljesülnek a feltételek: az ellenség aktívan mozog, akadály van előtte, az akadály fölött van elég hely az átjutáshoz, és az ellenség a földön van.
- Nem üldözés: Ez a rész akkor aktív, ha az ellenség éppen nyugalmi állapotban van. Az ellenség ilyenkor véletlenszerűen választott irányban mozog a roamSpeed sebességgel.

- Üldözés: Akkor aktív, amikor az ellenség üldözi a játékost. Itt érvényesül az esetleges lassító hatás is, például egy "oracle" entitás miatt.
- Fall: Alkalmazza a lefelé irányuló sebességet, amikor az ellenség a levegőben van. Ha az ellenség földre ér, jumped false-ra áll.
- OnDirectionTimerTimeout: Jelzésalapú függvény. Nyugalmi állapotban használatos, és egy véletlenszerű időtartamot számol ki, amely alatt az ellenség egy helyben marad. Elindítja a roamCooldown időzítőt a kiszámolt értékkel.
- OnRoamCooldownTimerTimeout: Jelzésalapú függvény, amely az ellenség mozgási idejét számolja ki nyugalmi állapotban. A mozgás irányát a dirChoose függvény választja ki.
- dirChoose: Kiválaszt egy véletlenszerű irányt, amerre az ellenség mozogni fog.
- AtkCooldownTimeout: Jelzésalapú függvény, amely akkor hívódik meg, amikor az ellenség befejezte a támadást. Ha a játékos még mindig támadási távolságon belül van, újra támad. Egyébként isAttacking értékét false-ra állítja.
- AttackRangeBodyEntered: Jelzésfüggvény. Ha a belépő objektum a játékos, akkor playerInAttackRange értékét true-ra állítja.
- AttackRangeBodyExited: Jelzésfüggvény. Ha a kilépő objektum a játékos, akkor playerInAttackRange értékét false-ra állítja.
- EngageAttack: Lejátssza a támadási animációt, isAttacking értékét true-ra állítja, és leállítja az ellenség mozgását.
- Attack: Alap (shell) függvény, amelynek léteznie kell az alaposztályban, mert más osztályok hivatkoznak rá. Valós működését az öröklődő osztályok valósítják meg.
- Hit: Kezeli az ellenség sebződését. Csökkenti az életerőt a megadott értékkel, és ha fejezett támadás találja el, alkalmaz egy visszalökő sebességet. Lejátssza a sebződési animációt, isHurt értékét true-ra állítja.
- OnHurtTimerTimeout: Jelzésfüggvény, amely isHurt értékét false-ra állítja.
- OnChaseBufferTimeout: Jelzésfüggvény, amely isChasing értékét false-ra állítja, vagyis az üldözés megszűnik.
- Flip: Felelős az ellenséghez tartozó node-ok tükrözéséért az aktuális nézési irány alapján.
- Animate: Kezeli az ellenség sprite animációit.
- OnSpriteAnimationFinished: Jelzésfüggvény, amely az animációk végén végrehajtandó eseményeket kezeli. Ha támadó vagy sebződési animáció fejeződött be,

visszatér az idle animációra. Ha halál animáció ért véget, meghívja a Die függvényt.

- DropItems: Felelős a megfelelő tárgyak ledobásáért, amikor az ellenség meghal. Kezeli a 100%-nál nagyobb shard dobási esélyeket úgy, hogy kiszámolja a garantált dobásokat, majd véletlenszám-generátorral dönt a nem garantált esetekről. Emellett kezeli a képességtárgyak ledobását is, ha a megfelelő bemenetek rendelkezésre állnak.
- Die: Meghívja a DropItems függvényt, majd megszünteti az ellenség objektumot.
- Update: Az ellenség fő delta ciklusa. Feladata a Movement és Fall függvények hívása, valamint az Attack függvény hívása, ha a játékos hatótávolságban van. Meghívja a Flip függvényt a nézési irány alapján. Haláli állapotban letiltja az ütközéseket, hogy ne blokkolják a játékos lövedékeit. Frissíti a látómezőt (lineOfSight), elindítja a követési pufferidőt a játékos utolsó ismert pozíciójára, és szükség esetén megszakítja az üldözést.
- PhysicsProcess: Beépített delta ciklus. Visszaállítja a slowFactor értékét, ha nincs lassító hatás alatt, valamint meghívja az Update és Animate függvényeket.

LightMeele (Könnyű közelharci ellenség)

- Ready: Példányosítási függvény, beállítja az ellenség típus statjait.
- Attack: Felülírt függvény. Kiszámítja a hitVector értéket, majd meghívja a játékos Hit függvényét.

HeavyMeele (Nehéz közelharci ellenség)

- Ready: Példányosítási függvény, beállítja az ellenség típus statjait.
- Attack: Felülírt függvény. Kiszámítja a hitVector értéket, majd meghívja a játékos Hit függvényét.

LightRanged (Könnyű távolsági ellenség)

- Ready: Példányosítási függvény, további node hivatkozásokat csatol, és beállítja az ellenség statjait.
- Flip: Felülírt függvény, kiegészíti az alap Flip szekvenciát egy további node kezeléssel.
- EngageAttack: Felülírt függvény. A támadás megkezdéséhez kiegészítő feltételként előírja, hogy a játékos a célzási látómezőben legyen.

- Attack: Felülírt függvény. Felelős CasterProjectile típusú lövedék létrehozásáért és konfigurálásáért. Először példányosítja a lövedéket, majd a jelenetfához kapcsolja testvérként. Ezután beállítja az irányát, forgását és sebzés értékét.
- Update: Felülírt függvény. A playerInAttackRange érték beállításához hozzáadja feltételként a játékos látómezőben való jelenlétét.
- PhysicsProcess: Felülírt függvény, amely a célzó vonalat a játékos irányába forgatja.

HeavyRanged (Nehéz távolsági ellenség)

- Ready: Példányosítási függvény, további node hivatkozásokat csatol, és beállítja az ellenség statjait.
- Flip: Felülírt függvény, bővíti az alap Flip működését egy új node kezelésével.
- EngageAttack: Felülírt függvény. A támadási szekvencia csak akkor indul el, ha a játékos a célzó vonalon belül van.
- Attack: Felülírt függvény. Létrehozza és konfigurálja a CasterProjectile típusú lövedékeket. A node-ot példányosítja, majd hozzáadja a jelenetfához mint testvér. Ezután beállítja az irányt, forgást, és a sebzési értéket.
- Update: Felülírt függvény. A playerInAttackRange érték beállításához szükséges, hogy a játékos a látómezőben legyen.
- PhysicsProcess: Felülírt függvény, amely forgatja a célzási vonalat a játékos felé.

BossUndead (Főellenség – Élőholt)

- Ready: Példányosítási függvény, beállítja az ellenség típus statjait.
- SpecAttackAnimLeft_AnimationFinished: Kezeli a baloldali speciális támadás animációjának végét. Elrejt a node-ot, és visszaállítja az animációs keretet nullára.
- SpecAttackAnimRight_AnimationFinished: Kezeli a jobboldali speciális támadás animációjának végét. Elrejt a node-ot, és visszaállítja az animációs keretet nullára.
- Attack: Felülírt függvény, amely a hitVector kiszámításáért és a játékos Hit függvényének meghívásáért felelős.
- EngageSpecialAttack: Az EngageAttack függvény egy variánsa, amely egy további támadási típushoz kapcsolódik. Beállítja az isAttacking értéket igazra, leállítja a mozgást és elindítja a speciális támadás animációját.

- `SpecialAttackAnim`: A speciális támadás vizuális hatását elindítja és láthatóvá teszi.
- `SpecialAttack`: Ez a függvény felelős a `hitVector` kiszámításáért és a játékos `Hit` függvényének meghívásáért, ha a játékost a speciális támadás eléri.
- `SpecialAttackRangeBodyEntered`: Jelző függvény, ha a triggerelő body a játékos, akkor beállítja a `playerInSpecialAttackRange` értéket igazra.
- `SpecialAttackRangeBodyExited`: Jelző függvény, ha a triggerelő body a játékos, akkor beállítja a `playerInSpecialAttackRange` értéket hamisra.
- `Die`: Felülírt függvény, amely hozzáad egy extra `DropItem` hívást a nyaklánc típusú tárgy ledobásához.
- `OnSpriteAnimationFinished`: Felülírt függvény, amely kezeli a speciális támadás animációját, ha az befejeződik.
- `PhysicsProcess`: Felülírt függvény, amely a speciális támadások aktiválásáért és hívásáért felelős.

BossMech (Főellenség – Mechanikus)

Ez az osztály nem használja az alap ellenség osztályt, mivel a mechanikája és viselkedése eltér a normál ellenségekéktől.

Változók:

- `health`: float, az életpontokat jelzi.
- `damage`: float, a sebzés mértékét jelzi.
- `shardDropRate`: int, a shardok ledobásának esélyét jelzi legyőzés után.
- `ACTIVE`: boolean, a főellenfél induló animációjának elkülönítésére szolgál.
- `playerSide`: boolean, igaz, ha a játékos a főellenfél jobb oldalán van, hamis, ha bal oldalán.
- `startup`: boolean, igaz, ha az induló animáció folyamatban van.
- `dead`: boolean, a főellenfél halott állapotát jelzi.
- `vulnerable`: boolean, jelez, amikor a főellenfél támadható.
- `laserCount`: int, a főellenfél szükséges lézer támadások számát jelzi, mielőtt sebezhetővé válik.
- `laserSlashes`: int, a lézer támadásban végrehajtott vágások számát jelzi.
- `laserActive`: boolean, igaz, ha a lézer támadás folyamatban van.

- `playerInRoom`: boolean, igaz, ha a játékos ugyanabban a szobában van, mint a főellenfél.

Függvények:

- `Ready`: Példányosítási függvény, csatlakoztatja a node hivatkozásokat és beállítja a statisztikákat.
- `BossMechArm_AttackFinished`: Jelző függvény, amelyet az arm node-ok hívnak, amikor befejeződik egy támadási szekvencia. Elindítja a megfelelő támadási típus hűtési idejét. Ha a támadási típus lézer, ellenőrzi a `laserCount` értékét. Ha a `laserCount` nullával egyenlő, elindítja a sebezhetőség állapotot és visszaállítja a `laserCount`-ot.
- `BossMechArm_EyeFlash`: Jelző függvény, amelyet az arm node-ok hívnak. Lejátsza a specifikált szemvillanás animációt. Ez az animáció figyelmezteti a játékost, hogy egy támadás következik.
- `StartupSequence`: Elindítja az induló animációt.
- `OnAnimationFinished`: Jelző függvény. Ha a befejeződött animáció az induló animáció volt, elindítja a támadási hűtési időt, beállítja az idle animációt és az `ACTIVE` értéket igazra. Ha a befejeződött animáció a halál animációja volt, eltávolítja az objektumot.
- `InitiateSingleAttack`: Jelzést küld a játékoshoz közelebb lévő kar számára, hogy elindítson egy "egyszeri" típusú támadást.
- `InitiateDoubleAttack`: Jelzést küld mindkét kar számára, hogy elindítsák a "dupla" típusú támadást.
- `InitiateSweepAttack`: Jelzést küld a játékoshoz közelebb lévő kar számára, hogy elindítson egy "söpörő" típusú támadást.
- `InitiateLaser`: Jelzést küld mindkét kar számára, hogy elindítsák a "lézer" típusú támadást.
- `LaserPowerUp`: Létrehozza a lézer lövedéket, és hozzáadja mint gyermek elemet. Csatolja a szükséges hivatkozásokat és függvényeket, majd konfigurálja a statisztikákat és értékeket.
- `LaserPowerDown`: Ez a függvény meghívja a lézer "TurnOff" funkcióját.
- `LaserBeam_TreeExiting`: Jelző függvény, amely visszaállítja a lézerhez kapcsolódó statisztikákat.

- EngageLaser: Ez a függvény meghatározza a lézer mozgásának irányát, és létrehoz egy Tween node-ot, hogy kezelje a transzformációt.
- LaserGrace_Timeout: Jelző függvény, amely meghívja az EngageLaser-t.
- LaserMove_Finished: Jelző függvény, amelyet akkor hívnak meg, amikor a lézer befejezte a mozgást. Ellenőrzi, hogy a laserSlashes nagyobb-e, mint nulla. Ha igen, létrehoz egy időzítőt, LaserGrace néven, egy rövid inaktív időszakra. Ha a laserSlashes nulla, meghívja a LaserPowerDown függvényt.
- OpenCore: Elindítja a főmag animációját, megállítja a támadási időzítőket, beállítja a vulnerable értéket igazra, és elindítja a sebezhetőségi időzítőt.
- VulnerabilityTimeTimeout: Lejátssza a főmag bezáró animációját, folytatja a támadási időzítőket, beállítja a vulnerable értéket hamisra.
- Hit: Ellenőrzi, hogy a vulnerable értéke igaz-e. Ha igen, csökkenti az életpontokat a kapott sebzés mértéke alapján, és lejátssza a sebzés animációt. Ha az életpontok eléri a nullát, meghívja a Die függvényt.
- Die: Beállítja a dead értéket igazra, az ACTIVE értéket hamisra, kétszer meghívja a DropItems függvényt (egyszer a shardok ledobásához, másodszor a speciális tárgy ledobásához), lejátssza a halál animációt, és meghívja a Globals-ban található GameBeaten függvényt.
- DropItems: Ez a függvény felelős a megfelelő tárgyak ledobásáért, amikor az ellenséget legyőzik. Kezeli a shardok ledobásának esélyét, ha az érték meghaladja a 100-at, kiszámítja, hány garantált ledobás történik, majd egy véletlenszám-generáló (RNG) függvényt futtat a nem garantált ledobás meghatározásához. Kezeli a képesség tárgyak ledobását is, ha megfelelő bemeneti változók állnak rendelkezésre.
- Movement: A főellenfél nem mozog a hagyományos értelemben. Ez a script felelős a főellenfél balra és jobbra való eltolásáért a térképen, miközben a játékos felé dönthető.
- OnSpriteAnimationFinished: Jelző függvény, amely eseményeket kezel a sprite animációk alapján. Lejátssza az idle animációkat, amikor az egyszeri animációk befejeződnek.
- Update: Fő delta ciklus. Meghívja a StartupSequence-t, ha a playerInRoom igaz. Ha az ACTIVE értéke igaz, akkor a következőket végzi:
 - o Meghívja a movement függvényt,
 - o Meghívja a támadási függvényeket, ha az időzítőik befejeződtek,

- Kezeli a lézer sprite-jának és ütköződobozának pozícióját a vezető raycast-hez,
- Meghívja a játékos HitTick függvényét, ha a játékos ütközik a lézerrel.
- PhysicsProcess: Beépített delta ciklus, amely meghívja az Update függvényt.

A script tartalmaz “Get” típusú függvényeket is

BossMechArm

Változók:

- Acceleration: float, a mozgáshoz használt gyorsulás értéke.
- maxSpeed: float, a maximális elmozdulás értéke.
- SIDE: boolean, meghatározza, hogy a kar melyik oldalon található. A Godot editor-ban konfigurálható.
- Origin: Vector2 típus, a kar kezdőpozícióját tárolja.
- atkSequence: integer, amely jelzi, hogy melyik támadási szakaszban van a kar.
- singleAtkWindupTime: float, az idő, amelyet a kar vár, mielőtt elindít egy egyszeri támadást.
- doubleAtkWindupTime: float, az idő, amelyet a kar vár, mielőtt elindít egy dupla támadást.
- sweepAtkWindupTime: float, az idő, amelyet a kar vár, mielőtt elindít egy söprő támadást.
- laserAtkWindupTime: float, az idő, amelyet a kar vár, mielőtt elindít egy lézer támadást.
- spikeAmount: int típus, amelyet a támadások használnak annak kiszámításához, hány tüskét kell létrehozni.
- singleAtk: boolean, igaz, ha a kar egy egyszeri támadási szekvenciában van.
- doubleAtk: boolean, igaz, ha a kar egy dupla támadási szekvenciában van.
- sweepAtk: boolean, igaz, ha a kar egy söprő támadási szekvenciában van.
- laserAtk: boolean, igazra van állítva, ha a kar lézer típusú támadási szekvenciában van.
- playerInRange: boolean, igazra van állítva, ha a játékos támadási távolságban van.
- isSlowed: boolean, jelzi, hogy az ellenség lelassult-e.

