



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSI NYELVEK ÉS FORDÍTÓPROGRAMOK
TANSZÉK

Többszemélyes Torpedó-játék OpenGL-ben

Témavezető:

Fülöp Endre

doktorandusz Msc

Szerző:

Negrut Ákos

programtervező informatikus BSc

Budapest, 2023

Tartalomjegyzék

1. Bevezetés	4
1.1. A Torpedó általános bemutatása	4
1.2. Téma bevezetése	4
1.3. Motiváció	5
1.4. Cél	5
2. Felhasználói dokumentáció	6
2.1. Rendszerkövetelmények	6
2.1.1. Telepítés	6
2.1.2. Minimális gépigény	6
2.2. A Torpedó játéktere	7
2.2.1. Játéktér méret	7
2.2.2. Játéktér felépítése	7
2.3. Használati útmutató	8
2.3.1. Virtuális magánhálózat konfigurációja	8
2.3.2. Szerver	9
2.3.3. Kliens - Menü	11
2.3.4. Kliens - Játékmenet	15
2.4. Hibakezelés	19
3. Fejlesztői dokumentáció	22
3.1. Követelmény specifikáció	22
3.2. Felhasznált technológiák	23
3.2.1. Programozási nyelv	23
3.2.2. Fejlesztői környezet	24
3.2.3. Verzió-kezelés	25
3.2.4. Rajzoló API	25
3.2.5. Ablak- és input-kezelés	26

3.2.6.	Hang lejátszás	26
3.2.7.	Szöveg kirajzolása	27
3.2.8.	Hálózati kapcsolat	27
3.2.9.	Nvidia Nsight	27
3.3.	Elméleti háttér	28
3.3.1.	Vertex Buffer	28
3.3.2.	Shader program	28
3.3.3.	Phong megvilágítási modell irány fényforrással	31
3.4.	Megvalósítás	32
3.4.1.	3D Picking	32
3.4.2.	Cylindrical Billboarding	32
3.4.3.	Átlátszóság	33
3.4.4.	Fájlkezelés	34
3.4.5.	Hálózati adatfolyam	34
3.4.6.	Verzió egyezés ellenőrzése	35
3.5.	Felhasználói interakciók	36
3.6.	Osztályok	39
3.6.1.	Kapcsolatokat kezelő osztályok	39
3.6.2.	Szerverrel kapcsolatos osztályok	41
3.6.3.	Kliens osztály hierarchiája	44
3.6.4.	Kliens - Kisegítő osztályok	45
3.6.5.	Klienssel kapcsolatos osztályok	49
3.7.	Kódolási konvenciók	64
3.8.	Továbbfejlesztési lehetőségek	65
3.9.	Tesztelés	66
3.9.1.	Szerver futása	66
3.9.2.	Kliens futása - Menü	67
3.9.3.	Kliens futása - Játékmenet	68
3.9.4.	Kliens erőforrás-használata	70
4.	Összegzés	71
	Irodalomjegyzék	72
	Ábrajegyzék	75

Táblázatjegyzék	77
Rövidítésjegyzék	78

1. fejezet

Bevezetés

1.1. A Torpedó általános bemutatása

A Torpedó(angolul *Battleship*[1]) egy nemzetközi szinten ismert, körökre osztott játék. Az első, kereskedelmi változata 1931-ben jelent meg "Salvo" néven. A játék az idők során megjelent már táblajáték és számítógépes játék formájában is. Ezenkívül, a háttér-logika egyszerűségéből fakadóan, Torpedózni lehet akár csak papír és írószer használatával is.

A játék lényege, hogy két játékos, egy négyzetrácsra alapuló pályára rakja le az előre meghatározott számú és szélességű hajóit. Ezt követően, felváltva bemondanak koordinátákat, megcélozva ezzel az ellenfél játékos játék-terét. Ez folytatódik mindaddig amíg az egyik játékos összes hajója el nem fogy.

