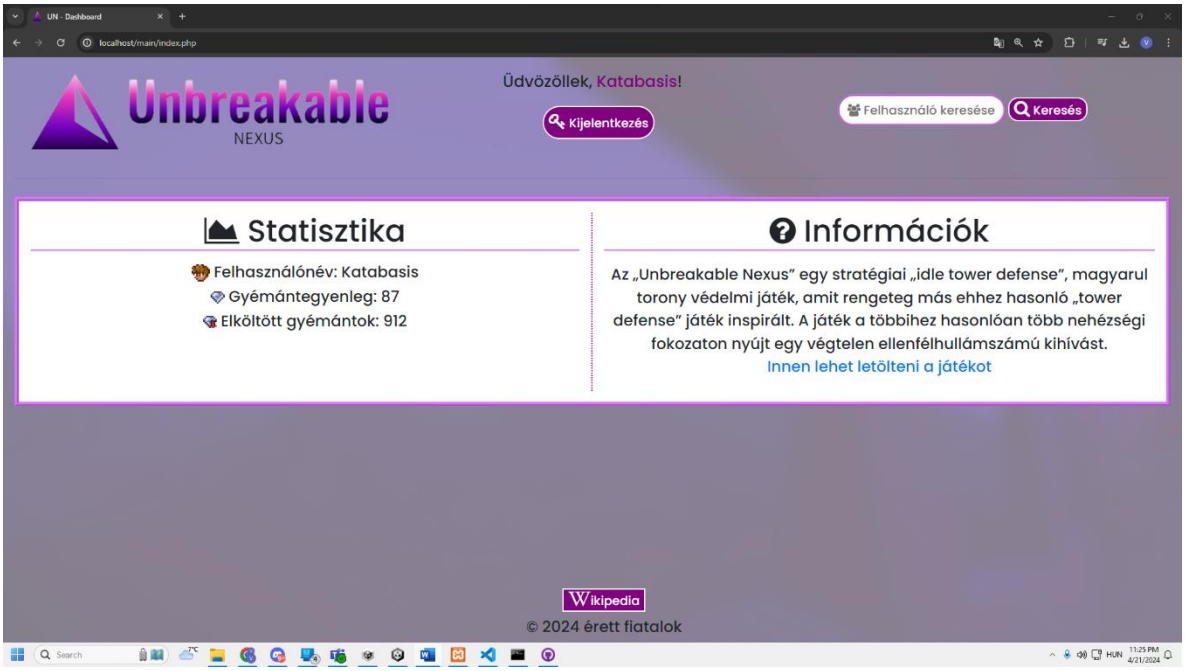

Készítette: Föőr Tamás Dániel, Molnár Elek Péter, Nyikon Nándor
Osztály: 2/14.2

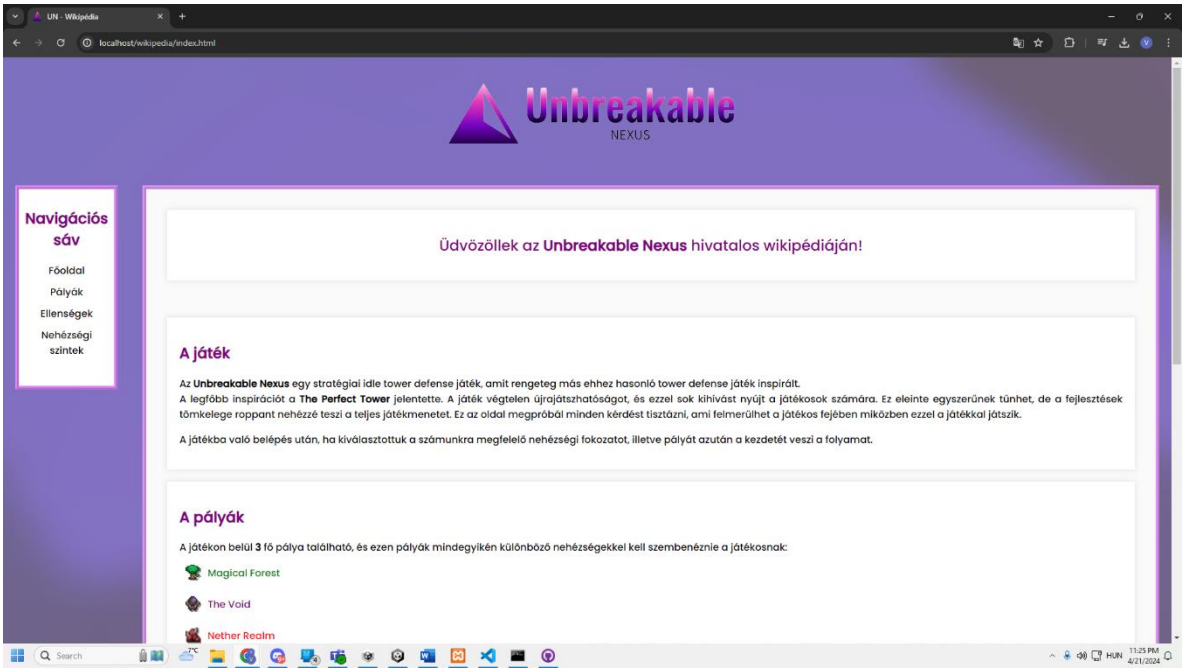
# Tartalomjegyzék

# Bevezetés

Az „Unbreakable Nexus” egy stratégiai „idle tower defense”, magyarul torony védelmi játék, amit rengeteg más ehhez hasonló „tower defense” játék inspirált. A játék a többihez hasonlóan több nehézségi fokozaton nyújt egy végtelen ellenfélhullámszámú kihívást. A játékban 3 féle ellenségtípus található, a sima, a gyors és a főellenség. Online funkcionalitással is rendelkezik a játék online mentések formájában, amelyet be lehet töltetni a játékban a „Save Management” menüpontban és a weboldalon is megjelenik bejelentkezés után. 3 féle pályan ölhetjük az alábbiakban említett ellenfeleket. A játék legnagyobb inspirációja a „The Perfect Tower” nevű játék volt, amelyet mind a hárman szabadidőnkben játszottunk, az inspiráció látszik a felhasználói felület dizájnján is.

A játékhoz tartozó weboldal 2 funkciót szolgál, az oldal felhasználói központként funkcionál, ahol a játékosok megtudják nézni a játékban összegyűjtött gyémántmennyiségüket és más felhasználóknak is megnézhetik ezt a mennyiséget. A másik funkciója egy útmutató, ún. tudásbázis a játékhoz, ahol a játékosok tájékozódhatnak a játékban megtalálható pályákról, mechanikákról és ellenfelekről.



## Használt technológiák

**Unity:** A Unity a videójáték fejlesztők között az egyik legismertebb és legtöbbet használt grafikai motor, amely 2D és 3D-s játékok futtatására lett készítve, de játékfejlesztésen kívül lehet használni 3D modellezésre is. A rugalmas szkript rendszere miatt volt a legjobb választásunk a vizsgaremekünk elkészítéséhez mivel szkripteket C# programozási nyelven kell megírni és ezt a nyelvet a tanáraink majdnem 5 éve tanítják nekünk.

**HTML, CSS és Bootstrap:** A HTML (magyarul Hiperszöveg Leíró nyelv) egy olyan webfejlesztési nyelv, amelyet a weboldal tartalmának elrendezésére használnak. A CSS (magyarul lépcsőzetes stíluslapok) egy olyan stílus leíró nyelv, amellyel az oldal tartalmának a stílusán változtathatunk pl.: színén, szegélyén stb.

**PHP:** A PHP (rekurzív mozaikszó, magyarul: Hiperszöveg Előfeldolgozó) egy olyan szerveroldali webfejlesztési nyelv, amely lehetővé teszi a dinamikus webes alkalmazások fejlesztését. A mi esetünkben lehetővé teszi az adatbázis szerver és kliens közti adatátvitel biztonságossá tételét vagy egyes esetekben magát az adatátvitelt teszi lehetővé.