- `slowFactor`: float, az érték, amely meghatározza, hogy mennyire lassul le az ellenség, amikor lassítják.

Függvények:

- `Ready`: Példányosítási függvény, amely összekapcsolja a node hivatkozásokat, és beállítja a karot a `SIDE` változó alapján.
- `LaunchSequence`: Meghívja a fő test `StartupSequence`-t, és indít egy időzítőt, hogy az aktív állapotot összehangolja a fő testtel.
- `LaunchSequenceTimer_Timeout`: Jelző függvény, amely engedélyezi az ütközést és elindítja az idle animációt.
- `OnPlayerDetectAreaEntered`: Jelző függvény, ha a kiváltó test a játékos, beállítja a `playerInRange`-et igazra.
- `OnPlayerDetectAreaExited`: Jelző függvény, ha a kiváltó test a játékos, beállítja a `playerInRange`-et hamisra.
- `AttackSignal`: Jelző függvény, amelyet a fő test hív meg, meghatározza az akciót az adott támadás típus alapján. Ha szükséges, beállítja a `spikeAmount`-ot, elindítja a támadási felkészülési időzítőt, beállítja a megfelelő támadási Booleant és az `atkSequence`-t egyesre.
- `AtkWindupTimeout`: Jelző függvény, amely elküldi a szem villanás jelzését a fő testnek, beállítja az `atkSequence`-t kettőre, és elindítja az `atkGrace` időzítőt.
- `AtkGraceTimeout`: Jelző függvény. Ha az `atkSequence` két volt a hívás idején, beállítja az `atkSequence`-t háromra. Ha a támadás típus lézer, akkor jelez a fő testnek. Ellenkező esetben meghívja a `ReturnToOrigin` függvényt.
- `SingleAttack`: Beállítja az `atkSequence`-t nullára, meghívja a `CreateSpikes` függvényt, ha a `spikeAmount` nagyobb, mint nulla, elindítja az `atkGrace` időzítőt.
- `DoubleAttack`: Beállítja az `atkSequence`-t nullára, meghívja a `CreateSpikes` függvényt, ha a `spikeAmount` nagyobb, mint nulla, elindítja az `atkGrace` időzítőt.
- `SweepAttack`: Kiszámítja a találati vektort és meghívja a játékos `Hit` függvényét.
- `CreateSpikes`: Felelős a `Spike` node-ok létrehozásáért és konfigurálásáért. Először beállítja a tüskék helyét balra és jobbra, ha a vektorok nincsenek beállítva. Létrehozza a bal oldali tüskét, hozzáadja a `Map` node-hoz, beállítja a helyét, meghívja a `SetStats` függvényt és beállítja a node nevét. Ugyanezt ismétli a jobb oldalra is. Létrehoz egy rövid időzítőt és összekapcsolja a `SpikeTimer_Timeout`-ot.

- SpikeTimer_Timeout: Jelző függvény, amely csökkenti egyet a spikeAmount-ból. Ha a spikeAmount nagyobb, mint nulla, a tuskék helyeit távolabb mozgatja a középponttól és meghívja a CreateSpikes-t.
- ReturnToOrigin: Meghívja a RotateElements-t nullás értékkel, így visszaállítja a forgást. Létrehoz egy tween-t, hogy visszahelyezze a kar pozícióját az eredeti pontra.
- Returnal_Finished: Jelző függvény, amelyet akkor hívnak meg, amikor a ReturnToOrigin által létrehozott tween befejeződött. Jelez a fő testnek, hogy a támadás befejeződött és beállítja a támadás Booleant hamisra.
- RotateTween_Finished: Beállítja magát nullára.
- MovementTween_Finished: Beállítja magát nullára.
- RotateElements: Forgatja a kart a bemeneti érték szerint, az adott időtartam alatt.
- Update: Fő delta függvény. Felelős azért, hogy mi történik a különböző atkSequence értékeknél, amikor a támadások folyamatban vannak.
- SingleAttack:
 - 1: A játékos fölött lebeg.
 - 2: Várakozik.
 - 3: Felgyorsul lefelé, meghívja a SingleAttack függvényt, amikor a tereppel ütközik.
- DoubleAttack:
 - 1: Mozog a cél pozíció felé.
 - 2: Várakozik.
 - 3: Felgyorsul lefelé, meghívja a DoubleAttack függvényt, amikor a tereppel ütközik.
- SweepAttack:
 - 1: Forog és mozog acél pozíció felé.
 - 2: Várakozik.
 - 3: Felgyorsul előre. Meghívja a SweepAttack függvényt, ha a playerInRange igaz ezen a ponton. Lassulni kezd, miután elér egy bizonyos pontot.
- PhysicsProcess: Beépített delta függvény, meghívja az Update függvényt.

Projectiles

Változók:

- Direction: Vector2 típus, meghatározza, hogy a lövedék milyen irányba fog repülni.
- damagePayload: float, meghatározza, mekkora kárt fog okozni a lövedék.
- targetHit: boolean, igazra vált, ha a lövedék ütközik valamivel.

Függvények:

- Ready: Példányosítási függvény, összekapcsolja a node hivatkozásokat és beállítja a Sprite animációt.
- HitTerrain: Meghívódik, amikor a lövedék ütközik a tereppel. Kiszámítja az ütközés normálját és animálja a lövedék robbanását annak megfelelően.
- HitEnemy: Meghívódik, amikor a lövedék ellenséges entitással ütközik. Meghívja az ellenség Hit függvényét és lejátssza a robbanás animációt.
- AnimationFinished: Jelző függvény, amely elpusztítja a lövedéket, amikor egy animáció befejeződik.
- Process: Delta függvény, amely irányítja a sebességet és ellenőrzi az ütközéseket.

ChargeProjectile:

Változók:

- chargeLevel: int, a lövedék töltöttségi szintjét jelzi.
- Direction: Vector2 típus, meghatározza, hogy a lövedék milyen irányba fog repülni.
- damagePayload: float, meghatározza, mekkora kárt fog okozni a lövedék.
- Imports: boolean, konfigurációk alkalmazására szolgáló jelző.
- TargetHit: boolean, igazra vált, ha a lövedék ütközik valamivel.

Függvények:

- Ready: Példányosítási függvény, összekapcsolja a node-hivatkozásokat, és beállítja a Sprite animációt.
- HitTerrain: Akkor hívódik meg, amikor a lövedék tereppel ütközik. Kiszámítja az ütközés normálját, és annak megfelelően animálja a robbanást.

- **HitEnemy:** Akkor hívódik meg, amikor a lövedék ellenséges entitással ütközik. Meghívja az ellenség Hit függvényét és lejátssza a robbanás animációt.
- **AnimationFinished:** Jelzőfüggvény, amely az animáció befejeződésekor lebontja az objektumot.
- **Process:** Delta függvény, beállítja a lövedék méretét és sebességét, majd a chargeLevel alapján megszorozza a damagePayload értéket. Irányítja a sebességet és ellenőrzi az ütközéseket.

SpellOracle

Változók:

- **targetPosition:** Vector2 típus, a node pozícióját jelöli.
- **Level:** int, a varázslat erejének szintjét jelzi.
- **slowFactor:** float, az érték, amellyel a varázslat lelassítja a célpontokat.
- **Slowing:** boolean, a lassító hatás aktív állapotát jelzi.
- **affectedEntities:** Node lista, a hatással érintett entitásokat tárolja.

Függvények:

- **Ready:** Példányosítási függvény, csatlakoztatja a node hivatkozásokat és elindítja a Sprite animációt.
- **OnAnimationFinished:** Jelzőfüggvény, irányítja a sprite animációkat. Egy adott animáció indítja el a lassító hatást, így ezen animáció lejátszásakor ez a függvény beállítja annak időtartamát. Ha a befejező animációval hívják meg, akkor meghívja a TreeExit függvényt.
- **OnBodyEntered:** Jelzőfüggvény, ellenőrzi, hogy a belépő test lelassítható-e, és ha igen, hozzáadja az affectedEntities listához.
- **OnBodyExited:** Jelzőfüggvény, eltávolítja az entitást az affectedEntities listából, ha ott van.
- **TreeExit:** Eltávolít minden entitást az affectedEntities listából, majd lebontja önmagát.
- **Process:** Delta függvény, meghatározza a lassítás mértékét. Amikor a Slowing igaz, akkor alkalmazza a lassító hatást a listán szereplő entitásokra, ha hamis, eltávolítja azt.

CasterProjectile

Változók:

- Direction: Vector2 típus, meghatározza, milyen irányba repül a lövedék.
- damagePayload: float, meghatározza a lövedék által okozott sebzést.
- targetHit: boolean, igazra vált, ha a lövedék ütközött valamivel.
- isSlowed: boolean, jelzi, hogy a lövedék lassítás hatása alatt áll-e.
- slowFactor: float, az érték, amellyel a sebesség csökken, ha le van lassítva.

Függvények:

- Ready: Példányosítási függvény, csatlakoztatja a node-hivatkozásokat, és beállítja a Sprite animációt.
- HitTerrain: Akkor hívódik meg, amikor a lövedék a tereppel ütközik. Kiszámítja az ütközés normálját, és annak megfelelően animálja a robbanást.
- HitPlayer: Akkor hívódik meg, amikor a lövedék a játékosal ütközik. Meghívja a játékos Hit függvényét, majd lejátsza a robbanás animációt.
- OnAnimationFinished: Jelzőfüggvény, amely az animáció végén lebontja az objektumot.
- Process: Delta függvény, alkalmazza a lassító hatást (ha szükséges), irányítja a sebességet, és ellenőrzi az ütközéseket.

Laser

Egyetlen egyedi változója van, a speedScale: float.

Függvények:

- Ready: Példányosító függvény, összeköti a node-hivatkozásokat, speedScale értékét egyre állítja, elindítja az összes szegmens sprite animációját.
- RotateBeam: Beállítja a node forgását.
- SetImpactPoint: Beállítja annak a sprite-nak a pozícióját, amely a becsapódás animációját játssza le.
- OnAnimationFinished: Jelzőfüggvény, irányítja a szegmens sprite-okat. Ha az intro animáció befejeződött, átvált a loop animációra. Ha az outro animáció fejeződött be, lebontja az objektumot.
- TurnOff: A speedScale értékét -1-re állítja, és az intro animációkat visszafelé lejátsza, így szimulálva az outro animációkat.

Spike

Változók:

- Damage: float típus, a sebzést jelenti.
- Facing: Boolean típus, jelzi, melyik irányba néz a tüske.
- playerInArea: Boolean típus, akkor igaz, ha a játékos a tüske sebző területén belül van.
- lastSpeedScale: float típus, azt mutatja, hogy az animáció visszafelé lett-e lejátszva.

Függvények:

- Ready: Példányosítási függvény, összeköti a node-hivatkozásokat.
- SetStats: Beállítja a node értékeit. Meghatározza a sebzést és az irányt. Véletlenszerűen forgatja és skálázza a node-ot. A lastSpeedScale értékét 1-re állítja.
- OnHitBoxAreaEntered: Jelzőfüggvény, ha a triggerelő test a játékos, playerInArea értékét true-ra állítja.
- OnHitBoxAreaExited: Jelzőfüggvény, ha a triggerelő test a játékos, playerInArea értékét false-ra állítja.
- StillTimerTimeout: Jelzőfüggvény, visszafelé lejátssza az animációt és a lastSpeedScale értékét -1-re állítja.
- OnSpriteAnimationFinished: Jelzőfüggvény. Ha az animáció normálisan lett lejátszva, elindítja a stillTimer-t. Ha visszafelé lett lejátszva, lebontja az objektumot.
- Process: Delta függvény, sebzést alkalmaz a játékosra, amikor a tüske animációja egy bizonyos képkockához ér. Teljes sebzés csak akkor történik, ha a tüske kifelé mozgásban van, egyébként a játékos Hit függvénye 5-ös sebzéssel hívódik meg.

2.6.3.4.5. Mechanics mappaUI

Függvények:

- Ready: Példányosítási függvény, összekapcsolja a node-hivatkozásokat.
- Player_Dashed: Jelzőfüggvény, elindítja a dash ikon hűtési módját.
- DashCooldown_Finished: Tween függvény, elrejt a dash cooldown sávot, és az ikont "Kész" állapotra váltja.

- Player_ChargeAttacked: Jelzőfüggvény, elindítja a töltött támadás ikon hűtési módját.
- CACooldown_Finished: Tween függvény, elrejt a töltött támadás cooldown sávot, és az ikont "Kész" állapotra váltja.
- Player_OracleCast: Jelzőfüggvény, elindítja az Oracle ikon hűtési módját.
- OracleCooldown_Finished: Tween függvény, elrejt az Oracle cooldown sávot, és az ikont "Kész" állapotra váltja.
- Player_RapidFireCast: Jelzőfüggvény, a RapidFire ikont "Aktív" állapotra állítja.
- Player_SpellEOver: Jelzőfüggvény, elindítja az E varázshely cooldown módját.
- SpellECooldown_Finished: Tween függvény, elrejt az E varázshely cooldown sávját, és "Kész" állapotra váltja.
- Player_ShieldCast: Jelzőfüggvény, a Shield ikont "Aktív" állapotra állítja.
- Player_SpellQOver: Jelzőfüggvény, elindítja a Q varázshely cooldown módját.
- SpellQCooldown_Finished: Tween függvény, elrejt a Q varázshely cooldown sávját, és "Kész" állapotra váltja.
- UpdateItems: Jelzőfüggvény, szinkronizálja a cooldown ikonokat a játékos felszerelt tárgyaival.
- UpdateStats: Szinkronizálja az állapotsávokat és a sebzéskijelzést a játékosal.
- Process: Delta-függvény, a játékos példányosításának befejeztével csatlakoztatja a jelzőfüggvényeket, minden képkockában meghívja az updateStats-ot.
- Player_ItemsModified: Jelzőfüggvény, meghívja az updateItems függvényt a játékos felszereléseivel, mint bemeneti adat.

Checkpoint

Függvények:

- Ready: Példányosítási függvény, összeköti a node hivatkozásokat.
- TriggerAreaBodyEntered: Jelzőfüggvény, meghívja a Player_EnteredRestArea függvényt, és ha a checkpoint korábban még nem volt aktiválva, meghívja a TriggerCheckpoint függvényt.
- TriggerAreaBodyExited: Jelzőfüggvény, ha ez az adott térkép első checkpointja (0), mentést indít. Minden esetben meghívja a Player_ExitedRestArea függvényt.
- OnAnimationFinished: Jelzőfüggvény, ha a befejezett animáció neve "triggered", meghívja az Empty függvényt.

- `TriggerCheckpoint`: Lebontási jelet küld az előző checkpointnak, és lecseréli a `Globals` hivatkozást saját magára, majd mentést indít.
- `Empty`: Az ikonképet "üres" képre váltja, majd elindítja a területi partikeleffektet.
- `Process`: Delta függvény, a játékos példányosításának befejeztével összeköti a jelzőfüggvényeket.
- `Player_ExitedRestArea`: Jelzőfüggvény, a játékos `Rest` függvényét hívja meg `false` értékkel.
- `Player_EnteredRestArea`: Jelzőfüggvény, a játékos `Rest` függvényét hívja meg `true` értékkel. Ha ez az első alkalom, amikor a checkpoint aktiválódik, akkor plusz 20 életerő-helyreállítással hívja meg a `Rest` függvényt. Extra ellenőrzés biztosítja, hogy egy checkpoint csak egyszer gyógyítson 20-at.

CameraController

Változók:

- `freeCamToggle`: boolean, jelzi, hogy a szabad kamera mozgás engedélyezett-e.
- `playerLock`: Boolean, jelzi, hogy a játékos jelenleg be van-e zárva a szobába.
- `Cooling`: Boolean, jelzi, hogy aktív-e a visszahűtési időzítő.

Függvények:

- `Ready`: Példányosítási függvény, összeköti a node hivatkozásokat.
- `RespawnMove`: Kikapcsolja a pozíció simítást, így a kamera azonnal mozog, kikapcsolja a `playerLock`-ot, és elindítja a visszahűtési időzítőt.
- `LockPlayer`: A bemeneti logikai érték alapján zárolja vagy feloldja a játékos mozgását.
- `MoveCamera`: Elmozdítja a kamerát a megadott irányban egy szoba hosszának vagy magasságának megfelelően.
- `CooldownTimerTimeout`: Jelzőfüggvény, visszakapcsolja a pozíció simítást.
- `Process`: Delta függvény, ellenőrzi, hogy a játékos másik szobába lépett-e, és ha igen, meghívja a `MoveCamera` függvényt. Ha `playerLock` aktív, korlátozza a játékos elmozdulását a kamera pozíciójához képest, így nem tudja elhagyni a szobát. Ha `freeCamToggle` aktív, figyeli a szabad kameramozgás billentyűit, és eszerint módosítja az eltolást.

EnemyControl

Változók:

- enemyAmount: int, az alá tartozó aktív ellenségek számát követi.
- playerInRange: boolean, igaz, ha a játékos egy szomszédos szobában van.
- playerInRoom: boolean, igaz, ha a játékos az adott szobában tartózkodik.
- enemiesActive: boolean, igaz, ha az ellenségek üldözési módban vannak, vagyis támadják a játékost.
- despawningInProgress: boolean, igaz, ha az ellenségek eltávolítása megkezdődött.
- roomCleared: boolean, jelzi, hogy a szobában az összes ellenség le lett-e győzve.
- camState: boolean, a CameraController playerLock változójának helyi reprezentációja.
- enemySpawnPoints: Node2D típusú lista, amely EnemySpawner típusú gyermek-elemek tárolására szolgál.

Függvények:

- Ready: Példányosítási függvény, összeköti a node hivatkozásokat, és minden alárendelt EnemySpawner node-ot hozzáad az enemySpawnPoints listához.
- AreaEntered: Jelzőfüggvény, érzékeli, ha a játékos belépett egy szomszédos vagy ugyanabba a szobába.
- AreaExited: Jelzőfüggvény, érzékeli, ha a játékos elhagyta a szomszédos vagy azonos szobát.
- OnTimerTimeout: Jelzőfüggvény, meghívja a DespawnEnemies függvényt.
- SpawnEnemies: Leállítja a Spawntimer elemet, meghívja az összes alárendelt EnemySpawner Spawn függvényét, és az activeEnemies változót az újonnan létrehozott ellenségek szerint állítja be.
- DespawnEnemies: Eltávolítja a specializált lövedékeket, valamint az összes általa irányított ellenséget. A kamera zárolását false-ra állítja, az activeEnemies értékét pedig 0-ra.
- SetRoomCleared: Beállítja a roomCleared logikai értéket a megadott értékre.
- Process: Delta függvény. Ha a roomCleared hamis, a következőket végzi:
 - A playerInRoom értékét false-ra állítja, ha a játékos meghal, hogy az ellenségek visszatérjenek alapállapotba, és ne üldözzék a legyőzött játékost.

- Ellenségeket idéz meg, ha a `playerInRange` igaz, és az ellenségek még nem aktívak.
- Elkezd az eltüntetés folyamatot, ha a `playerInRange` hamis, az ellenségek aktívak, és az eltüntetés még nem indult el.
- Megszakítja az eltüntetés folyamatot, ha a `playerInRange` ismét igaz lesz, miközben az folyamatban van.
- Ha `playerInRoom` igaz, meghívja a `LockPlayer(true)` függvényt, és támadási jelet küld minden spawnernek.
- Ha `enemyAmount` eléri a 0-t, miközben `playerInRoom` igaz, akkor a `roomCleared` értékét `true`-ra állítja.
- Ha `roomCleared` igaz, meghívja a `LockPlayer(false)` függvényt.
- `isRoomCleared`: Visszaadja a `roomCleared` logikai értéket.

EnemySpawner

Változók:

- `EnemyClass`: `EnemyClass` típusú enum; Export érték. Lehetővé teszi az ellenség típusának beállítását a Godot szerkesztőből.
- `BossSpawner`: boolean; Export érték. Átkapcsolja a spawner működését egyszerű ellenségről főellenségre. A Godot szerkesztőben módosítható.
- `BossClass`: `BossClass` típusú enum; Export érték. Lehetővé teszi a főellenség típusának beállítását a Godot szerkesztőből.
- `ActiveEnemy`: `Enemy` típus, a spawner által létrehozott ellenség referenciája.
- `ActiveBoss`: `MechBoss` típus, a spawner által létrehozott főellenség referenciája.

Függvények:

- `Ready`: Példányosítási függvény, összeköti a node hivatkozásokat.
- `Spawn`:
 - Ha `BossSpawner` igaz: a `BossClass` alapján idéz meg főellenséget, és létrehozza a referenciát. Sikeres idézés esetén `true`-t ad vissza.
 - Ha `BossSpawner` hamis: a `EnemyClass` alapján idéz meg ellenséget, és beállítja a referenciát. Sikeres idézés esetén `true`-t ad vissza.
- `Despawn`: A `playerInRoom` értékét `false`-ra állítja, lebontja az ellenséget vagy főellenséget, és nullázza a referenciát. Sikeres eltávolítás esetén `true`-t ad vissza.
- `Process`: Delta függvény, továbbítja a `playerInRoom` értéket a főellenségnek.

StatsScreen

Függvények:

- Ready: Példányosítási függvény, összeköti a node hivatkozásokat.
- UpdateStats: Szinkronizálja a szövegmezőket a pontos statisztikák megjelenítéséhez.
- Process: Delta függvény, meghívja az UpdateStats-ot, amikor érzékeli, hogy a játékot befejezték.

2.6.4. Függőségek

A Godot Engine lehetőséget nyújt egy kiterjedt, közösség által készített bővítménykönyvtár elérésére. Ebből a könyvtárból egy olyan bővítményt használtunk, amely lehetővé tette az .ase fájlok importálását és kezelését – ezeket az Aseprite program hozta létre.

Az összes felhasznált asset linkje:

- Tileset: <https://octoshrimpy.itch.io/tranquil-tunnels>
- Ellenségek:
 - o <https://xzany.itch.io/headless-horseman-2d-pixel-art>
 - o <https://xzany.itch.io/cyclops-2d-pixel-art>
 - o <https://xzany.itch.io/witch-2d-pixel-art>
 - o <https://xzany.itch.io/skeleton-warrior-2d-pixel-art>
 - o <https://xzany.itch.io/skeleton-mage-2d-pixel-art>
- Lövedékek & Effektek: <https://bdragon1727.itch.io/>
- Lézer: <https://kanpelle.itch.io/ultimate-laser-pack>
- Cooldown ikonok: <https://free-game-assets.itch.io/free-skill-3232-icons-for-cyberpunk-game>
- Kurzorkészlet: www.kenney.nl
- Nyaklánc: <https://ssugmi.itch.io/16x16-rpg-assets>

2.7. Tesztelés

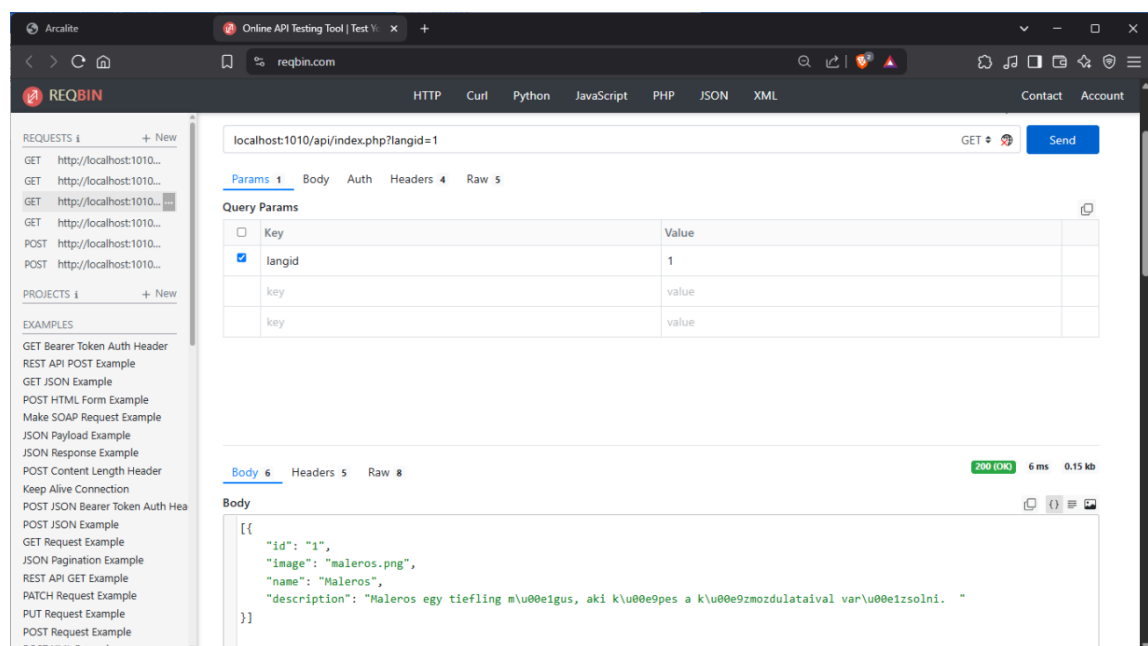
A szoftverfejlesztés nélkülözhetetlen lépése a tesztelés. Ezzel biztosíthatjuk, hogy a fejlesztett szoftver megbízhatóan működjék, illetve megfeleljen a célnak és a megrendelő elvárásainak.

A tesztelés elvégzését sokféleképpen megtehetjük, kezdve az egyszerű manuális tesztekkel, ahol mi magunk találunk ki tesztadatokat és teszteljük ki őket, egészen a tesztelésre készített programokig, melyek akár maguknak generálnak tesztadatokat, majd ellenőrzik, hogy az elvárt kimenetet kapták-e. Mi a projekt jellege és keretei alapján a manuális tesztelés mellett döntöttünk.

2.7.1. Backend tesztelés

Az API teszteléséhez a ReqBin tesztelőalkalmazást használtuk. A ReqBin egy ingyenes online tesztelőeszköz, mely egy böngészős bővítménnyel a helyi szerverre is tud API-hívásokat küldeni. A kezelőfelületen egyszerűen megadható az URL, a meghívás típusa és törzse, illetve egyből látjuk a kérésre kapott választ, így könnyen és gyorsan tudjuk az API-nkat tesztelni. Fontos megjegyezni, hogy az oldal tényleges API-hívásokat intéz, így azok ténylegesen módosíthatják az adatainkat – amennyiben persze ilyen funkciót is kínál az API-nk.

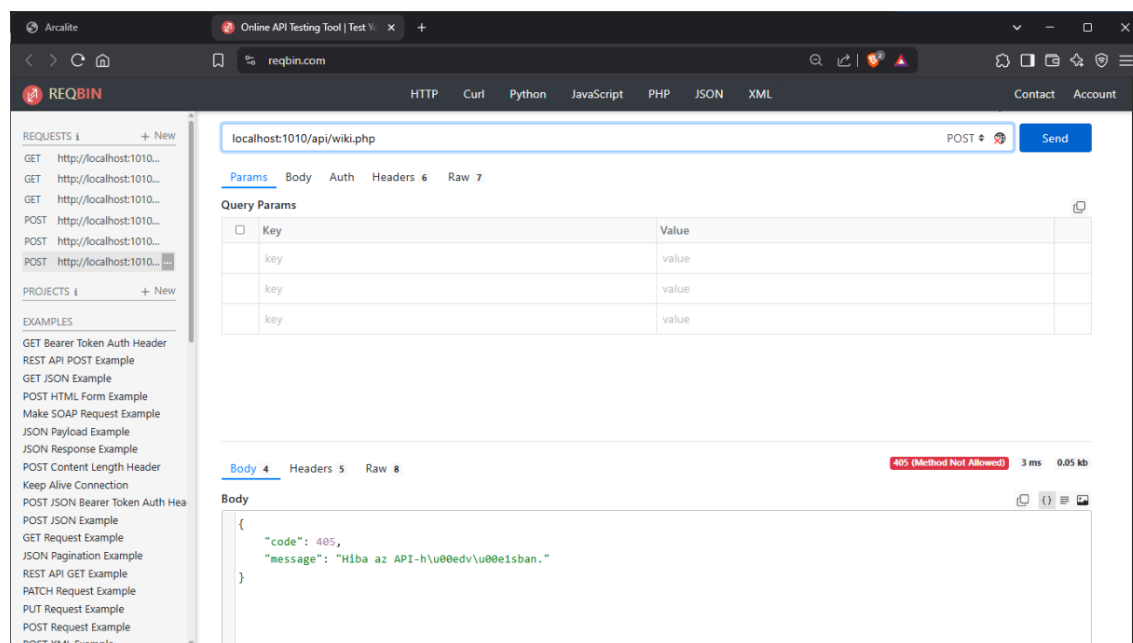
A következő oldalakon példákat hozunk a tesztelési folyamatból.



A ReqBin felülete

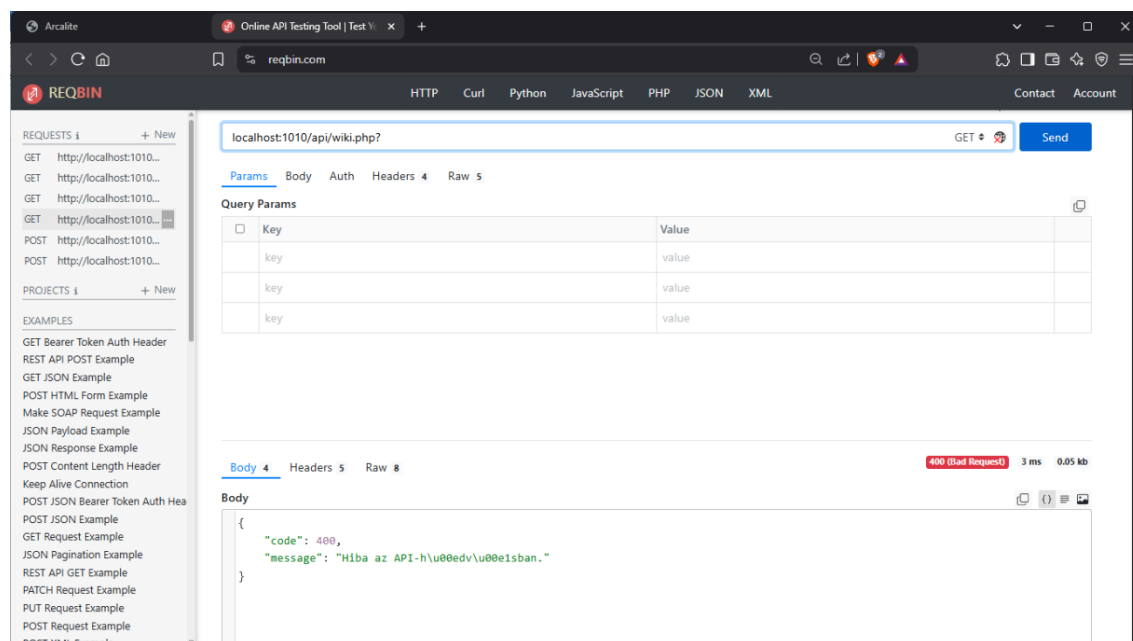
2.7.1.1. 1. teszteset: hibás metódus

Ha egy végpontot nem a megfelelő metódussal hívunk meg, eljárás szerint 405-ös (Method Not Allowed) válaszkódot kell visszaadnia, emellett JSON formátumban úgyis ezt a 405-ös kódot, illetve a „Hiba az API-hívásban.” üzenetet. A képen látható, hogy az elvárt kimenetet kaptuk.