**SQL:** Az SQL (magyarul: struktúrált lekérdező nyelv) egy speciális nyelv adatbázis kezelésre, amely lehetővé teszi az adatok lekérését, módosítását, de még az ezeket tároló táblákat és adatbázisokat is lehet ezzel módosítani. E-nélkül a nyelv nélkül a játék online funkcionalitása nem jöhetett volna létre mivel ezt használtuk felhasználói adatok mentésére.

# A vizsgaremek felépítése

## Felhasználói Felület

A Unity-n belül a felhasználói felületek úgynevezett „vásznak”, amelyekre lehet helyezni a különböző elemeket, mint például gombokat a szkriptekkel való interakcióhoz vagy beviteli mezőket az adatfelvitelhez. A játékmotoron belül már pár éve preferált a „TextMeshPro” szöveg megoldás a saját „Legacy Text Mesh” helyett mivel gazdag

szövegszerkesztési lehetőségeket biztosít. Ez alatt értjük azt, hogy lehet a szövegeket belül emotikonokat használni és jobb a szövegformázása az előző iterációhoz képest.

Az első jelenet, ahol a felhasználói felületet használja a játékos az a főmenü, itt 3 gomb található a játék indítására, a mentés menedzsmentre, és a játékból való kilépére. A mentés menedzsment menüpontban a felhasználó képes az összegyűjtött gyémántjait és megszerzett végleges fejlesztéseit elmenteni és feltölteni az SQL adatbázisban regisztráció után. A regisztrációs menüt a „Log In/Swap Accounts” gombbal lehet elérni, ahol a regisztrációs mezők kitöltése után és miután meggyőződtünk róla, hogy a jelszavunk biztosan az, mint amit első regisztrálhatjuk a fiókunkat. Ezután az előző menübe visszalépve az újonnan regisztrált fiókunkkal be tudunk jelentkezni. Ha esetleg visszatérő játékosok lennénk akkor a mentés menedzsmentbe visszalépve a „Load Save From Cloud” gombbal tudjuk a régebben mentett játékadatot betölteni, hogy folytassuk, ahol befejeztük. Minden játékmenet végén a „Save Data To Cloud” gombbal tudjuk az adatbázisba feltölteni az adatainkat. Visszatérve a főmenübe a játék indítás gombjával átkerülünk az akció választó menübe, ahol vagy a végleges fejlesztéseinket érjük el vagy tovább mehetünk a nehézségi fokozat választó menüre, ahonnan már csak ki kell választani a csataterünket.

A többi jelenetben csak kevés dolog tér elezért nem szánunk rá külön bekezdéseket, az egyetlen eltérés a felületben az a csatatéri gyengítést jelölő ikon. Itt viszont mind a 3 jelenetben megtalálhatóak a státusz jelölő kiírások, mint például a gyémánt mennyiségünk, érme mennyiségünk és a torony ún. „Nexus” életerő mennyisége. Ezen kívül itt találhatóak a játéksebesség állító gombok és az ideiglenes fejlesztésekhez vezető 3 gomb is.

## Szkriptek

Ebben a részben a játékot felépítő szkripteket szeretnénk bemutatni, amelyekkel bemutatjuk a játék működését. A játék elemeit 12 szkript köti össze a felhasználói felület gombjaitól az ellenfelek megidézéséig.

**WebHandler:** A WebHandler szkript felelős a játékban való bejeletkezésért és a regisztrációért. A lefutás kezdetekor meghívja a „LoadUserData” funkciót, ami a játékos helyileg elmentett adataiból előhívja a felhasználó egyéni azonosítóját és felhasználónevét azért, hogy a mentés menedzsment menüben kiirassa a képernyőre a jelenleg bejelentkezett

felhasználónak az adatait amennyiben a bejelentkezést megejtette a játékos. A bejelentkezést 2 funkció kezeli. A „LoginButton”, amely a bevitt jelszót és felhasználónevet átadja a nagyobb „Login” funkciónak. Ez a programrész előbb definiál egy változót, ami az API-t tartalmazó weboldalt hordozza magában, a mi esetünkben a localhoston található ingamelogin.php-nak fogjuk az adatok elküldeni feldolgozásra. Ez után egy var típusú változóban konkatenáljuk a bevitt adatokat majd ezt egy HTTP kliensen belül kódolt űrlappá alakítjuk, hogy a szervernek feldolgozhatóvá tegyük az adatot. Végül ezt egy aszinkron POST típusú kérést küldünk a szervernek, amely 2 féle adatot küldhet vissza. A szerver válaszát egy elágazáson kezeljük, ha a válaszüzenet 00-val kezdődik akkor mindenképpen hibakódot kaptunk vissza és azt helyén kezeljük. Ez a hibakód lehet 001, ami egy hibás jelszót jelöl vagy lehet 002, ami egy nem létező felhasználót jelöl, mind a 2 esetben egy informáljuk a felhasználót egy szövegdobozon keresztül, hogy a bevitt adata nem felel meg az eltárolt adatnak. Amennyiben nem kapunk vissza hibakódot a program elmenti egy fájlba, a Unity-nek a „PlayerPrefs” rendszerébe ezzel megspórolva a felhasználónak időt azzal, hogy nem kell a játékba bejelentkezni minden egyes alkalommal amikor a játék elindul. A mentés menedzsment menüben lévő szöveget frissítjük ezután és tudatjuk a felhasználóval egy szövegdobozon keresztül, hogy sikeresen belépett és visszadobjuk a főmenübe. Az imént említett ingamelogin.php felelős az adatok biztonságos átviteléért és az adatbázissal való kommunikációért. A felhasználótól megkapott adatot összehasonlítja az adatbázisban megtalált adattal a PHP-ban megtalálható „password_verify” funkció segítségével. A maradék 2 funkció a regisztrációért felelős, a „RegisterButton” funkció ugyan úgy átviszi a funkciót a nagyobb „Register” funkciónak viszont ez a funkció végzi a kettő bevitt jelszó megegyezésének átvizsgálását és a felhasználó tudatását abban az esetben, ha nem egyezik a 2 jelszó. A „Register” funkció szintén nagyban megegyezik a „Login” funkcióval, ugyanúgy kezeli a bevitt adatot, ugyan úgy a hibakód 00-val kezdődik, de itt a hozzákötött ingameregister.php kód nem ad vissza felhasználói azonosítót, csak 00-val kezdődő válaszkódot. A 001-es kód jelöli a sikeres regisztrációt és tudatja a felhasználóval, a 002-es kód, hogy a bevitt név már foglalt az adatbázisban, a 003-as hogy a szerverhez nem lehet csatlakozni, a 004-es ha az SQL parancs nem tud lefutni és a 005-ös ha a felhasználó nem vitt be adatot. A hozzá kapcsolódó PHP kód gondoskodik a felhasználói biztonságról mivel

az adatbázisban nem mentjük el a bevitt adatot egyből, hanem a PHP-ban beépített „password_hash” funkció titkosítja az eltárolás elött BCRYPT algoritmussal.