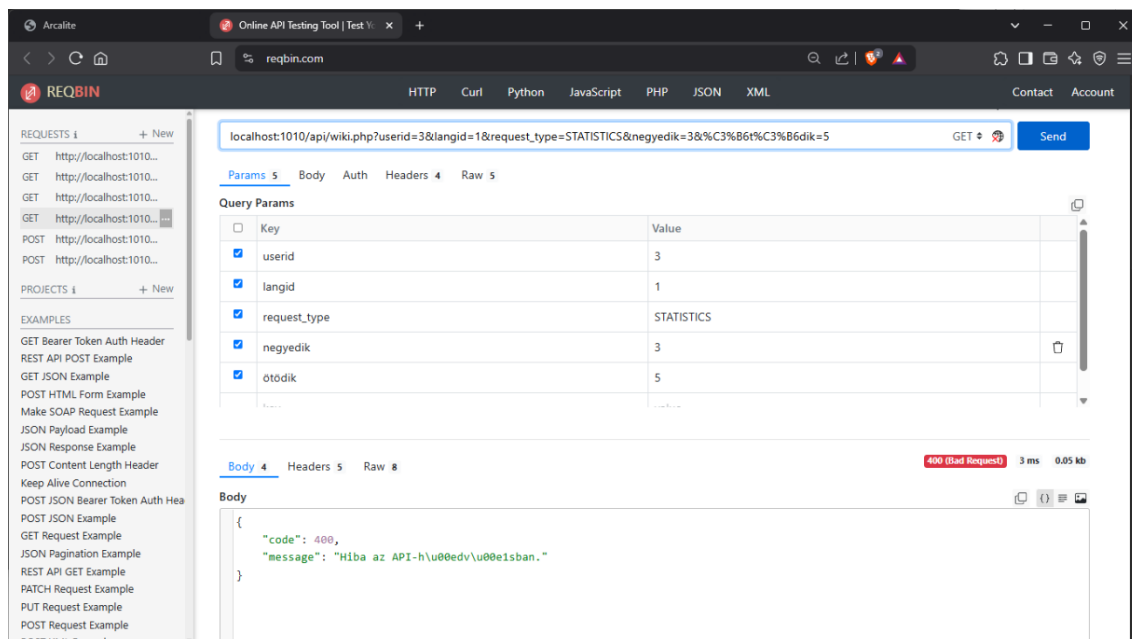


2.7.1.2. 2. teszteset(-csoport): hibás bemenet

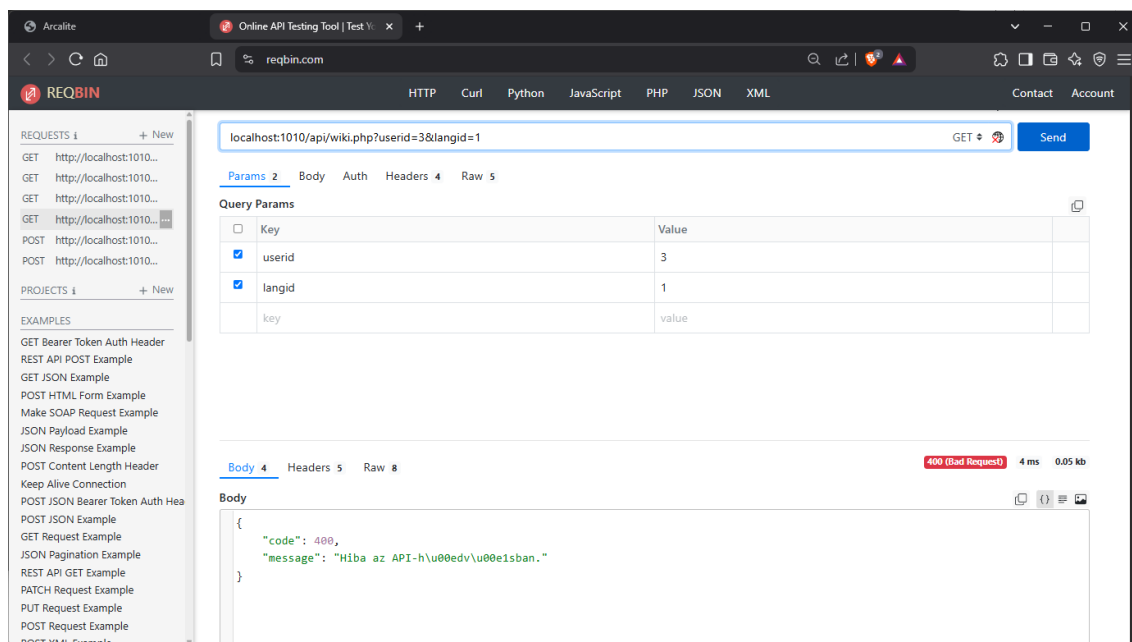
Ha a hívás metódusa megfelelő, viszont a bemenő adatokkal van valami probléma, akkor 400-as (Bad Request) kódot kell az API-nak visszaadnia, illetve a „Hiba az API-hívásban.” üzenetet. Az első ilyen tesztnél bemenet nélkül hívtuk meg az API-t:



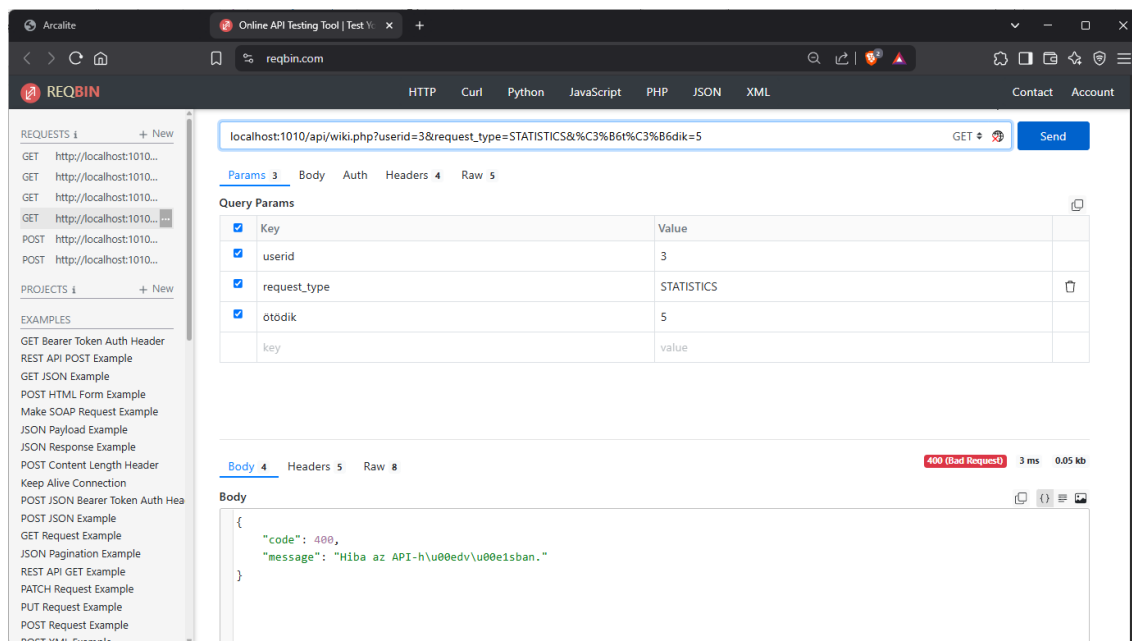
A második esetben több adatot adunk meg, mint amennyi elvárt. Itt látszik az is, hogy attól független is hibát ad vissza az API, hogy a kért adatok szerepelnek a bemenetben:



A harmadik esetben az elvártnál kevesebb adatot adunk meg:

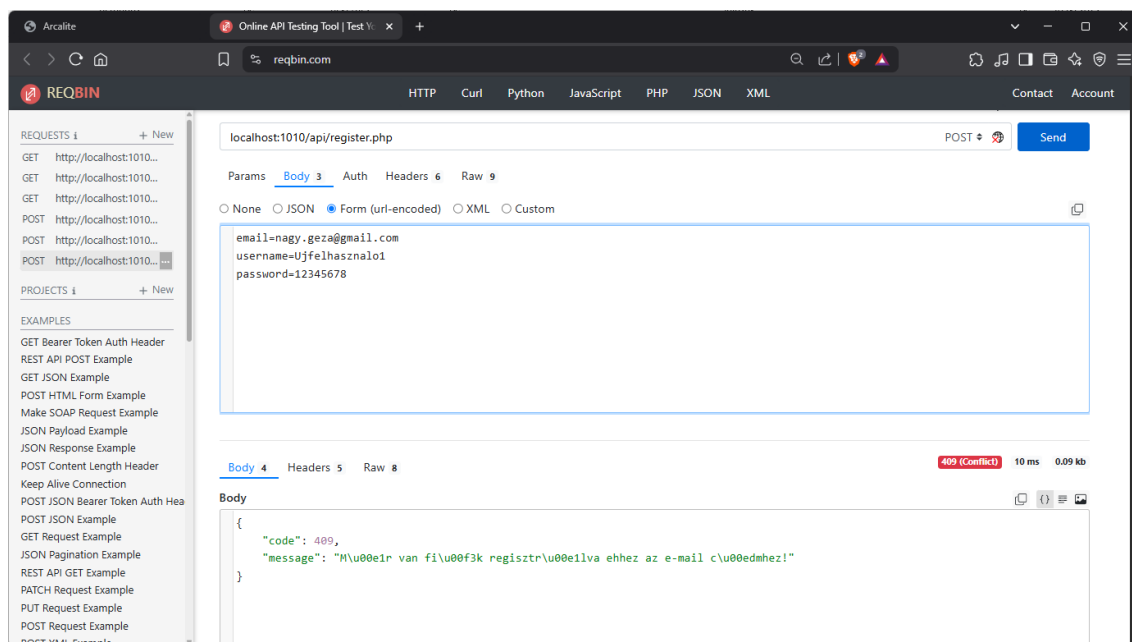


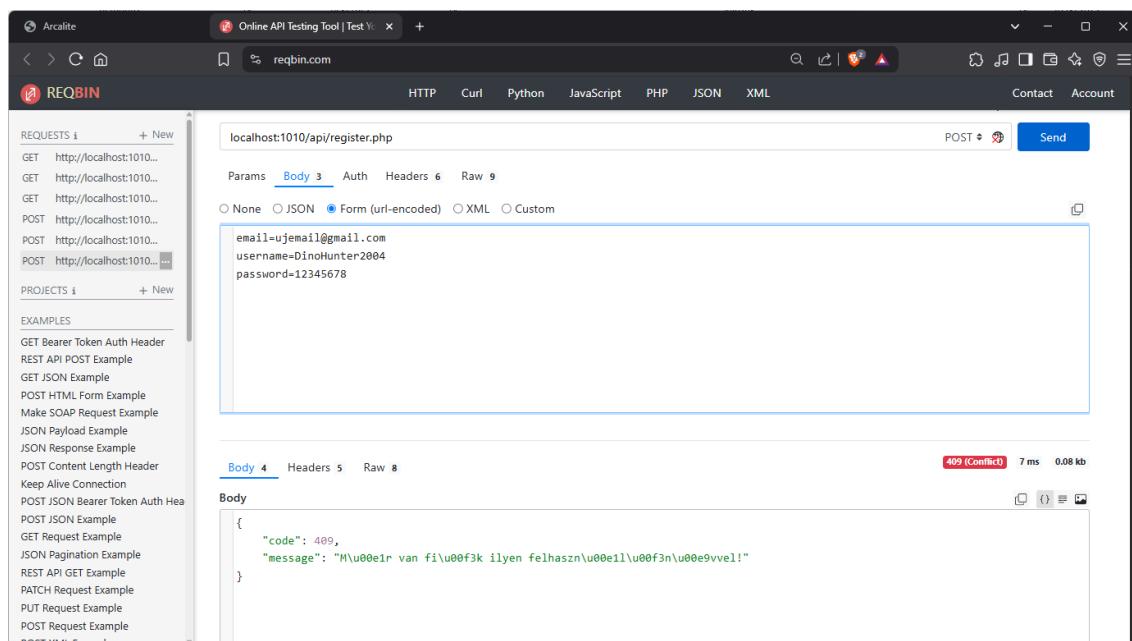
A negyedik teszt során megfelelő számú, viszont nem a hívás által elvárt adatokat adjuk meg:



2.7.1.3. 3. tesztet: helyes bemenet – egyéb hiba

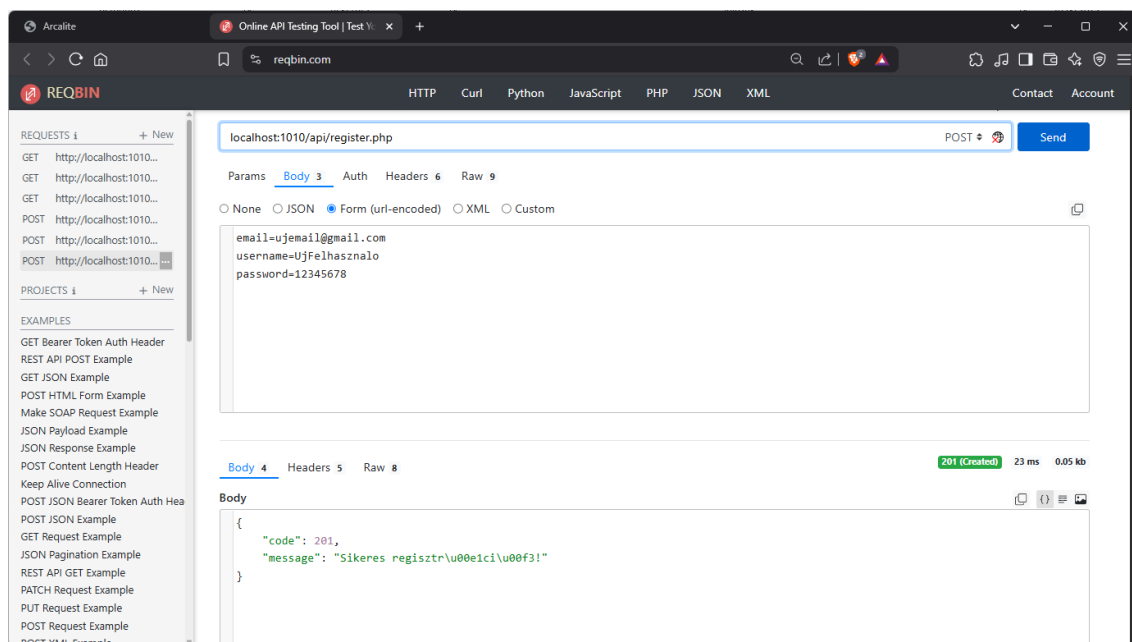
Ennél a tesztetnél az API-hívásunk helyes, ám valami egyéb hiba folytán hibaüzenetet kapunk vissza. Ilyen például regisztrációnál, ha már létezik a megadott felhasználónév vagy e-mail cím más (aktív) felhasználónál. Ilyenkor 409-es (Conflict) válaszkódot és a „Már van fiók regisztrálva ehhez az e-mail címhez!” vagy a „Már van fiók ilyen felhasználónévvel!” üzenetet kapjuk vissza.

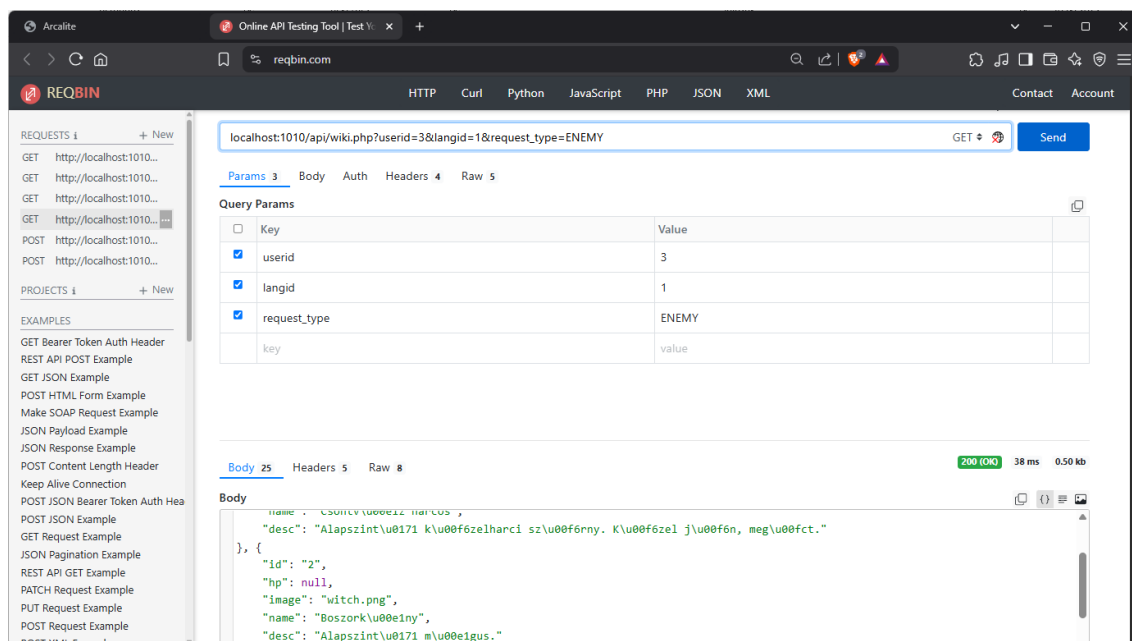




2.7.1.4. 4. tesztet: helyes bemenet és kimenet

Végül azt is ellenőrizzük, hogy megfelelő bemenettel helyes adat jön-e ki. Ilyenkor 200-as (OK) (regisztráció esetén 201 (Created)) válaszkódot kapunk, és a lekért adathalmazt, vagy egy sikeres lefuttatást visszaigazoló üzenetet.





2.7.2. Frontend tesztelés

2.7.2.1. Adatok megjelenítése

A weboldal fejlesztése során a leginkább kitesztelni való részek – nyilvánvalóan – azok, melyek változó adatokat tartalmaznak. Ilyen a lexikon, a ranglisták és a profil oldal. Itt alapvetően az adatok megjelenítését vizsgáljuk. Főleg a következő eseteket különböztetjük meg:

- ha nincs megjelenítendő adat
- ha az API hibát ad vissza
- ha az „átlagosnál” több adat van
- ha „átlagos mennyiségű” adatot kell megjeleníteni

Az „átlagos mennyiség” változó az oldalaknál, például a lexikonban – mivel ez csak a játékkal együtt bővül – csak általunk kiszámíthatóan sok adattal kell számolnunk. A ranglistáknál, mivel akárhány felhasználó regisztrálhat az oldalra, fontos a megfelelő skálázhatóság. A profil oldalon a felhasználó játékmeneteit soroljuk fel, és mivel elméletileg ez is bővíthet a végtelenségig, itt is fontos szerepet kellett kapjon a skálázhatóság. Viszont valószínűleg gondolkodva elmondható, hogy átlagosan több felhasználó lesz az oldalon, mint játékmenete egy felhasználónak, így a ranglistákat általánosan nagyobb adatmennyiségre kellett felkészítenünk, mint a profil oldalt.

2.7.2.2. Reszponzivitás

Az adatok megjelenítésében az is fontos szerepet játszik, hogy különféle méretű eszközről is megfelelően látszódjanak az adatok. Minden tesztesetnél nem csak a helyes betöltést kell vizsgálni, hanem a kisebb-nagyobb képernyőkön való megjelenést is.

2.7.3. A játék tesztelése

A program felépítését alapul véve, mivel minden, ami megjelenik manuálisan készül, így a tesztelés is manuálisan történt.

A következő funkciók lettek letesztelve:

- Minden szobában megjelenik az összes hozzárendelt elem (ellenségek, ellenőrzőpontok)
- A játékos nem tudja elhagyni a szobát, ha nem fejezte be (nem ölt meg minden ellenséget, nem használta az ellenőrzőpontot)
- A felhasználó tud új játékot kezdeni, vagy folytatni egy már meglévő végigjátszást
- Ha már létező végigjátszást indít a felhasználó, a legutóbbi ellenőrzőpontra doba be
- Az ellenőrzőponttól visszafelé nem jelennek meg az elemek amiket már használt a játékos
- A beállítások releváns funkciói működnek

2.8. Továbbfejlesztési lehetőségek

2.8.1. Játék

- Több pálya: Az új pályák hozzáadása az Arcalite egyik fő fejlesztési iránya. Több pálya hosszabb játékidőt, több változatosságot és új, eltérő témájú helyszíneket kínálna, amelyek új ellenségtípusok és történeti elemek bevezetését is lehetővé tennék.
- Vertikális pályák: A szobák közti függőleges mozgás hozzáadása teljesen új szintre emelné a játékelményt és a pályatervezés komplexitását.
- Több ellenség: Új ellenségtípusok és variánsok megjelenése várható a későbbi pályákon, melyek a játékos fejlődését és a nehézséget fokozzák.
- Több főellenség: Új főellenések közvetlenül kapcsolódnak a játék előrehaladásához – mind játékmeneti, mind narratív szempontból. Ezek a főellenések lehetőséget teremtenek új tárgyak és képességek bevezetésére is.
- Több játszható karakter: További játszható karakterek nagyobb testreszabhatóságot és választási szabadságot nyújtanának a játékosoknak.
- Mérföldkövek
- Osztályok: A játékos osztályok mélyebb játékmenetet biztosítanak, lehetővé téve a játéktílusok diverzifikálását és az újrajátszhatóság növelését.
- Történet: Egy háttértörténet vagy narratíva hozzáadása fontos szempont a játék jövőbeni fejlesztésében.

2.8.2. Weboldal

- Többnyelvűség: Az adatbázis fel van készítve többnyelvű kiírásra, a weboldalon egyelőre erre nincsen opció
- Biztonsági fejlesztések

3. Felhasználói dokumentáció

3.1. A program rövid ismertetése

Projektünk két fő részből áll: egy játék és egy hozzá kapcsolódó weboldal.

3.1.1. Játék

3.1.1.1. Bevezetés

Játék címe: Arcalite

Műfaj: Rouge-like

Rövid leírás: Egy 2D oldalsó nézetű játék, amelyben a játékos célja minden ellenfél legyőzése, folyamatos fejlődéssel minden harc után, egészen addig, amíg el nem éri a világ (a pálya) végét.

3.1.1.2. Technológia

- Engine: Godot 4.4 .NET – 2D tér
- Entitások: 2D sprite sheets
- Környezet: 2D Tileset – manuálisan készült pályák
- Felhasználói felület: 2D grafika

3.1.1.3. Rendszerkövetelmény

- Windows 10, 11

3.1.2. A weboldal

A weboldal három célt szolgál: egy bemutató és tudástár a játékhoz, néhány statisztika megjelenítése és profilkezelés.

Az oldalra alapból vendégként lépünk fel, viszont a menüben lehetőségünk van bejelentkezni. A bejelentkezés oldalon tudunk regisztrálni is. Bejelentkezés után megjelenik a *Profil* menüpont, melybe belépve láthatjuk külön játékmeneteink adatait, illetve a profilműveleteket végző gombokat. Itt lehetőségünk van nevet, e-mailt, jelszót változtatni, kijelentkezni és törölni a profilt.

A tudástár vendégként nem elérhető. Bejelentkezés után válik elérhetővé, viszont akkor is csak arról olvashatunk információkat, amikkel már találkoztunk.

A statisztika oldal elérhető vendégként is, viszont bejelentkezés után kiemelve látjuk a hozzánk tartozó adatokat.

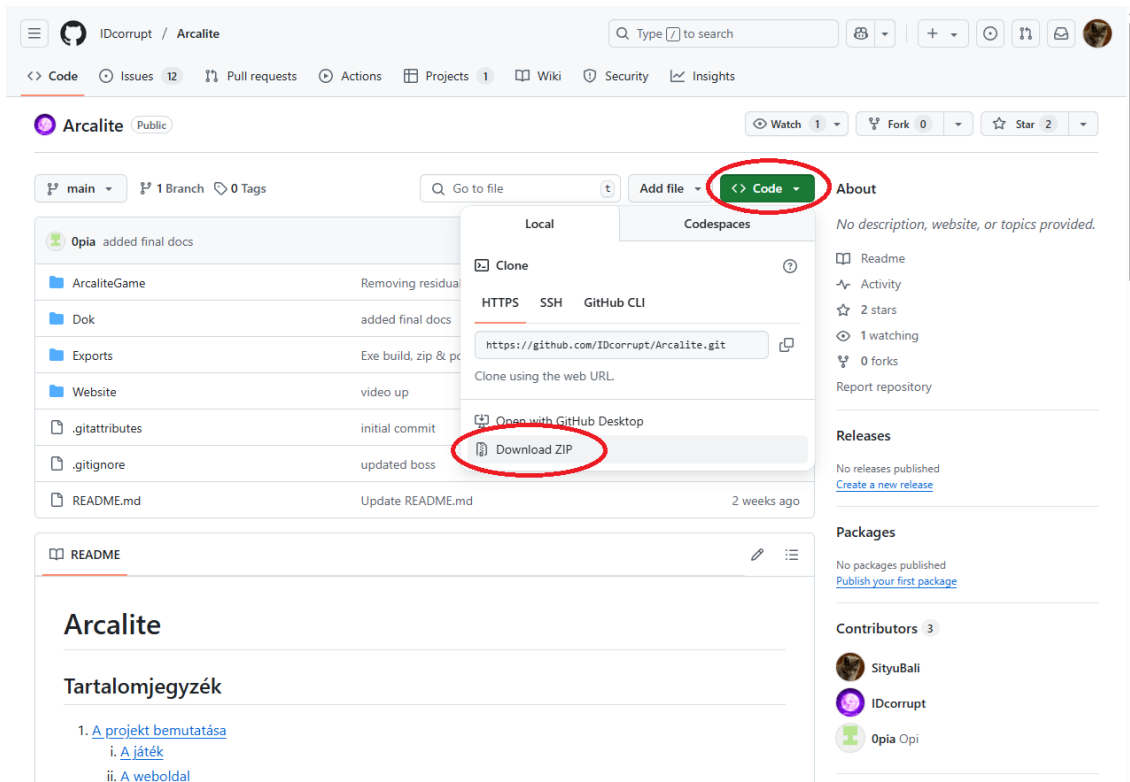
3.2. A program feltelepítése

3.2.1. Feltételek

- XAMPP szoftvercsomag

3.2.2. A projekt letöltése

A projektet a GitHub webes felületén a *Code* lenyíló menüre kattintva a *Download ZIP* opcióval tudjuk letölteni.



Ez egy tömörített (zip) fájlt fog letölteni, melyre a jobb egérgombbal kattintva feljön egy menü, melyben a „Kitömörítés” vagy „Kicsomagolás” szót tartalmazó menüponttal kitömöríthetjük a projektet. Így már egy egyszerűen elérhető mappában lesz a projekt.

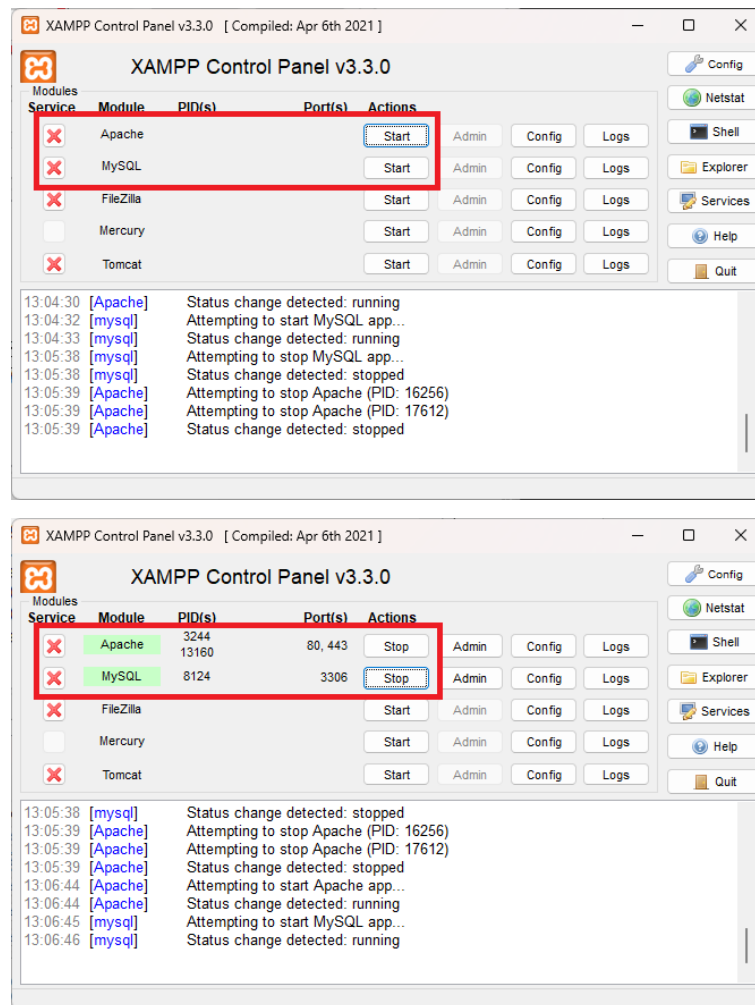
3.2.3. A játék telepítése

A játék már játszható és megnyitható állapotban található az Arcalite\Export\Exe mappában. Az arcalite.exe fájlal indítható el a játék.

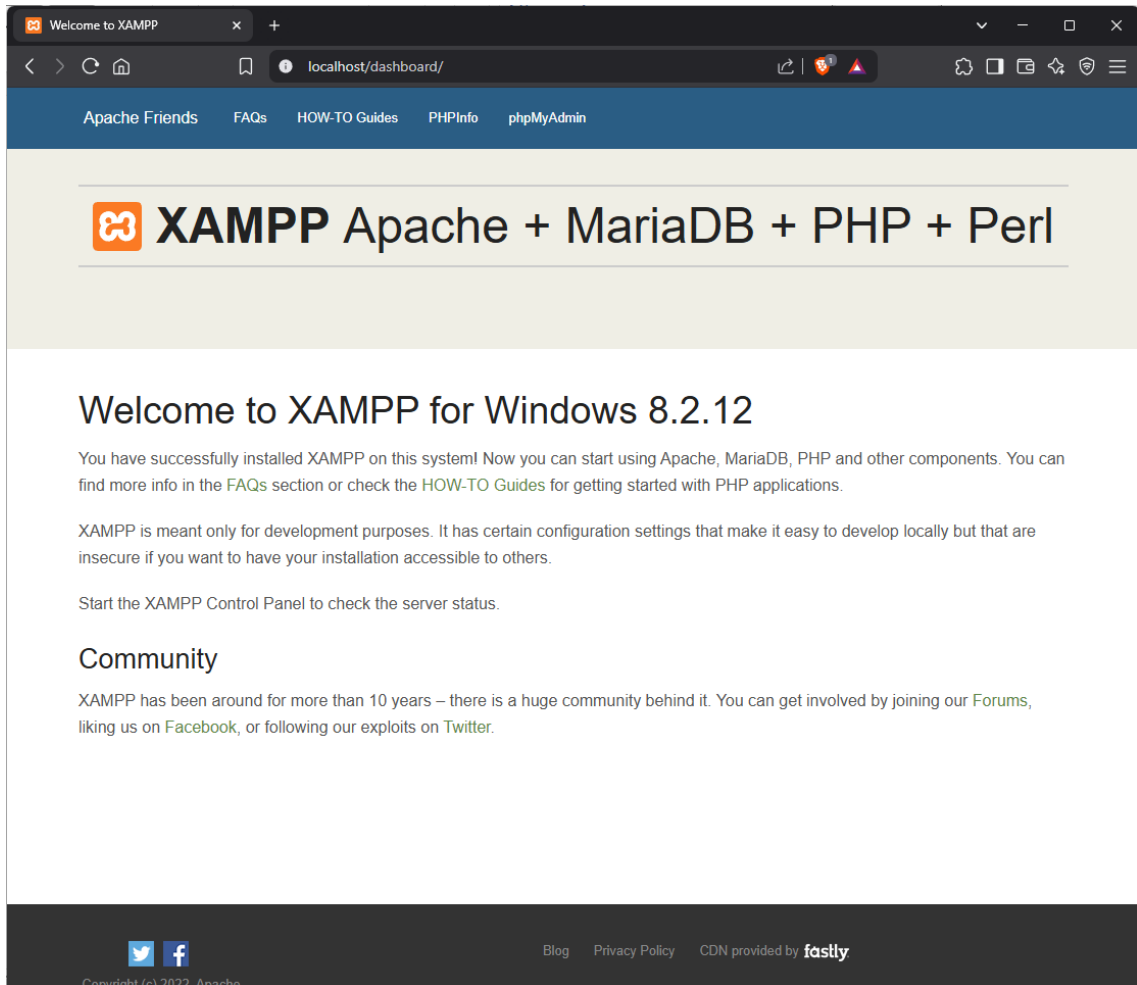
3.2.4. A weboldal és adatbázis üzembe helyezése

A weboldal és az adatbázis működtetését a XAMPP szoftvercsomaggal oldjuk meg. A szerver felállításának lépései a következők:

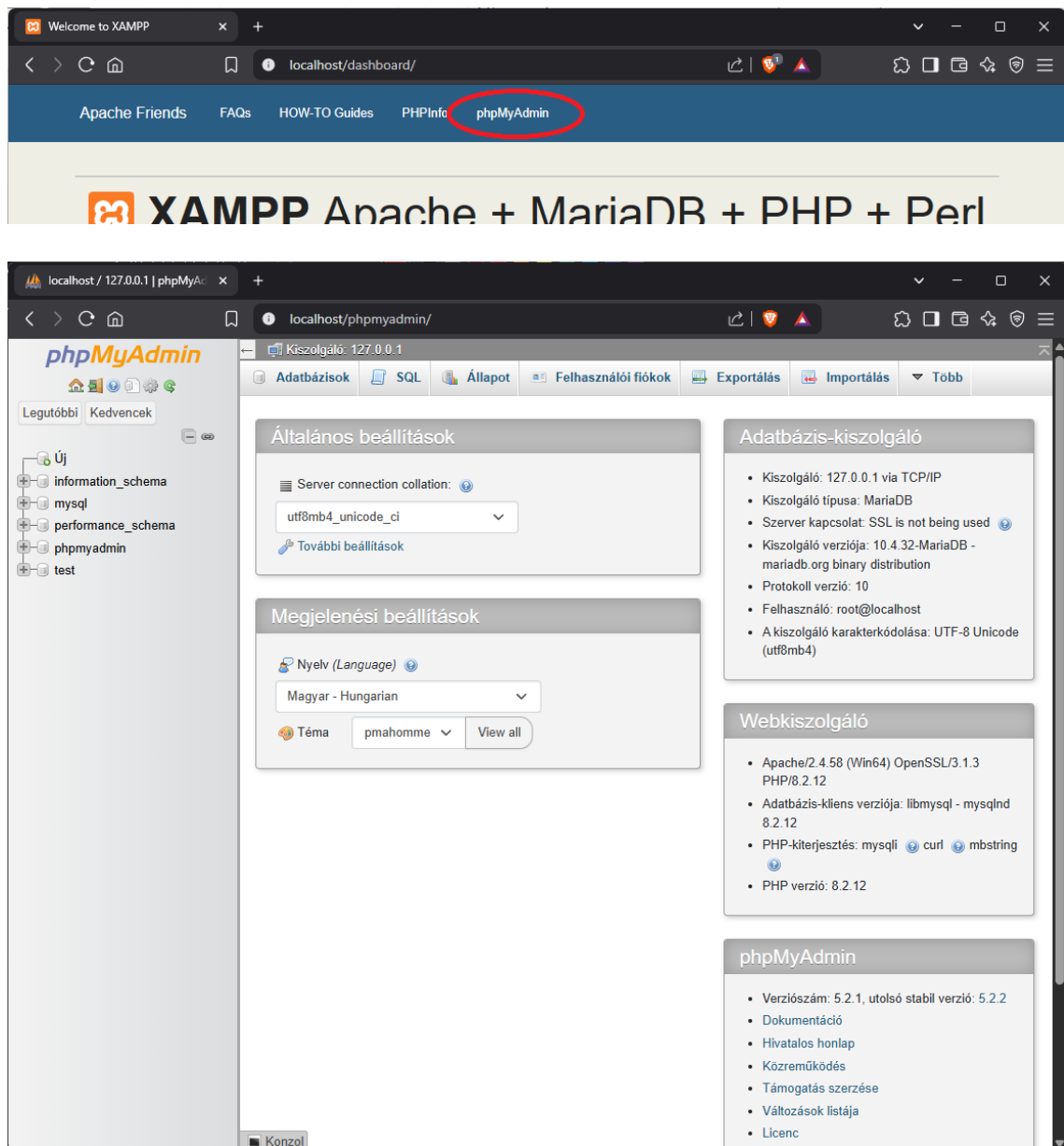
1. Megnyitjuk a **XAMPP** kezelőfelületet.
2. Elindítjuk az **Apache** és **MySQL** szolgáltatásokat.



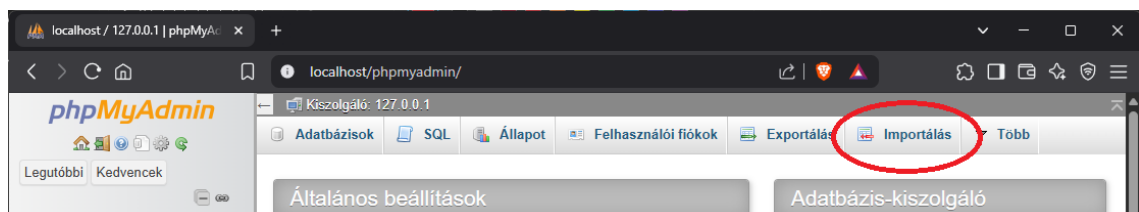
3. Megnyitjuk a XAMPP webes felületét. Ezt a böngészőnkben a **localhost** felkeresésével tehetjük meg. (A képen a `/dashboard/` automatikusan kerül a cím végére az oldal felkeresése után)



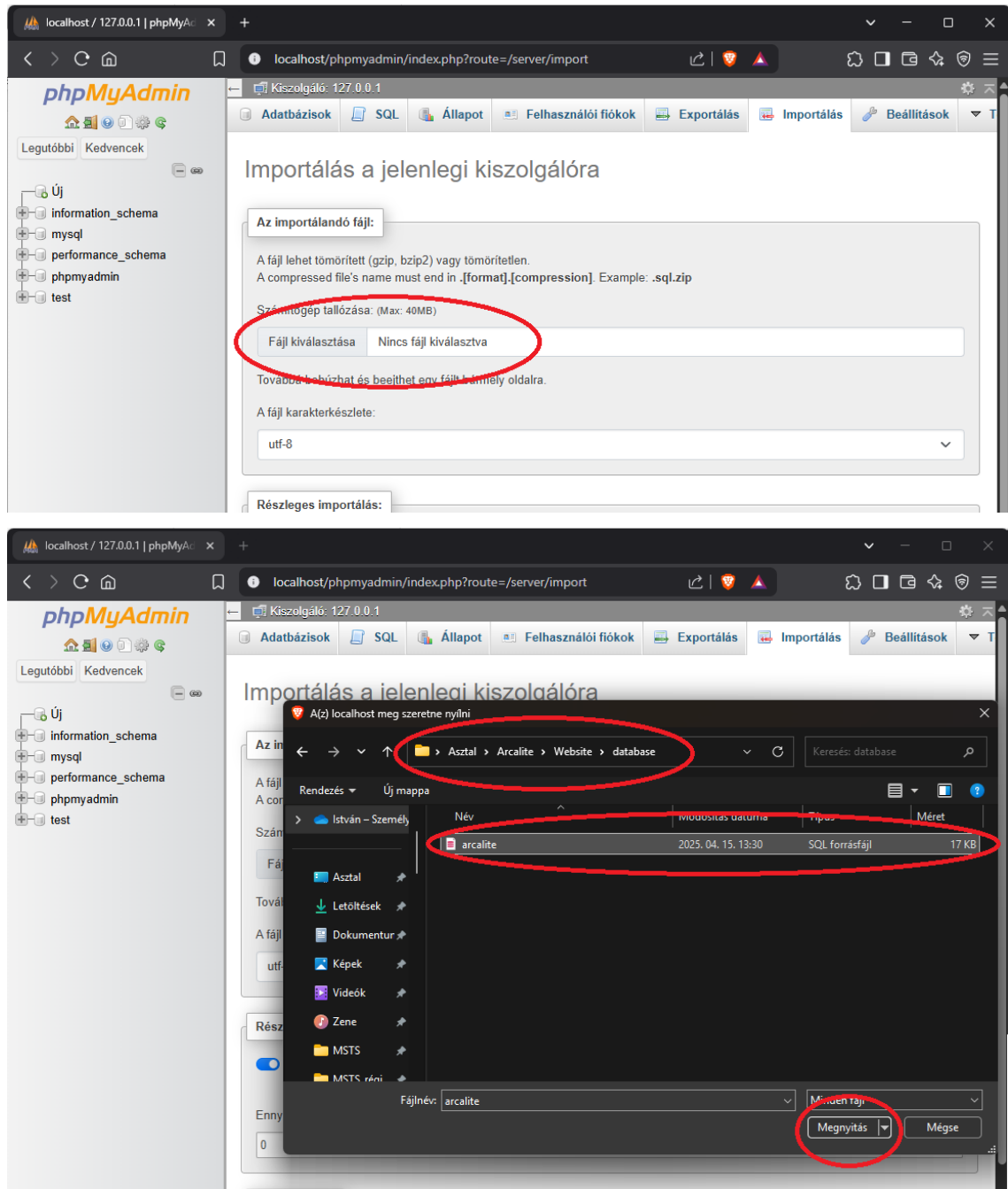
4. Megnyitjuk a **phpMyAdmin** felületet.



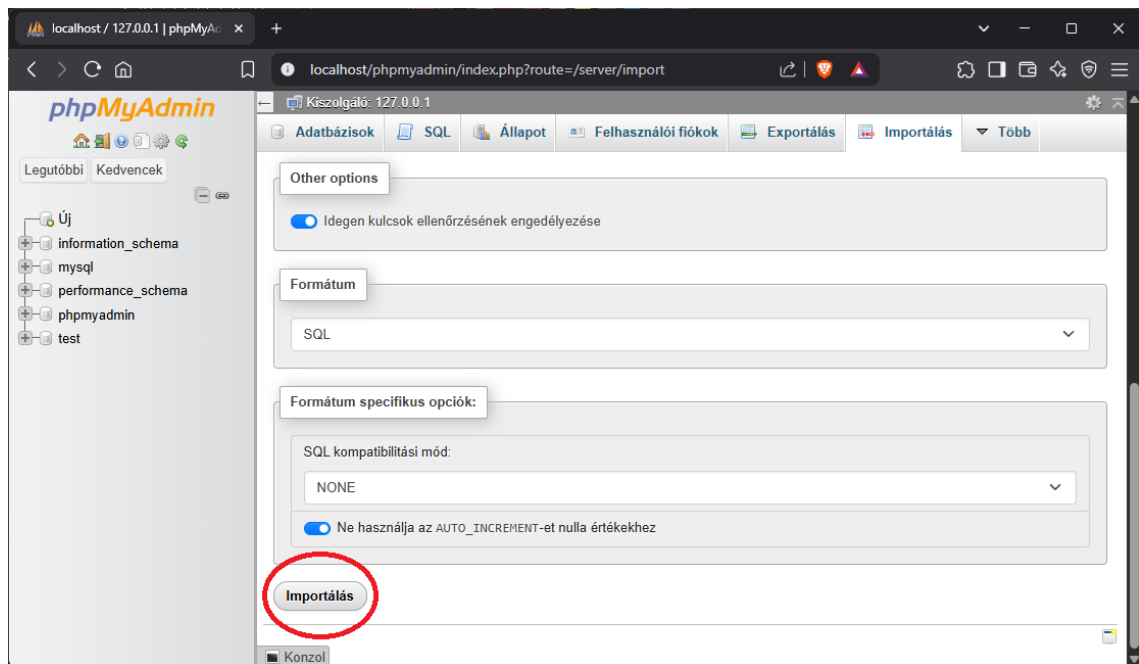
5. Megnyitjuk az **Importálás** oldalt.



6. A **Fájl** kiválasztása gombra kattintva kiválasztjuk a *Website\database* mappából az *arcalite.sql* fájlt.



7. Az oldal alján lévő **Importálás** gombbal beimportáljuk az adatbázist.



8. Megnyitjuk a **parancssort**.

9. Elnavigálunk a **Website** mappába. Ezt a parancssorban a „cd *útvonal*” parancssal tehetjük meg. Például, ha az asztalra csomagoltuk ki a projekt mappáját:

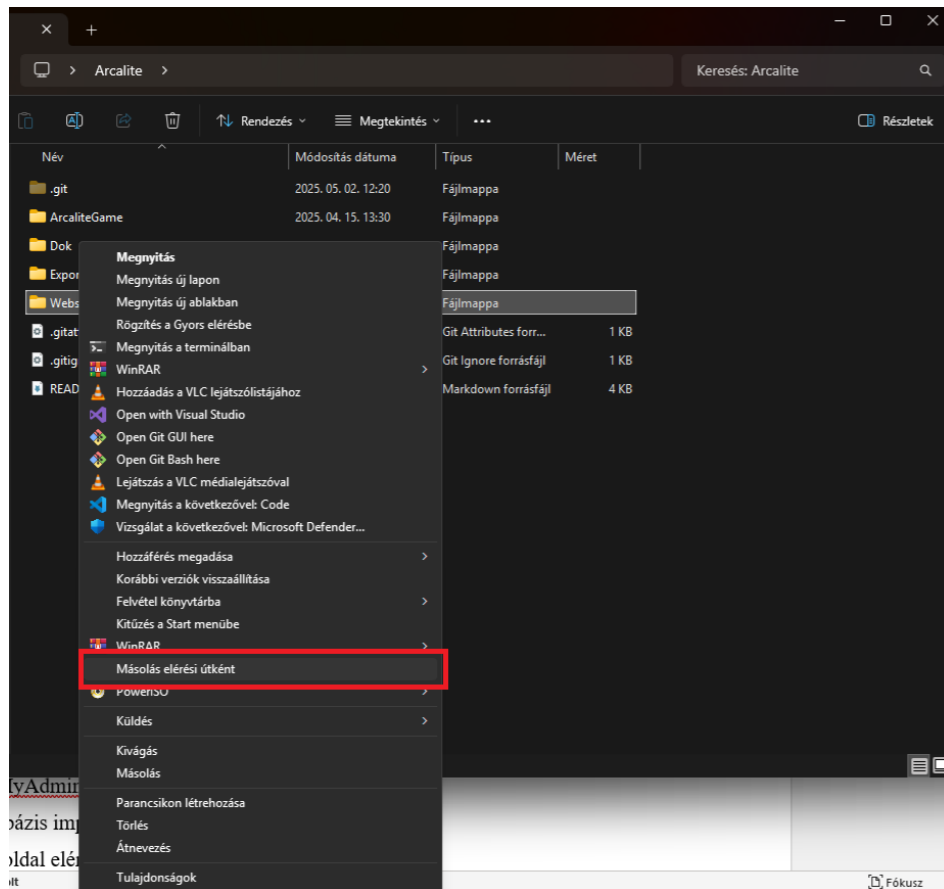
```

Parancssor
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. Minden jog fenntartva.

C:\Users\User>cd C:\Users\User\Desktop\Arcalite\Website

C:\Users\User\Desktop\Arcalite\Website>
  
```

- Ezt az útvonalat úgy is megkaphatjuk, hogy a fájlkezelőben megnyitjuk a projekt mappáját (*Arcalite*), majd a *Website* mappára jobb egérgombbal kattintva előhízott menüben a „**Másolás elérési útként**” menüpontot választjuk.