**SQLHandler:** Az „SQLHandler” szkript felelős a játékbeli végleges fejlesztések eltárolásáért az adatbázisban és onnan a kliensbe töltéséért abban az esetben, ha valaki visszatérő játékos lenne. 2 funkció szolgál az adatátvitelre, a „StatLoadSQL” és a „StatSaveSQL” de ahhoz, hogy ezek működhessenek az osztály elején a kapcsolódáshoz szükséges paramétereket definiáljuk, mint például az SQL szerver címe, a kiválasztott adatbázis stb. Aztán ezt egy karakterláncba sűrítjük, hogy a „MySql.Data” NuGet csomag tudjon csatlakozni a kívánt adatbázisunkhoz. A „StatSaveSQL” nevéhez hűen elmenti a játékos fejlesztéseit, gyémántjait és elköltött gyémántjait. A funkció elején lekérjük a felhasználó azonosítóját az „upgrades” táblában megnézve, hogy a felhasználó létezik-e. Amennyiben nem a funkció nem csinál semmit csak Debug.Log-al kiiratja hogy nem létezik a felhasználó, amennyiben viszont van lefuttat 5 UPDATE utasítást kicserélve az adatbázisban tárolt számokat és karakterláncokat. A másik funkció, a „StatLoadSQL” az adatok betöltéséért felelős ahogy a neve is árulkodik róla. Először 2 lekérést intéz az adatbázis szerver felé mind a kettőt a felhasználó azonosítójával kettő tábla felé, az „upgrades” és a „userstats” tábla felé, hogy megnézze létezik-e a felhasználó ezekben. Aztán egy elágazáson megnézi melyik táblából hiányzik a felhasználó, pl.: ha csak az „upgrades” táblából hiányzik a felhasználó akkor küld egy INSERT utasítást az alap adatokkal. A funkció minden esetben egy „UpgradeStore” nevű listához adja a különböző fejlesztés típusokat, elköltött gyémántokat és a meglévő gyémántokat is. Az „UpgradeStore” listát a végén tömbé alakítja, hogy egy másik szkript feltudja dolgozni.

**PermaUpgradeHandler:** Ez a szkript felelős a **végleges** fejlesztésekért, a fejlesztésekhez tartozó változóknak a karakterláncba konvertálásáért és a SQLHandler-nek átadásért. Összesen 47 változó tartozik az összes fejlesztéshez és a játékos gyémántjainak eltárolásához, minden fejlesztéshez tartozik 3 változó, a fejlesztés árához, a fejlesztett aspektus értékéhez (pl.: hány életerőnk van) és ahhoz, hogy egy adott fejlesztést hányszor végeztünk el. Minden fejlesztéshez tartozik egy funkció, amit a játékban rá lehet tenni egy gombra, a fejlesztések ára a fejlesztés erejétől függ, de általában ugyan azzal az egyenlettel vannak kiszámolva az értékek: $ár = alapérték * skálázódás^{fejlesztések\ száma}$ . Ezeken

kívül 3 funkció van, amely 45 értéket a 47-ből az előbb említett karakterláncokba sűríti fejlesztés típus szerint (offenzív, defenzív, nem csata befolyásoló) és ezt az „SQLUpgradeSaving” funkció adja át az „SQLHandler” szkriptnek mivel a Unity sajátos limitációja miatt, ha egy funkció bekér egy értéket akkor nem helyezhető gombra vagy játék objektumra és ez által nem lehet direkt használni. 1 funkció van ezen kívül, az „SQLUpgradeLoad”, ami az adatbázisból átveszi és szétválasztja a visszakapott adatokat. Az „SQLHandler”-től visszakapott „UpgradeStore” tömböt felbontja és „PlayerPrefs”-ben eltárolja lehetővé téve az online mentést.

**GameManager:** Ez a szkript van a játék szívénél mivel ez kezeli a játékban az **ideiglenes** fejlesztéseket, és a felhasználói felülettel kapcsolatos számátvitelt. A „PermaUpgradeHandler”-hoz képest ugyan úgy van 45 változó a fejlesztéseknek és ezen kívül még el van tárolva a játékmenet során szerzett ideiglenes fejlesztésekhez használt érme mennyiség és gyémánt mennyiség. A fejlesztések ugyan úgy működnek, mint a másik szkript mivel ebből lettek másolva. A szkript lefutása elején „PlayerPrefs”-ből a játékos végleges fejlesztéseit előhozzuk hogy ne csak dísznek legyen meg egy végleges fejlesztés a játékosnak és az elmentett gyémánt értékét is előhozzuk hogy ki tudjuk egészíteni. Az „UpdateUIText” funkció felelős a játék közben használt „vászonon” szereplő szöveg frissítéséért abban az esetben, ha a felhasználó fejleszt és a hozzá tartozó „TextFormat” és „TextConverter” felelősök a képernyőre kiírt számoknak a rövidítéséért, hogy a játékosnak ne kelljen pl.: 11 karakteres fejlesztés árat néznie. Ezen kívül ez a szkript kezeli a fejlesztés menüben található 3 gombot, amivel ki lehet választani egyszerre hányszor akarjuk a tornyunk statisztikáit fejleszteni, van lehetőség egyszer, ötször vagy akár a játékos érmemennyiségének a legtöbbet megengedő mennyiséget fejleszteni (Max fejlesztés). A szkript utolsó 3 funkciója átadja egy másik scriptnek a megszerzett gyémánt és érme értékeket.

**DiffHandler:** Ez a projekten belül az egyik legrövidebb szkript mivel egy nagyon egyszerű feladatot lát el, a játékon belüli nehézségi fokozatot állítja és adja át ezt az értéket. Minden nehézségi fokozatnak vagy egy saját funkciója, ahol definiálja a nehézségi szorzó értékét. A többi szkript ennek a segítségével tudja megváltoztatni az ellenfelek életerejét és támadási erejét nem mellesleg a játékosnak adott végleges fejlesztésre használt, gyémánt mennyiséget is.

**HealthBar:** Ez felelős az életerő csík mozgatásáért, amivel jelezzük a játékos felé mennyi életereje van.

**HoverOn:** Ennek a szkriptnek a segítségével tudtuk megoldani, hogy amikor a játékos egy UI elem felé húzza az egerét akkor több információt tudjon meg róla tehát hogy mit jelez az adott elem.

**MainMenu:** Eme szkripten keresztül tudunk tenni a főmenüben található gombokra funkciókat. Kifejezetten a csatatér választó menüben található 3 gombra referálunk mivel ezek nem olyan játékobjektumot változtatnak, amelyek egy „jeleneten" belül találhatóak meg hanem egy másik jelenetet töltenek be ezért kell hozzá egy vezérlő szkript.

**Pause:** Ez a szkript kezeli a játék megállítását, a menet újrakezdését és a 3 sebességkezelő gombot. Minden képkockán figyel a játék arra, hogy megnyomjuk-e az „escape" gombot, hogy ne csak a pause gombbal lehessen megállítani a játékmenetet. Tartalmazza ezen kívül még a „gameOver" funkciót, ami a nevéről árulkodva kezeli a játékmenet véget érését, megnyitva a hozzá tartozó képernyőt és a Unity-n belüli „timescale"-t nullára állítja teljesen megállítva a játékot ezzel. A „PauseM" funkció ugyan ezt a feladatot végzi el csak a hozzá tartozó képernyőn van egy „Resume" gomb, ami visszaállítja a játék sebességét és a UI-t is visszahozza. Ezen kívűl a 3 sebességkezelő gombra van 3 funckció ami a játék „timescale"-jét egyszeres, kétszeres vagy négyszeres sebességre állítja.

**spawn:** A „spawn" szkript tartalmazza az ellenfél megidézési logikát és számolja hány ellenfél van hátra a következő hullám megidézéséig. Ezen kívül kezeli a „Magical Forest" csatatérnek a gyengítési effektusát, a „Mystical Guard"-ot. A „waveSpawner" funckió intézi az ellenfelek megidézését ameddig az „_isWaveActive" változó igaz és a „_stopSpawning" változó hamis. Ezen kívül kezeli még az ellenfelek számát, ami 9-cel kezdődik az 1. hullámban, majd maximum 50-ig növeli az ellenfelek számát hullámonként és kezeli még az ellenfelek beidézési sebességét, ami pedig 2 másodpercenként idéz be ellenfeleket és ez 3%-kal csökken hullámonként a minimum 0,25 másodpercig. Az előbb említett „Mystical Guard" effektust pedig minden 10. hullámban applikálja az ellenfelekre, ha a jelenlegi csatatér a „Magical Forest".