- Ezután a parancssorba a **jobb egérgomb** megnyomásával tudjuk beilleszteni.

```

Parancssor
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. Minden jog fenntartva.

C:\Users\User>cd "C:\Users\User\Desktop\Arcalite\Website"
  
```

10. Elindítjuk a szerveret a **php -S localhost:1234** paranccsal (ahol az 1234 egy általunk választott (szabad!) portszám) (az *S* betű nagy!)

```

Parancssor - php -S localhost
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. Minden jog fenntartva.

C:\Users\User>cd "C:\Users\User\Desktop\Arcalite\Website"

C:\Users\User\Desktop\Arcalite\Website>php -S localhost:1234
[Fri May 2 13:19:59 2025] PHP 8.2.12 Development Server (http://localhost:1234) started

```

Fontos, hogy ez a parancssor-ablak futtatja a szerveret, így, ha ezt **bezárjuk**, a szerver is **leáll!**

- Előfordulhat, hogy a rendszer nem ismeri fel a **php** parancsot. Ilyenkor a legegyszerűbb a XAMPP által telepített php programot használni. Ehhez a **php** elé a XAMPP saját mappáján belüli **php** mappa elérési útját kell megadnunk – alapértelmezetten **C:\xampp\php**. Így a hosszú parancs:

```

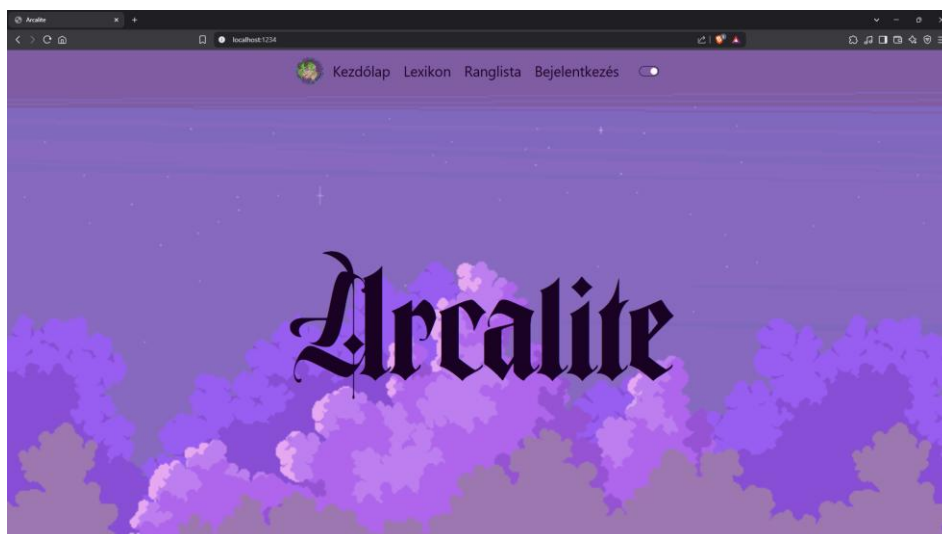
Parancssor - C:\xampp\php\p
Microsoft Windows [Version 10.0.26100.3915]
(c) Microsoft Corporation. Minden jog fenntartva.

C:\Users\User>cd "C:\Users\User\Desktop\Arcalite\Website"

C:\Users\User\Desktop\Arcalite\Website>C:\xampp\php\php -S localhost:1234
[Fri May 2 13:23:46 2025] PHP 8.2.12 Development Server (http://localhost:1234) started

```

11. A szerver indítása után a weboldalt a **megadott címen** érhetjük el. Az előző pont példája alapján a **localhost:1234** címen.



3.3. Használati útmutató

3.3.1. Játék és weboldal csatlakozása

A játékot megnyitva a program bedob a főmenübe. Ha a játékos szeretné a weboldalon követni a haladását, a bal felső sarokban található bejelentkezést kell használnia.

- Ha már van profilja
 1. A Sign ingombra kattintva megnyílik a bejelentkezés panel
 2. Már beregisztrált e-mail és ahhoz tartozó helyes jelszó beírása
 3. Sign in megnyomása ismét
 4. Ha helyesek az adatok, a program visszadobja a felhasználót a főmenübe, mostmár a bal felső sarokban a profilját megjelenítve. Ellenkező esetben a program jelzi a helytelen adatokat.
- Ha nincsen profilja
 1. A Register gombra kattintva a program eldob a weboldal regisztrációs oldalára.

3.3.2. A játék irányítása

3.3.2.1. Játék indítása

A játékot megnyitva a program bedob a főmenübe. Innen a játék menüjének a navigációja a következő:

- Start
 - o Continue: meglévő végigjátszást folytat
 - o New Game: új végigjátszást indít
- Settings: Beállítások ablakba dob
- Website: eldob a weboldalra
- Quit: kilép a játékból

3.3.2.2. Beállítások

A beállításokon belül minden panel változásai csak akkor maradnak meg, ha a felhasználó a Save gombbal elmenti őket. Ha a felhasználó vissza szeretné állítani alaphelyzetbe a beállításait, a Reset to default gombbal megteheti.

Game

Itt lehet állítani a játék nehézségét, illetve a sprint képesség típusát. A játék lehet könnyű (easy), átlagos nehézségű (normal), vagy nehéz (hard). A sprint iránya függhet a játékos jelenlegi irányától (8 direction), vagy a kurzor helyzetétől (follow mouse).

Video

Itt lehet állítani a játék ablakának a típusát (windowed, borderless, exclusive), felbontását és vsync-et lehet kapcsolni be vagy ki.

Audio

Itt lehet állítani a játék hangerejét. Ez jelenleg nem releváns, mert a játékban nem szerepelnek hangok.

Controls

Itt a felhasználó átállíthatja a játék irányítását magának. Az adott gombra nyomva a program figyelni kezdi, milyen billentyűt vagy egérgombot ad meg a felhasználó, majd azt újraköti.

3.3.2.3. Játékmenet

Egy új játékmeneten a játékos a pálya elején, a bal oldalán fog megjelenni. Balról jobbra tud haladni. A játék célja átjutni a pályán és minden szobát végigvinni.

Egy szobában lehetnek ellenségek, vagy lehet egy ellenőrzőpont.

Az ellenségek különböző fajtájúak lehetnek, és van rá esély, hogy legyőzésükkel erősítik a játékost.

A szobát nem lehet elhagyni amíg az ott található ellenségek nincsenek legyőzve, vagy az ellenőrzőpont nincs aktiválva.

Pár szoba után a játékos eljut a főellenséghez. A főellenség előtt mindig ellenőrzőpont szoba található. A főellenség legyőzésével a játékos megszerez egy tárgyat, amit onnantól használni tud az E vagy Q billentyűkkel (alapbeállítás szerint).

Halál esetén a játékos visszakerül a legutóbbi ellenőrzőponthoz.

Az ellenőrzőpontok töltenek vissza életpontot és manapontot.

3.3.2.4. Játék alapkoncepciója

Játék mechanikák

A játék alapvetően lineáris felépítésű: a játékos belép egy szobába, teljesíti az adott szoba feladatát, majd továbbhalad a következőbe. A feladatok közé tartozik az ellenfelek legyőzése, ellenőrzőpontok aktiválása és különböző környezeti kihívások teljesítése. Az ellenések dinamikusan tűnnek elő a játékos közelségétől függően, ezzel zökkenőmentes átmeneteket biztosítva a szobák között és csökkentve az erőforrás-felhasználást.

A játékos képességei

A játékos három fő adattal rendelkezik: életpontok, mana és sebzés, amelyek a játék előrehaladtával skálázódnak. Az életerő a játékos állapotát jelzi, a mana képességek aktiválásához szükséges, a sebzés pedig meghatározza a támadások hatékonyságát.

A mozgás gyors és dinamikus, a klasszikus 2D oldalnézetes (balra, jobbra, ugrás, guggolás) mozgás mellett egy gyors sprint képességgel is bővült, amely nagy sebességgel repíti előre a karaktert rövid ideig.

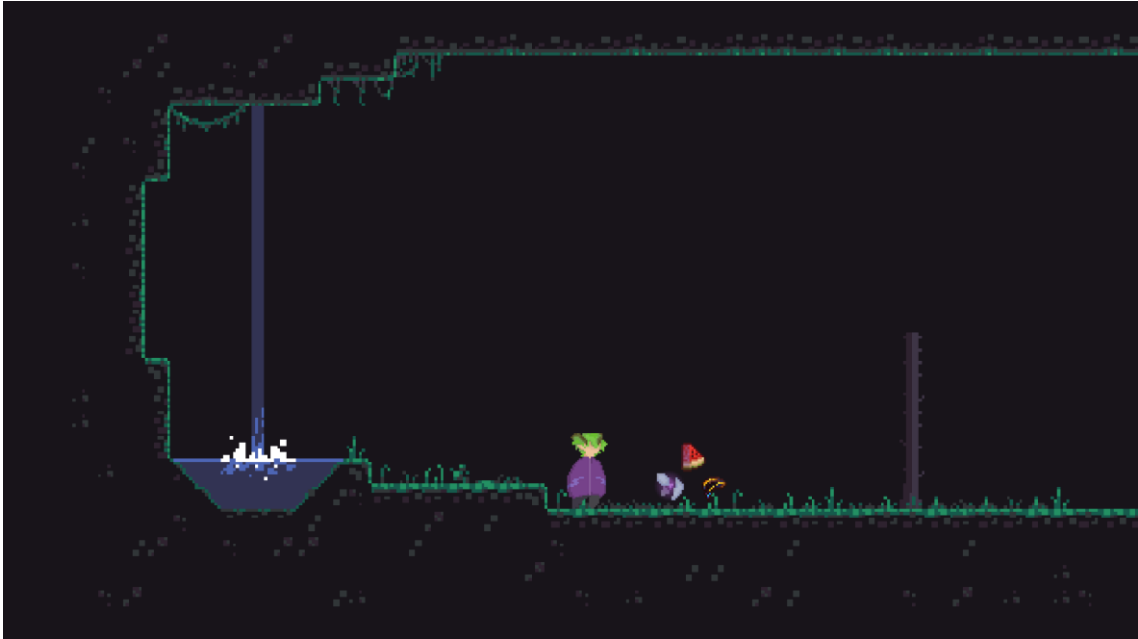
Két típusba sorolhatóak a képességek: támadás és varázslat.

A játékosnak két fajta támadása van: egy alap varázs lövés, és egy erősebb feltöltött lövés. Az alaptámadás folyamatosan használható, amíg a feltöltött felfüggeszt minden más tevékenységet amíg töltődik, minél tovább tölt annál erősebb és nagyobb less.

Jelenleg három varázslat van a játékban.

- Oracle – egy időt megállító kör
- Rapid-fire – egy ideiglenes felerősítés az alaptámadásnak
- Shield – egy ideiglenes teljes immunitást biztosító varázslat

Az Oracle az indulástól elérhető, míg a másik két varázslat speciális tárgyak megszerzésével szerezhető meg, amelyeket bizonyos ellenfelek vagy főellenfelek dobnak.



A játékos és a felvehető elemek

Ellenségek:

Az ellenségeknek 2 fő adatuk van: életerő és sebzés. Van egy speciális adatuk is: a pontdobási gyakoriságuk. Ez határozza meg, hogy mekkora eséllyel ejtenek el tárgyat halálukkor. Ez az érték konfigurálható úgy, hogy több mint egy tárgyat is dobjanak, akár százfeletti értékkel is. Az elejtett tárgyak segítségével a játékos fejlesztheti adatait. Különböző tárgyak (képesség tárgyak) 100%-os eséllyel esnek ki, de csak speciális ellenségekből.

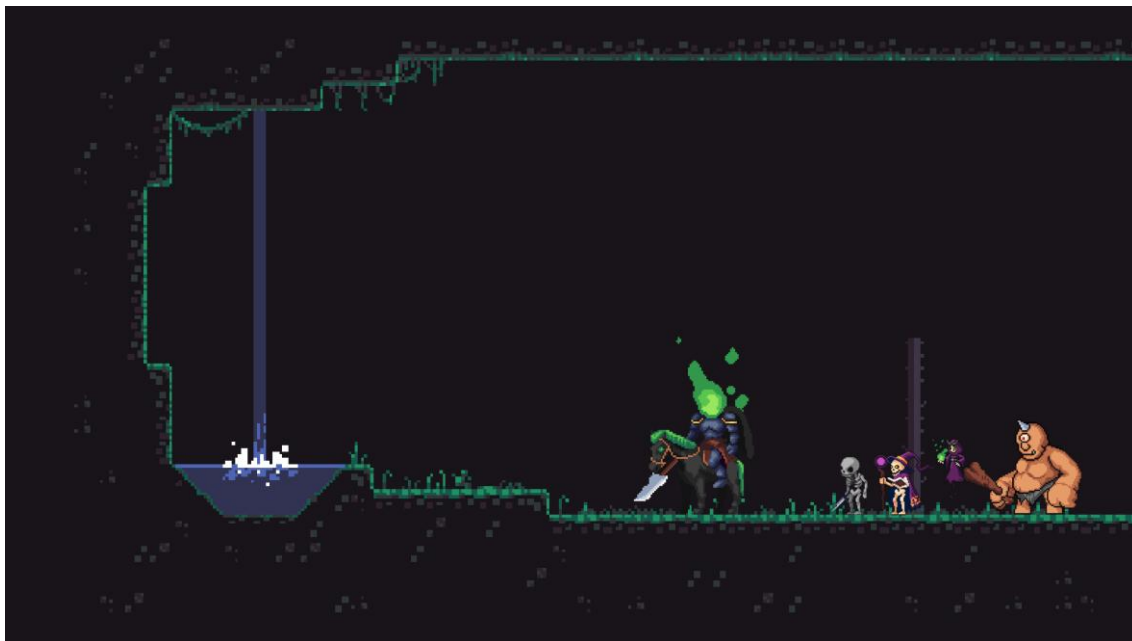
Az ellenségeket egy szobánkénti irányító rendszer jeleníti meg. Amikor a játékos belép egy szomszédos szobába, az irányító lehívja a szoba ellenségeit.

Megjelenés után az ellenségek nyugodt állapotban járőröznek véletlenszerű irányokba. Amikor a játékos belép a szobába, az irányító jelzést küld, és az ellenségek üldöző módba lépnek. Szimulált látómezejük van, tehát csak akkor üldözik a játékost, ha nem akadályozza semmi a látóvonalukat. Ha elvesztik a látást, egy ideig még a legutóbbi ismert pozíció felé mozognak, majd leállnak.

Összesen ötféle ellenség típus van, kétféle kategorizálással: távolság és osztály. Vannak közelharci és távolsági típusok, mindkettőből könnyű és nehéz változat. Végül van az ötödik típus: a változó "Elit" típus, amely régiótól függően teljesen eltérő dizájnnal és mechanikával rendelkezik.

Minden ellenség típusnak van egy támadási szekvenciája, az Elit típusnak pedig egy extra speciális támadása is.

Az ellenségek megszakíthatók erősebb támadásokkal, amelyek "flinch" effektust váltanak ki, kicsit hátralökve őket. Ez megszakít bármilyen aktuális támadást.



A jelenleg már implementált ellenfelek

Felhasználói felület

Két része van a felhasználói felületnek: adatok és képességek töltési ideje.

Az adatok a bal felső sarokban vannak elhelyezve. A karakter életpontjait egy sáv jeleníti meg, benne két számmal, amelyek a karakter maximális és aktuális életét jelzik. Az alatta lévő sáv a manapontokat jeleníti meg. Ez kisebb és nincs a pontos száma kiírva. Az utolsó a karakter sebzése, ami csak egy szám.

A jobb felső sarokban a töltési idők vannak elhelyezve.

Alapból három látható ikon van: sprint, feltöltött lövés és Oracle, amelyeket két tárgyképesség-hely bővíthet. Az ikonoknak 2 vagy 3 állapotuk lehet: kész, aktív vagy töltődés alatt. Az ikonok alapértelmezés szerint „Kész” állapotban vannak, és amikor aktiválsz a képességet/műveletet, az ikon „Aktív” vagy „Töltődés alatt” állapotba vált. Ha az adott képesség rendelkezik aktív időtartammal, akkor először „Aktív” állapotba kerül, majd azt követően vált „Töltődés alatt” állapotba. A „Töltődés alatt” állapotban lévő ikonok alatt egy töltősáv jelenik meg. Ez a töltősáv jelzi az adott képesség visszatöltési idejét, amely eltűnik, amikor a képesség újra „Kész” állapotba kerül.