**nexus:** Ez a szkript kezeli a toronyhoz, ún. „nexus”-hoz mind a rávonatkozó 45 statisztikát a támadási erejétől kezdve a támadás határsugaráig. Minden toronyhoz köthető funkció és mechanika itt van összegyűjtve mint például a kettő másik csatatér specifikus gyengítés effektus a „The Void”-hoz kötött „Void Corruption” és a „Nether Realm”-hez kötött „Purifying Flames” is. A szkript lefutásának a kezdeténél a játék megnézi a jelenlegi csatatér nevét azért hogy eldöntse melyik effektust kell beaktiválni, meghívja a „getNexusStats” funckiót amiben a „PlayerPrefs”-ből lekéri az elmentett végleges fejlesztéseket és egy ismétlődő meghívást intéz a „HPRegen” funkció felé 1 másodpercenként. Ebben a szkriptben a „VCStack” (Void Corruption) és a „PurifyingFlames” funkció intézi a csatatéri effektusokat vagy indít „Coroutine”-t (Unity-hez specifikus aszinkron funkció) és a hozzá tartozó vizuális jelző frissítő funkció is itt fut velük, a „VoidCorruptionIndicatorUpdater” és a „PurifyingFlamesIndicatorUpdater”. A lövéssel kapcsolatos mechanikák, mint az életerő lopás, az imént említett regeneráció, illetve a védelmi mechanizmusok, mint a százalékos védelem és az „abszolút védelem” itt van. 2 funkció van, ami magát a lövést kezeli, az egyik a „FindTarget” ami a Unity beépített „Raycast” (magyarul sugárvetés) rendszerével egy adott játékos számára nem látható gömbön belül a „nexus”-hoz legközelebb álló ellenfelet fogja célozni. A másik funkció, a „ShootAtTarget” az előzőből visszaadott „currentTarget” értéken lévő ellenfelet fogja sebezni. A „nexusHurt” funkció fogja azt kezelni, ha a „nexus” sebzést szenved el, a map effektusokat alkalmazza majd a védelmi statisztikák alapján elvégzi a sebzés kiszámolását és elveszi a „nexus” életerejét.

**enemy:** Minden ellenféllel kapcsolatos statisztikai számítás ebben a szkriptben történik, nem mellesleg az ellenfelekért kapott érmemennyiséget is ez intézi. A kód elején lefutó „SetEnemyStats” funkció beállítja az ellenfél statisztikát az alábbi egyenletek alapján:

$$\acute{e}leter\H{o}\ \acute{e}s\ t\acute{a}mad\acute{o}er\H{o} = kerek\acute{i}tve(alap\acute{e}rt\acute{e}k) * 1.4^{hull\acute{a}msz\acute{a}m} * nehézségi\ fokozat)$$

Az egyenlet akkor változhat, ha a „Magical Forest” csatatéren játszunk és ha a főellenség él mivel az megnöveli ezeket az értékeket 50%-kal. A szkript kezeli a vizuális elemek frissítését is intézi az „CheckBossIsAlive” funkcióban. Emellett a „Thorns” funkció intézi, ha a játékos vett ilyen fejlesztést, hogy az ellenfelek kapjanak sebzést attól függően, hogy hányszor fejlesztette ezt a játékos.

# Adatbázis

Az adatbázisnak egy nagyon egyszerű struktúrája nagyon egyserű mivel csak 3 táblát tartalmaz és mindent a felhasználó azonosítóján kötöttünk össze.

**users:** Ez a tábla nevéhez hűen a felhasználó azonosítóját, nevét és titkosított jelszavát tárolja.

**upgrades:** A tábla tartalmazza a felhasználó összes fejlesztését három oszlopra szeparálva a három fejlesztés típust (attackupgrades, defensiveupgrades, utilupgrades).

**userstats:** Tartalmazza a felhasználó által összegyüjtött gyémántokat és az elköltött gyémántok számát ahhoz hogy vissza lehessen adni ha a „Refund Upgrades” gombra rányomunk.

# Weboldal

A weboldalnak fontos szerepe van a projekten belül, mivel itt a felhasználók megnézhetik a játékban már korábban megszerzett, illetve eltárolt gyémántjaikat. A weboldalra belépve a felhasználó egy bejelentkezési felülettel találja szembe magát, ahol, ha a játékos rendelkezik regisztrált fiókkal, akkor be tud jelentkezni. A weboldalon HTML és CSS, illetve PHP környezet segítségével tudtuk kivitelezni mindazt, amit elterveztünk. A továbbiakban csak a fontosabb dolgokat és tudnivalókat fogjuk megemlíteni, mert ha minden kis elembe beleásnánk akkor az nem férne bele a keretekbe.

Ez a folyamat a „login.php” állományon belül zajlik le. A weboldal alapstruktúráját a „Bootstrap” használatával tudtuk létrehozni. Az oldal CSS részét nem részletezzük, a képeken minden látható lesz. Az egész oldal „<body>” tagjében, található egy „container” osztálynévvel, és „main” azonosítóval ellátott „ ” tag, ami igazság szerint a bejelentkezési felületet adja. Ezen belül található egy „Bootstrap”-es alaposztállyal, ami a belső margót állítja 4 pixel nagyságúra (p-4), illetve „login-form” azonosítóval ellátott űrlap, aminek a kitöltése után a weboldal küld egy úgynevezett POST típúsú kérést. Ezen az űrlapon belül található ismételten egy „Bootstrap”-es alaposztállyal, ami az alsó margót állítja 4 pixel nagyságúra (mb-4) ellátott kép ami 300 pixel szélességűre van állítva a „width” attribútum használatával. Ez a kép igazából a játékunk logója, amit a dokumentum

legelején is használtunk. Továbbá található kettő darab bemeneti mező, amelyek típusa a „type” attribútum igénybevételével lett meghatározva. Az első mező típusa „text”, ami egy alap szövegállományt jelent. Ez „username” névvel lett ellátva. A második mező típusa „password”, ami egy olyan szövegállományt hoz létre, amelybe bármilyen szöveget beírva automatikus rejtve lesz a jelszavunk. Ezáltal egyfajta biztonságot adva a felhasználónak, hogy nem jelenik meg nyíltan a képernyőn az általa használt jelszó. Ezt a mezőt „password” néven neveztük el. Az utolsó elem, ami igazából az egész bejelentkezési folyamatért felelős, az egy gomb, ami egy „loginGomb” azonosítóval lett ellátva. Emellett még kapott ez a gomb egy „submit” típust, amit szintén a „type” attribútum segítségével értünk el.

Amikor a felhasználó megnyomja ezt a gombot, lefut a háttérben az erre specializáltan megírt PHP-kód, ami felelős az egész bejelentkeztetésért. Itt látható a PHP-kód, amit a továbbiakban részletesebben kifejtünk:

```
if (isset($_POST["loginSubmit"])) {
  $felhasznalonev = $_POST["username"];
  $jelszo = $_POST["password"];
   require_once "database.php";
   $sql = "SELECT username,password,diamonds,spentdiamonds FROM users JOIN userstats ON users.id = userstats.id WHERE username = '$felhasznalonev'";
   $result = mysqli_query($conn, $sql);
   $user = mysqli_fetch_array($result, MYSQLI_ASSOC);
   if ($user) {
      if (password_verify($jelszo, $user["password"])) {
         session_start();
         $_SESSION["user"] = "yes";
$_SESSION["felhasznalo"] = $felhasznalonev;
```

```
            $_SESSION["dia"] = $user["diamonds"];
            $_SESSION["spentdia"] = $user["spentdiamonds"];
            header("Location: index.php");
            die();
        }else{
            echo '<div class="alert error-box">Hibás jelszó!</div>';
        }
    }else{
        echo '<div class="alert error-box">Hibás felhasználónév!</div>';
    }
}
```

A kód első sorában egy feltétellel megvizsgáljuk azt, hogy a „loginGomb” névvel ellátott gomb megnyomásra került-e. Ha a felhasználó megnyomta a bejelentkeztető gombot akkor létrejön egy POST típusú kérés. A kódon belül a felhasználónevet, illetve jelszót eltároljuk egy változóba ($felhasznalonev, $password), aminek a tartalma a POST kéréskor bejött két beviteli mezőbe a felhasználó által beírt felhasználónév, illetve jelszó párosítás. Ezek után a „require-once” kulcsszó használatával meghívjuk a „database.php” fájlt, amelyen belül a MySQL kapcsolatot teremtjük meg a weboldal, illetve az adatbázis között. Ezután egy SQL utasítás használatával kiszűrjük az adatbázisban található felhasználók közül azt a nevet, amit a felhasználó megadott. Ezután egy újabb feltételt állítunk, ami megvizsgálja azt, hogy a felhasználó létezik-e, és a feltétel teljesül, akkor következik a jelszóellenőrzés a „password-verify()” függvény segítségével, ami összehasonlítja a játékos által megadott jelszót, a korábban megadott felhasználónévhez társított jelszóval. Ha ez egyezést mutat akkor a „session_start()” segítségével indítunk egy munkafolyamatot, ami után a felhasználót automatikusan átirányítja a főoldalra („index.php”). Viszont, ha nem létezik a felhasználó, és nem teljesül a feltétel akkor figyelmezteti az oldal a felhasználót a bejelentkezési panel felett egy piros szövegdobozban

arról, hogy hibás a felhasználónév, vagy jelszó. A „login.php” oldal igazából csak ennyiért felelős.

A korábban említett „database.php”-t fogjuk most taglalni, amit azért hoztunk létre hogy a saját dolgunkat könnyítsük, és csak meghívni kelljen amikor szükségünk van egy MySQL kapcsolódásra. Ennek az oldalnak semmilyen frontend része nincsen, ez csak a háttérben dolgozik. Az oldal kódjában csak egy PHP-kód található, ami így néz ki:

```
$host = "localhost";

$user = "root";

$pw = "";

$dbNev = "nexusdb";

$conn = mysqli_connect($host, $user, $pw, $dbNev);

if (!$conn) {

    echo '<div class="alert error-box">SQL ERROR!</div>';

}
```

Itt eltároltuk az összes adatot, ami szükséges egy sikeres MySQL kapcsolat létrehozásához. Fontos kiemelni, hogy az itthoni tesztelések miatt lokális szervert használtunk. Kezdve a „$host” változóval, amiben a hosztnevet tároljuk, ez most jelen esetben „localhost”. Ezután a „$user” változóban deklaráltuk az adatbázishoz tartozó felhasználónevet, ami most ebben az esetben „root”. Ezek után, a „$pw” változóban elraktároztuk a jelszót, ebben az esetben nem határoztunk meg jelszót. Végül a „$dbNev” változóban található az adatbázis neve, amit mi „nexusdb”-nek neveztünk el. Mindezek után megpróbáljuk létrehozni a kapcsolatot a „mysqli_connect” függvény segítségével, ahol a korábban deklarált 4 változót használva tudjuk létrehozni a kapcsolatot. Itt egy feltételt állítunk fel, és amennyiben a kapcsolat sikertelen, akkor a weboldal ismételten figyelmeztet minket erről egy szövegdobozban, hogy az SQL kapcsolat meghiúsult valamilyen hibából kifolyólag. A „database.php” oldal igazából csak ennyiért felel.

Ha esetleg új felhasználó látogatja meg az oldalt, akkor a bejelentkezésnél található linkre kattintva, átirányítódik a regisztrációs felületre, amit a „register.php” kezel. Az oldal CSS részét nem részletezzük, a képeken mint a bejelentkezésnél, minden látható lesz. A felépítése kifejezetten hasonlít, a bejelentkeztető felülethez. A weboldal alapstruktúráját a „Bootstrap” használatával tudtuk létrehozni. Az egész oldal „<body>” tagjében, található egy „container” osztálynévvel, és „main” azonosítóval ellátott „ ” tag, ami igazság szerint a bejelentkezési felületet adja. Ezen belül található egy „Bootstrap”-es alaposztállyal, ami a belső margót állítja 4 pixel nagyságúra (p-4), illetve „register-form” azonosítóval ellátott űrlap, aminek a kitöltése után a weboldal küld egy úgynevezett POST típusú kérést. Ezen az űrlapon belül található ismételten egy „Bootstrap”-es alaposztállyal, ami az alsó margót állítja 4 pixel nagyságúra (mb-4) ellátott kép ami 300 pixel szélességűre van állítva a „width” attribútum használatával. Ez a kép igazából a játékunk logója, amit a dokumentum legelején, és a bejelentkezésnél is már használtunk. Továbbá található három darab bemeneti mező, amelyek típusa a „type” attribútum igénybevételével lett meghatározva. Az első mező típusa „text”, ami egy alap szövegállományt jelent. Ez „username” névvel lett ellátva. A második mező típusa „password”, ami egy olyan szövegállományt hoz létre, amelybe bármilyen szöveget beírva automatikus rejtve lesz a jelszavunk. Ezáltal egyfajta biztonságot adva a felhasználónak, hogy nem jelenik meg nyíltan a képernyőn az általa használt jelszó. Ezt a mezőt „password” néven neveztük el. A harmadik beviteli mező típusa szintén „password”, mivel itt a felhasználónak kötelezően meg kell erősítenie az általa korábban beírt jelszót, hogy az száz százalékosan megegyezzen. Erre a mezőre „repassword” néven hivatkozunk. Az utolsó elem, ami igazából az egész regisztrációs folyamat elindításáért felelős, az egy gomb, ami egy „registerGomb” azonosítóval lett ellátva. Emellett még kapott ez a gomb egy „submit” típust, amit szintén a „type” attribútum segítségével értünk el.

Amikor a felhasználó megnyomja ezt a gombot, lefut a háttérben az erre specializáltan megírt PHP-kód, ami az egész regisztrációs folyamat lebonyolításáért felel. Itt látható a PHP-kód, amit a továbbiakban részletesebben kifejtünk:

```
<?php
        if (isset($_POST["registerSubmit"])) {
```

```
$felhasznalonev = $_POST["username"];
$jelszo = $_POST["password"];
$jelszoUjra = $_POST["repassword"];

$passwordHash = password_hash($jelszo, PASSWORD_DEFAULT);

$hibaüzenet = array();

if (empty($felhasznalonev) OR empty($jelszo) OR empty($jelszoUjra)) {
 array_push($hibaüzenet,"Töltsd ki az összes mezőt!");
}
if (strlen($jelszo)<5) {
 array_push($hibaüzenet,"A jelszónak minimum 5 karaktert kell
tartalmaznia!");
}
if ($jelszo!==$jelszoUjra) {
 array_push($hibaüzenet,"A jelszavak nem egyeznek!");
}
if (!preg_match('/^[a-zA-Z0-9]+$/', $felhasznalonev)) {
 $hibasKar = preg_replace('/[a-zA-Z0-9]/', '', $felhasznalonev);
 array_push($hibaüzenet,"Helytelen karakter a felhasználónévben! (" .
$hibasKar . ")");
}
```

```
require_once "database.php";
$sql = "SELECT * FROM users WHERE username = '$felhasznalonev'";
$result = mysqli_query($conn, $sql);
$rowCount = mysqli_num_rows($result);
if ($rowCount>0) {
 array_push($hibaüzenet,"Ez a felhasználónév már regisztrálva van!");
}
if (count($hibaüzenet)>0) {
 foreach ($hibaüzenet as $hiba) {
    echo '<div class="alert error-box">'.$hiba.'</div>';
 }
}else{

 $sql = "INSERT INTO users (username, password) VALUES ( ?, ?)";
 $stmt = mysqli_stmt_init($conn);
 $prepareStmt = mysqli_stmt_prepare($stmt,$sql);
 if ($prepareStmt) {
    mysqli_stmt_bind_param($stmt,"ss",$felhasznalonev, $passwordHash);
    mysqli_stmt_execute($stmt);
    echo '<div class="alert valid-box">Sikeres regisztráció!</div>';
 }else{
    echo '<div class="alert error-box">SQL ERROR!</div>';
 }
```

```
    }
}
?>
```