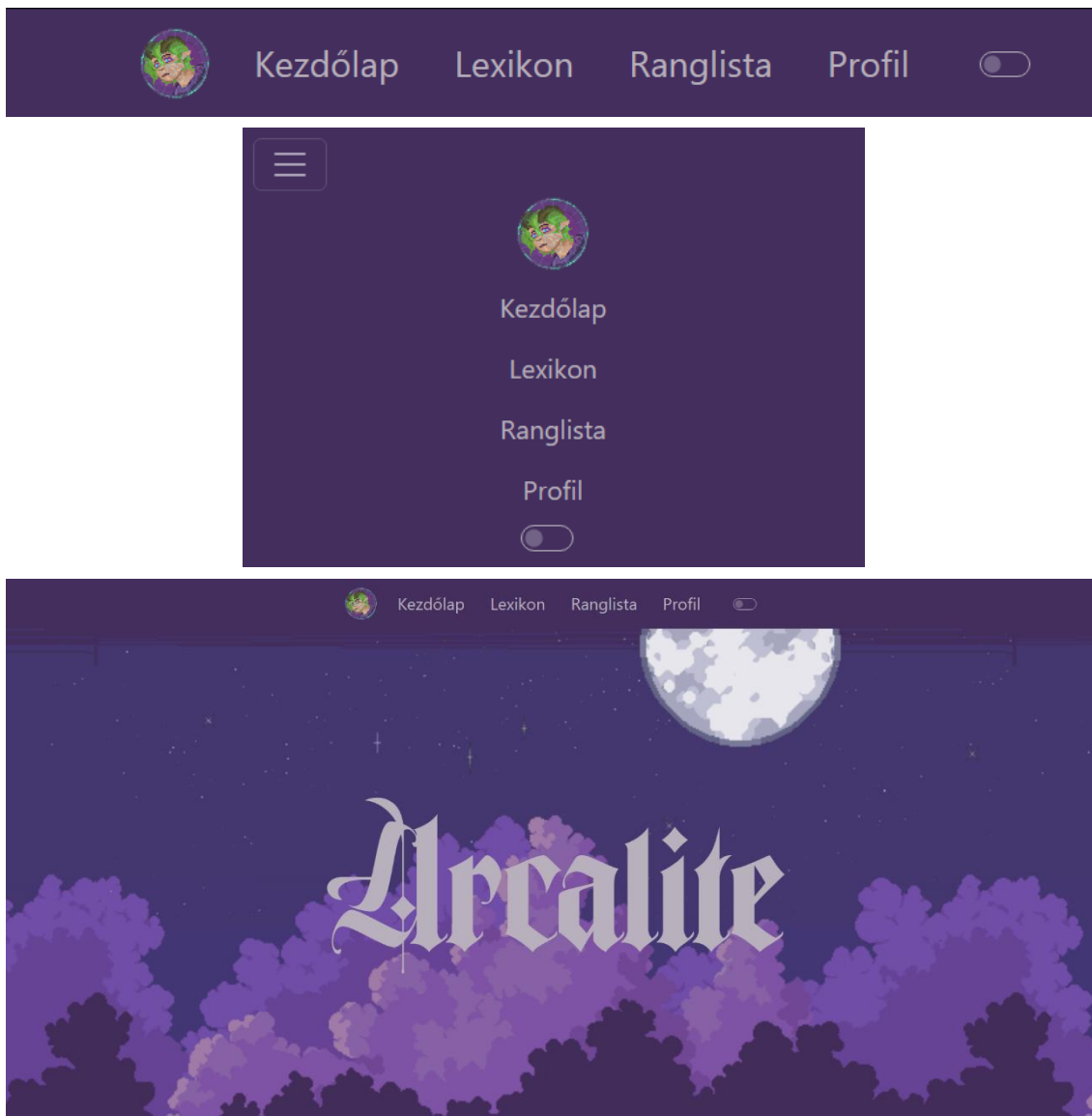
A felhasználói interfész (GUI)

3.3.3. A weboldal használata

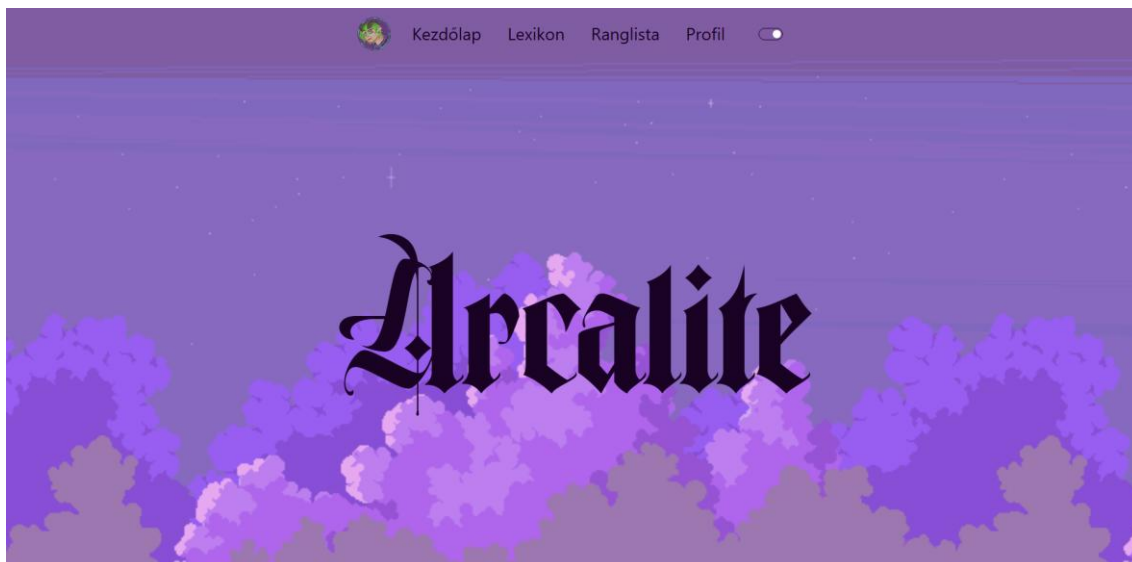
3.3.3.1. Menü

A weboldal tetején található menüsor az oldalak közötti navigálásra ad lehetőséget, és az oldal témájának változtatására. A jobb oldalon a menünek a kapcsolóval tud a felhasználó váltani sötét és világos nézet között.

Mobil nézeten egy hamburger menübe összeugrik, az ikonra kattintva lenyílik.



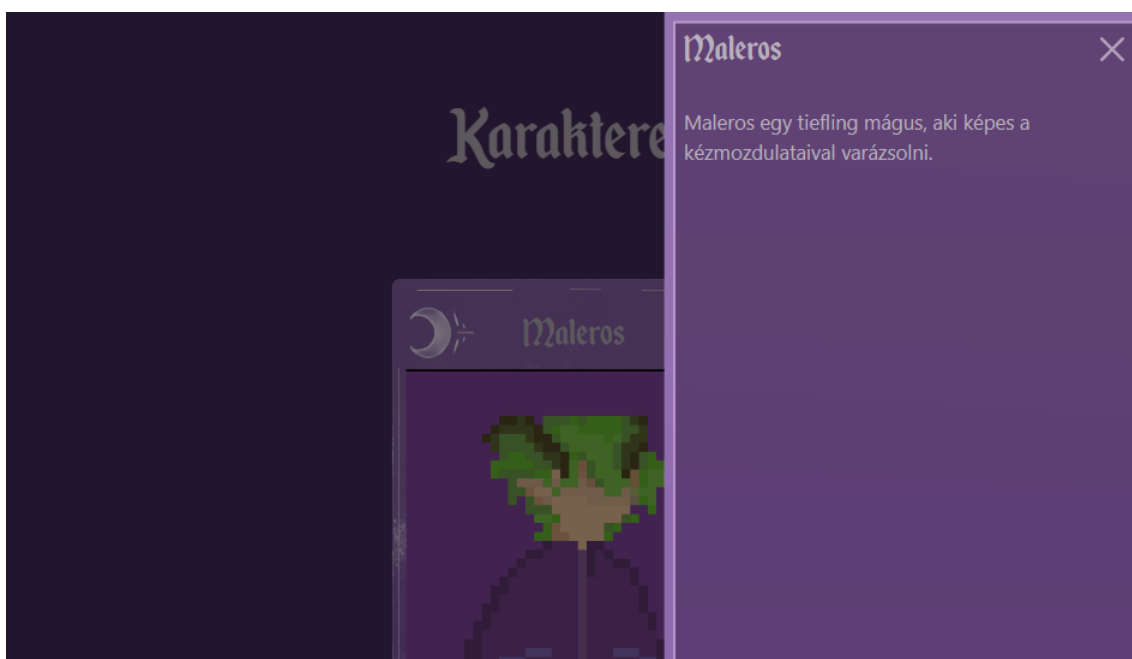
Sötét mód

*Világos mód*

3.3.3.2. *Index.html*

Ez a weboldal bemutató oldala. Itt található egy bemutató videó a játékról, az alap irányításai a játéknak és amit csinálnak, az elérhető karakterek, a készítő és elérhetőségek.

Valamelyik karakterre rákattintva feldob egy leírás oldalablakot az oldal.



3.3.3.3. *Lexikon.html*

Csak bejelentkezve elérhető oldal!

Ezen az oldalon megtalálható minden ellenség és tárgy, amivel már a felhasználó találkozott, és rögzítve lett a profiljába. Egy számláló mutatja az összes ellenségből és tárgyból mennyit talált már meg a felhasználó, és még mennyi van hátra. Egy ellenségre vagy tárgyra kattintva feldob egy leírás oldalablakot az oldal.

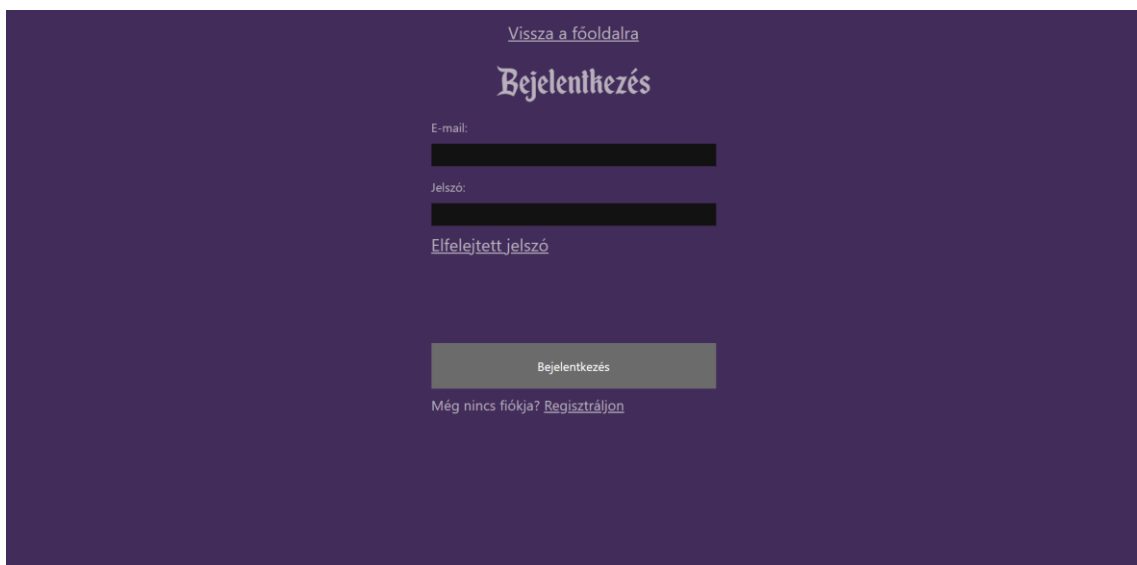
3.3.3.4. *Ranking.html*

Itt a regisztrált felhasználókról készült ranglista található. Lehet maga a felhasználót, vagy végigjátszásait megtekinteni a rangsorban. A különböző kategóriákra kattintva (pl. játékidő) a táblázat aszerint rendez.

Ha a felhasználó be van jelentkezve, a rá vonatkozó profil és végigjátszás ki lesz emelve.

3.3.3.5. *Login.html*

A bejelentkezés menüpont megjeleníti a bejelentkezés panelt. Itt ha a felhasználó rendelkezik profillal, az e-mail címét és jelszavát használva be tud jelentkezni.



Vissza a főoldalra

Bejelentkezés

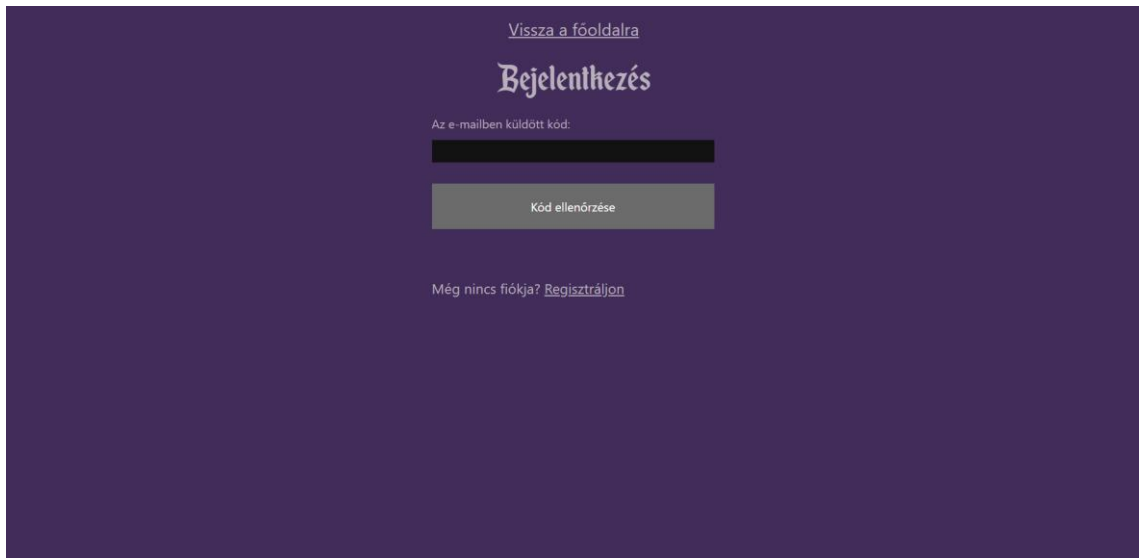
E-mail:

Jelszó:

[Elfelejtett jelszó](#)

Még nincs fiókja? [Regisztráljon](#)

Ha elfelejtette jelszavát, az Elfelejtett jelszó-ra kattintva e-mailben kap egy biztonsági kódot.



[Vissza a főoldalra](#)

Bejelentkezés

Az e-mailben küldött kód:

Kód ellenőrzése

Még nincs fiókja? [Regisztráljon](#)

Ha nincs még regisztrálva, a [Regisztráljon](#) linkre kattintva eldobja a regisztrációs oldalra.

Jelentkezzen be' (Already have an account? Log in)." data-bbox="163 380 880 629"/>

[Vissza a főoldalra](#)

Regisztráció

Felhasználónév:

E-mail:

Jelszó:

Jelszó újra:

Regisztrál

Már van fiókja? [Jelentkezzen be](#)

A regisztrációhoz szükséges adatok kiküldése után az oldal elküld egy biztonsági kódot a megadott e-mail címre. Az oldal bekéri ezt a kódot, ha sikeresen megadja a felhasználó, a profilja létrejön.

Sikeres bejelentkezés esetén a menüben található Bejelentkezés menüpont Profil menüponttá alakul.

3.3.3.6. Profile.html

A felhasználó profilját mutatja. Megjelenik a felhasználóneve és az eddig elmentett végigjátszásai, azoknak adatai.

A következő profilműveleteket tudja végrehajtani:

- Név megváltoztatása: felhasználónév megváltoztatása
- E-mail cím megváltoztatása
- Jelszó megváltoztatása
- Kijelentkezés
- Profil törlése

4. Források és linkek

A projekt GitHub repository-ja: <https://github.com/IDcorrupt/Arcalite>

A weboldal összes felhasznált eleme:

- Hátterek:
 - o <https://digitalmoons.itch.io/pixel-skies-demo>
- Ikonok: <https://fontawesome.com/icons/>
- Gombok:
 - <https://uiverse.io/dylanharriscameron/good-dingo-46>
 - <https://uiverse.io/ahmed150up2/stale-crab-79>

Az játék összes felhasznált asset linkje:

- Tileset: <https://octoshrimpy.itch.io/tranquil-tunnels>
- Ellenségek:
 - o <https://xzany.itch.io/headless-horseman-2d-pixel-art>
 - o <https://xzany.itch.io/cyclops-2d-pixel-art>
 - o <https://xzany.itch.io/witch-2d-pixel-art>
 - o <https://xzany.itch.io/skeleton-warrior-2d-pixel-art>
 - o <https://xzany.itch.io/skeleton-mage-2d-pixel-art>
- Lövedékek & Effektek: <https://bdragon1727.itch.io/>
- Lézer: <https://kanpelle.itch.io/ultimate-laser-pack>
- Cooldown ikonok: <https://free-game-assets.itch.io/free-skill-3232-icons-for-cyberpunk-game>
- Kurzorkészlet: www.kenney.nl
- Nyaklánc: <https://ssugmi.itch.io/16x16-rpg-assets>

5. Irodalomjegyzék

W3Schools: <https://www.w3schools.com/>

Godot Reddit: <https://www.reddit.com/r/godot/>

StackOverflow: <https://stackoverflow.com/questions/>

Flexbox Guide: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

ItchIO: <https://itch.io/>

DaFont: <https://www.dafont.com/>

FontAwesome: <https://docs.fontawesome.com/>

EmailJS: <https://www.emailjs.com/docs/tutorial/overview/>

Bootstrap: <https://getbootstrap.com/docs/>

Krita: <https://krita.org/en/features/>

Aseprite: <https://www.aseprite.org/docs/>