A kód első sorában egy feltétellel megvizsgáljuk azt, hogy a „registerGomb” névvel ellátott gomb megnyomásra került-e. Ha a felhasználó megnyomta a regisztráció gombot akkor létrejön egy POST típusú kérés. A kódon belül a felhasználónevet, illetve jelszót (2x) eltároljuk külön változókba ($felhasznalonev, $password, $jelszoUjra), aminek a tartalma a POST kéréskor bejött három beviteli mezőbe a felhasználó által beírt felhasználónév, illetve jelszó párosítás. Ez a három mező nem lehet üres, ez a kódban ellenőrzésre kerül. Ezek után a „require-once” kulcsszó használatával meghívjuk ismételten a „database.php” fájlt, amelyen belül a MySQL kapcsolatot teremtjük meg a weboldal, illetve az adatbázis között. Ezután egy SQL utasítás használatával megvizsgáljuk, hogy az adatbázisban létezik-e már ilyen felhasználónévvel rendelkező ember. Ha már létező felhasználónevet ad meg a felhasználó, akkor figyelmezteti egy szövegdobozban az oldal arról, hogy az általa megadott név már foglalt. Vannak kritériumok a felhasználónév, és jelszó tekintetében mégpedig a különböző kritériumok közé tartozik az, hogy a jelszónak 5 karakternél többnek kell legyen, a megadott jelszavaknak feltétlenül egyezniük kell, és végsősoron a felhasználónév csak számokat, illetve betűket tartalmazhat. Ha ezeknek a feltételeknek nem felelnek meg a felhasználó által bevitt adatok, akkor ismételten egy szövegdobozban értesíti az oldal a felhasználót specializáltan arról a kritériumról, aminek nem felelt meg. Ha minden megfelel, és a felhasználó sem létezik az adatbázisban akkor egy SQL utasítás segítségével megtörténik az adatbázisba való behelyezés ahol a jelszó a „passwordHash()” függvény segítségével titkosításra kerül, ezzel növelve a felhasználók adatainak biztonságát, és erről végső soron egy szövegdobozban értesül a felhasználó. Ezek után visszamehet a bejelentkeztető oldalra, és bejelentkezhet a főfelületre.

A főfelületért az „index.php” a felelős, erre az oldalra irányítódik át a felhasználó a bejelentkezés után. Az weboldal alapstruktúráját a „Bootstrap” használatával hoztuk ismételten létre. Az oldal CSS részét nem részletezzük, a képeken itt is minden látható lesz.

Az oldal „<body>” tagjében, található egy „container-fluid” osztálynévvel, és „navbar” azonosítóval ellátott „ ” tag, ami 3 különböző „ ” -re van felosztva amelyek a „Bootstrap”-es „col-md-4” osztályjelölővel vannak ellátva.

Az első „ ”-ben található, egy kép ami ismételten a logónkra hivatkozik. A másodikban, egy üdvözlő üzenet található, ahol a PHP segítségével a „$_SESSION['felhasznalo']” kiíratásával személyre szabottan tudjuk köszönteni az oldalra belépett felhasználót. Ezalatt a szöveg alatt található egy gomb, amivel ki tudunk jelentkezni. Erre a gombra kattintva meghívódik a „kij()” függvény, ami átirányít minket a „logout.php”-ra, ami a kijelentkezésért felel. Ebbe a „logout.php” fájlban a „session_destroy()” függvény segítségével megszakítjuk az aktuális munkafolyamatot, ami a kijelentkezést jelenti. A harmadik „ ”-ben egy játékos kereső mechanizmust helyeztünk el, ezen belül egy űrlapot helyeztünk el aminek a kitöltésekor egy POST kérés hajtódik végre. Ezen belül található egy bemeneti mező, ahova a keresett játékos nevét tudja beírni a felhasználó, és ezt „kereses” azonosítóval láttuk el. Alatta található egy „searchGomb” azonosítóval és „submit” típussal megáldott gomb, aminek a megnyomásakor megtörténik a keresés amit PHP segítségével tudtunk kivitelezni, ezt majd a későbbiekben kifejtjük.

Továbbá az oldalon található még egy „container-fluid” osztálynévvel, és „panel” azonosítóval ellátott „ ” tag, ami egy összesítő táblázat formájában tárul a felhasználó elé. Ez szintén fel van osztva, csak 2 további „ ” -re, amelyek a a „Bootstrap”-es „col-lg-6” osztályjelölővel vannak ellátva.

Az első „col-lg-6” osztályjelölővel ellátott „ ”-ben a felhasználó játékbeli statisztikái vannak felsorolva, pontosabban a felhasználónév, gyémántok, és a játékban elköltött gyémántok, ezeket az adatokat PHP munkafolyamat segítségével tudtuk kiszedni az adatbázisból. A keresés végrehajtásakor, ugyanezen a felületen jelennek meg a keresett felhasználó statisztikái.

A második „col-lg-6” osztályjelölővel ellátott „ ”-ben a játékról találhatóak különféle információk, ez igazából csak designelem, semmi fontosabb jelentőséggel nem rendelkezik.

Az oldal „footer”, vagyis láb részében található egy „wikiGomb” azonosítóval ellátott gomb, ami átirányít az oldal Wikipédiájára, ami csak frontend nyelveket tartalmaz (pl. HTML, CSS), ezáltal nem tartalmaz semmilyen szerveroldali kommunikációt, szóval így arról nem szeretnénk méltóbb bemutatót tartani.

Most rátérnénk a felhasználónév alapján történő keresésnek, amit már korábban említettem a PHP kódjára, amit most fogunk kicsit részletesebben elmagyarázni:

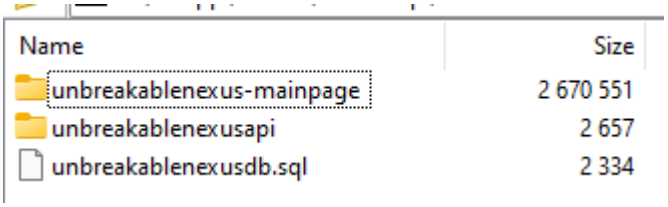
```php
<?php
if (isset($_POST["searchGomb"])) {
    require_once "database.php";
    $keresettnev = $conn->real_escape_string($_POST["keresettUser"]);

    $sql = "SELECT username,diamonds,spentdiamonds FROM users JOIN userstats ON users.id = userstats.id WHERE username LIKE '$keresettnev'";

    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        while($row = $result->fetch_assoc()) {
            echo "<script>document.getElementById('user_name').innerHTML = ' <img src=\"img/user_icon.png\" id=\"user-icon\"> Felhasználónév: " . $row["username"] . "';</script>";
            echo "<script>document.getElementById('diamonds').innerHTML = ' <img src=\"img/diamond.png\" id=\"diamond-icon\"> Gyémántegyenleg: " . $row["diamonds"] . "';</script>";
            echo "<script>document.getElementById('sdiamonds').innerHTML = ' <img src=\"img/spent_diamonds.png\" id=\"diamond-icon\"> Elköltött gyémántok: " . $row["spentdiamonds"] . "';</script>";
        }
    } else {
        echo "<script>document.getElementById('searchResult').innerHTML = 'Nincs ilyen felhasználó!'; document.getElementById('searchResult').style.display = 'block';</script>";
    }
}
?>
```

A kód első sorában egy feltétellel megvizsgáljuk azt, hogy a „loginGomb” névvel ellátott gomb megnyomásra került-e. Ha a felhasználó megnyomta a bejelentkeztető gombot akkor létrejön egy POST típusú kérés, és a „kereses” azonosítóval ellátott beviteli mezőbe beírt név alapján elkezdődik a szűrés. Ezután egy SQL utasítás segítségével ellenőrzésre kerül az, hogy létező felhasználóra keres-e a felhasználó. Ha létezik akkor a felhasználóhoz tartozó játékbeli adatokat soronként kiíratjuk oda, ahol a felhasználó statisztikái vannak felsorolva. Ha nem található ilyen felhasználónév, akkor az oldal figyelmeztet minket a beviteli mező alatt erről.

# Felhasználói Útmutató

## Program előkészítése

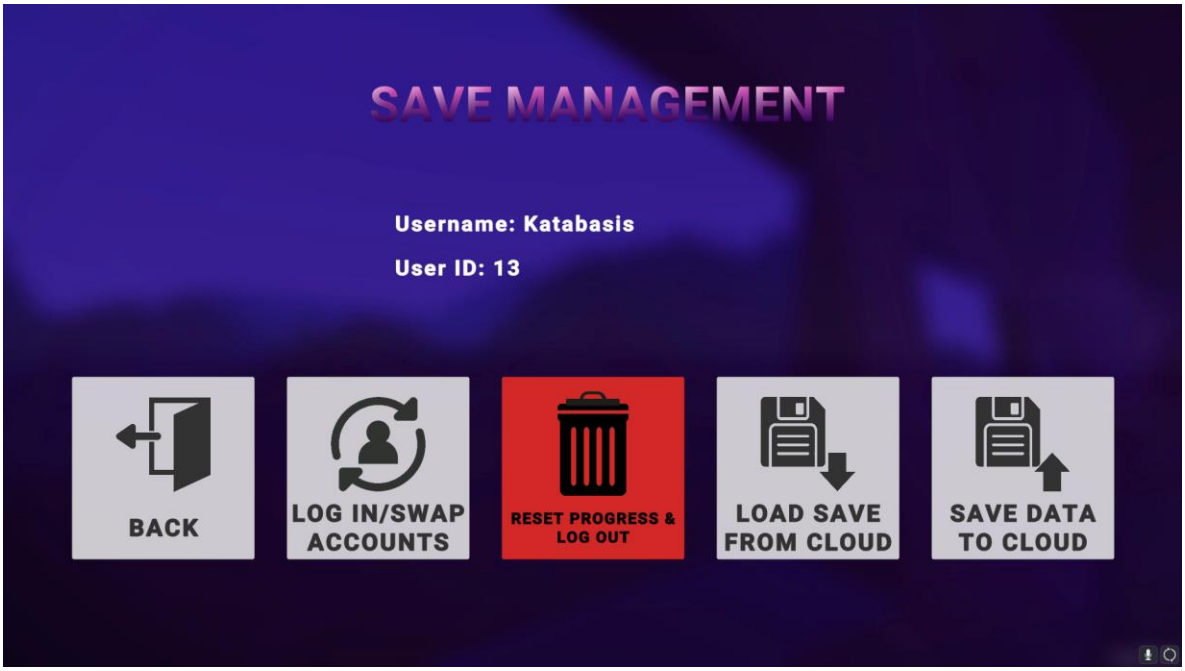
Miután a mellékelt GitHub repo-ról letöltöttünk ideje a játékot üzembe helyezni. Elsősorban kelleni fog a XAMPP az adatbázis, az API és a weboldal hostolására mivel „hard-coded” hogy a játék hova csatlakozik fel de a forráskódban csak néhány sor átírásával működhetne a távoli csatlakozás és lehetne a weben hostolni. Ha csak helyileg akarunk játszani és nem akarjuk a mentésünket átvinni sehová akkor ez a lépés kihagyható.




A letöltött adatbázist importáljuk be phpmyadminon vagy parancssoron keresztül. A letöltött weboldalakat így ahogy van tegyük a htdocs mappába különben a játék nem fog tudni csatlakozni a szerverhez. Ha ezzel megvagyunk indítsuk el a játékot.

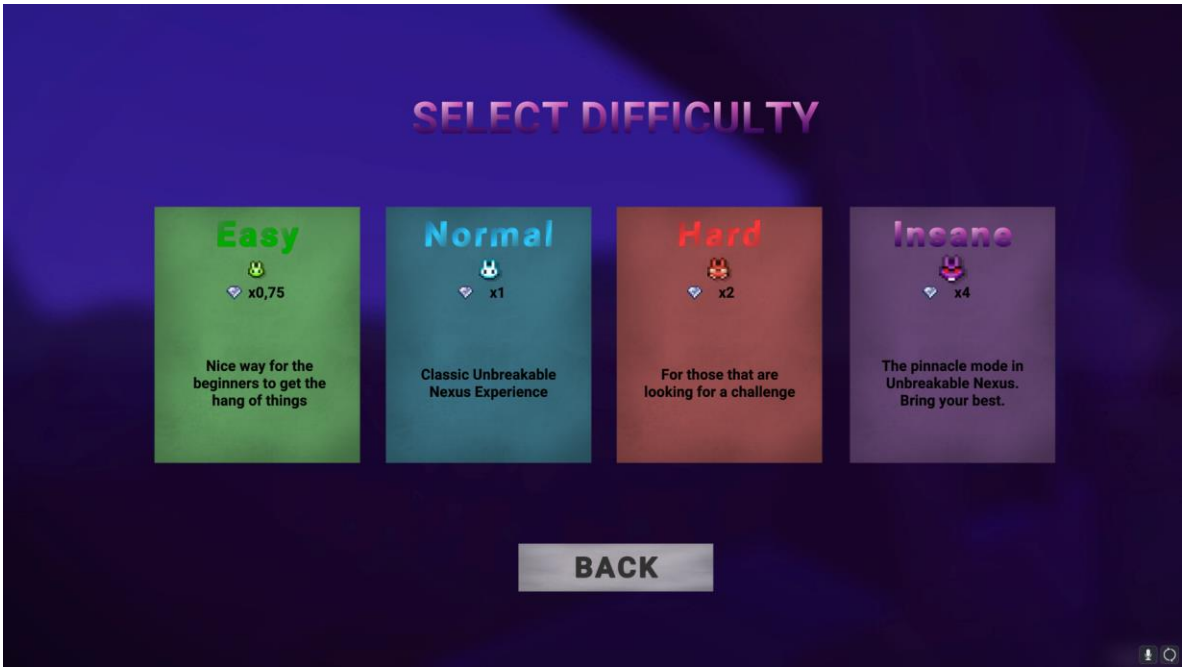
## Játékmenet


A játékba való belépés után a főmenü fog minket fogadni, ahol ha szeretnénk kihasználni az online mentés lehetőségét akkor a „Save Management” menüben tudunk regisztrálni fiókot. Amennyiben nem szeretnénk ezzel a lehetőséggel élni tovább léphetünk a „Play” gombbal.

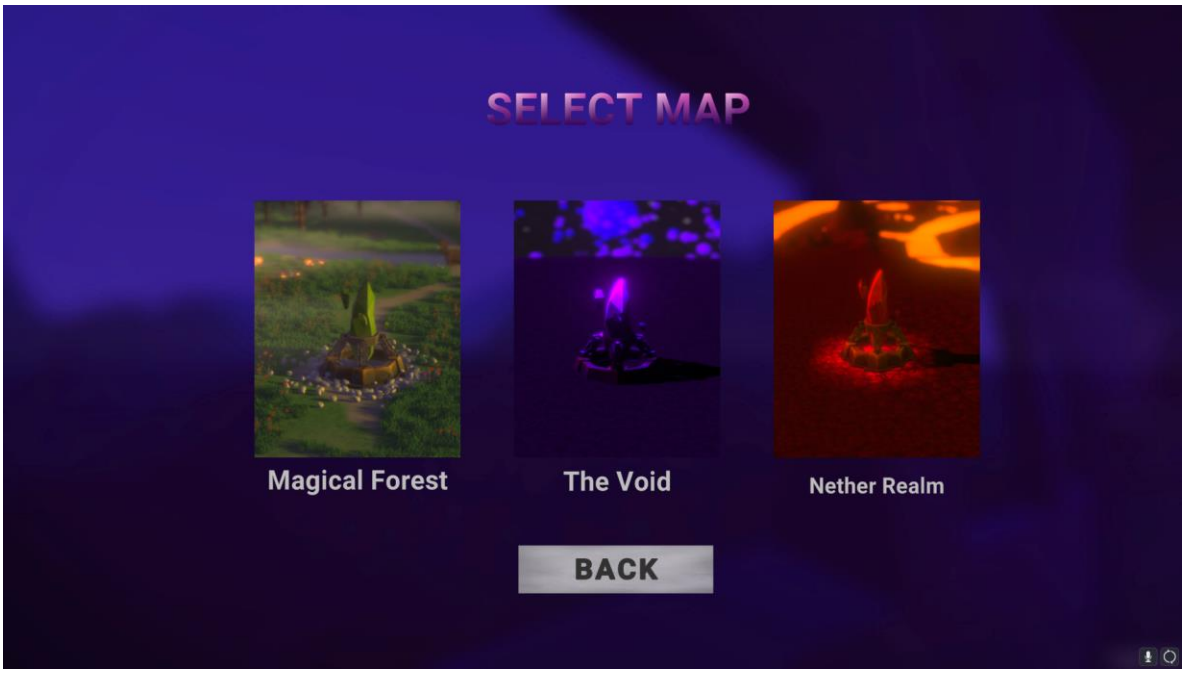




Itt az akcióválaszó menü vár minket, ahol tudunk a játékba továbblépés és a végleges fejlesztések növelése között választani, de mivel a játék elején még nincsen ezért lépjünk tovább a nehézség választó menübe.




Itt a játéknak a 4 nehézségi fokozata között tudunk választani, ha még a toronyvédelmi játék típussal akkor ajánlom az „Easy” nehézséget vagy ha már esetleg lenne tapasztalatunk akkor a „Normal” nehézség jobb lesz neked.Ha megtaláltuk a számunkra megfelelő nehézséget akkor a csatatér választó menü fog minket fogadni




A csatateret ne csak kinézet alapján válasszuk ki mivel minden egyes csatatérnek egyéni effektusa van ami valamilyen módon megfogja nehezíteni a játékmenetünket.

**Magical Forest – Mystical Guard:** Ameddig a főellenség él a többi ellenfél 50%-kal több életerőt és támadóerőt kap.

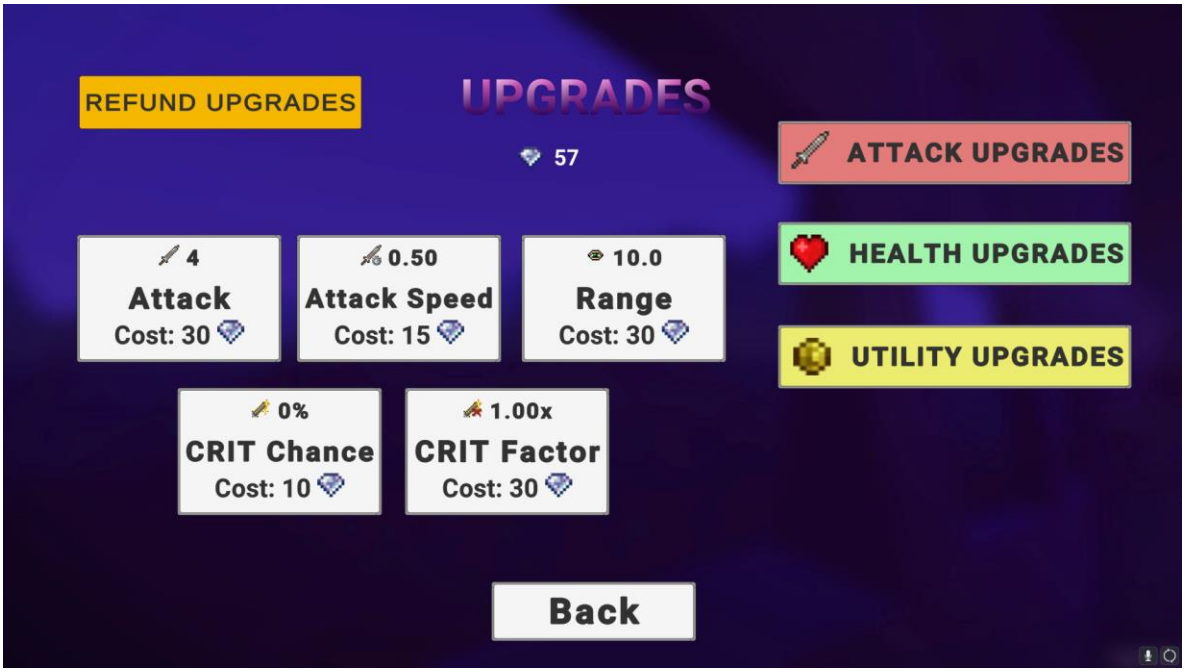
**The Void – Void Corruption:** Minden egyes alkalommal amikor a torony sebzést szenved el, a torony maximális életerejét lecsökkenti 10%-kal és a védelmét 5%-al stackenként. (Max. 5 stack, 5 másodpercig tart egy stack)

**Nether Realm – Purifying Flames:** Az ellenfél támadások égetik a tornyot, védelmen áthatoló sebzést okozva és a torony védelmét 25%-kal csökkenti (Max 6 stack de a védelemcsökkentés csak egyszer applikálódik)

Ha döntöttünk arról, hogy melyik csatatéren harcolunk, ez a UI fog minket várni.


Itt a három féle fejlesztés típussal tudjuk a tornyunkat jobbá tenni, hogy nagyobb hullámszámot érjünk el. Hullámonként a játék adni fog gyémántot, amivel végleges fejlesztéseket tudunk választani, amely a  játékmenetekben segíteni fog minket. Ezeket a

gyémántokat az akcióválasztó menüben található „Upgrade” menüben tudjuk fejleszteni.




**A játék fő célja hogy minél nagyobb hullámszámot érjünk el és hogy a maximalizáljuk az összes végleges fejlesztést.**

# Összegzés

A kezdeti nehézségek ellenére, amelyek között számos technikai akadály (mint pl.: a Unity Version Control felállítása csapatmunkához) és a csapatmunka kezdeti koordinációs problémái is helyet kaptak, végül minden kihívás csak jobbá tett minket ezen a téren. A projekt során nem csak az egyes technológiák mélyebb megismerése vált lehetővé, hanem a problémamegoldó képességünk is jelentősen javult. Minden újabb probléma, amellyel szembenéztünk, egy újabb lehetőséget jelentett arra, hogy kreatív megoldásokat találjunk, és ezáltal bővítsük tudásunkat.  Az együttműködés köztünk fokozatosan javult, ahogy egyre több időt töltöttünk együtt a projekten dolgozva. Megtanultuk értékelni minden egyes csapattag egyedi hozzájárulását, és rájöttünk, hogy a különböző perspektívák ötvözése által sokkal komplexebb és hatékonyabb megoldások születhetnek. Összességében, ez a projekt nem csak a programozási készségeinket fejlesztette, hanem a csapatmunkában való jártasságunkat is. Megtanultuk, hogy a sikeres projektmenedzsment olyan kulcsfontosságú készségeket igényel, mint az időgazdálkodás, az erőforrás-koordináció és a hatékony

kommunikáció. Ezek az ismeretek nemcsak a jelenlegi, hanem jövőbeli projektjeinkben is nagyban hozzájárulnak majd a sikerhez.