1.2. Téma bevezetése

A szakdolgozatomban a Torpedónak egy számítógépes játék változatát kívánom megvalósítani. Ebből fakadóan, a program interaktív elemeket és animációkat tartalmaz. 3 dimenziós (**3D**) grafikával, 2 dimenziós (**2D**) felhasználói felülettel, hangok és zenék lejátszásával teszem élvezhetővé a felhasználói élményt. Egy játékmenetben két játékos képes játszani egymás ellen egy vagy több számítógépen. Az utóbbi esetben ez hálózaton keresztül történik. A program szerver-kliens architektúrára alapul. Ezen felül viszont a kliens is szétbontható egy "menü" és egy "játékmenet" részre. A menü elsődleges célja, hogy egy interaktív, **2D**-s

felhasználói felületen keresztül lehessen csatlakozni a szerverre. A játékmenet célja a Torpedó-játék **3D**-s megjelenítése, a játék állapotának szöveges kirajzolása és a játék folyamatát befolyásoló input események kezelése. A szerver feladata pedig egy Torpedó játékmenet autoritatív módú vezénylése két kliens-program között.

1.3. Motiváció

Az egyetemi évek során megismerkedtem a számítógépes grafika és a hálózati kommunikáció által felkínált lehetőségekkel. Ezt követően hamar párhuzamot hoztam a játékfejlesztés és az előbb említett témakörök között. A motivációt e téma megválasztásában ez az érdeklődés adja.

1.4. Cél

Céлом egy olyan számítógépes játék készítése, amely logikáját tekintve egyszerű, de tartalmazza a mai játékokban megtalálható, alapvető elemeket. Ezek közé tartozik a grafika, hang, felhasználói felület és hálózati kommunikáció. További cél, hogy a projektet az alapokról építsem fel, csak néhány segédkönyvtár segítségével. Szándékosan el akarom kerülni a *játék-motorok*[2] használatát, ezzel tapasztalatot szerezve a használatuk nélküli játékfejlesztésbe. Ez által, a jövőben könnyebb lesz egy saját *játék-motor*[2] fejlesztése, illetve a meglévők működését és felépítését is könnyebb lesz átlátni.

A dokumentáció tartalmaz egy felhasználói útmutatót a program futtatására, használatára és a hibák kezelésére. Szintén tartalmaz egy fejlesztői dokumentációt, amely leírja a követelmények specifikációját, a felhasznált eszközöket és könyvtárakat, a megvalósítás módszereit, a projekt felépítését és a tesztelést. Ezen felül ad egy minimális elméleti háttérrel a projekt számítógépes grafikához kapcsolódó fogalmaihoz.

2. fejezet

Felhasználói dokumentáció

2.1. Rendszerkövetelmények

2.1.1. Telepítés

1. A futtatáshoz szükséges a *Visual Studio Redistributable 2019*[3] letöltése és telepítése.
2. Nagytávolságú hálózaton(WAN) keresztüli játékhoz virtuális magánhálózatot(VPN) szolgáltató szoftver telepítése szükséges. Ehhez én a *Radmin VPN*[4] nevezetű ingyenesen használható szoftvert ajánlom.



2.1. ábra. Radmin VPN Logója (<https://www.radmin-vpn.com/img/logo-03.png>)

2.1.2. Minimális gépigény

Operációs Rendszer: 7-es verziószámú Windows.

Processzor: 2 maggal rendelkező processzor.

Videókártya: OpenGL 4.6 támogatottsággal és 512 MB memóriával rendelkező videokártya.

Memória: 2 gigabájt.

Szabad terület: 1 gigabájt.

Egyéb: Több számítógépen való futtatás esetén szükség van egy alaplapha beépített, vagy azon kívüli hálózati kártyára és internetes kapcsolatra.

2.2. A Torpedó játéktere

2.2.1. Játéktér méret

A játék logikáját tekintve csak annyi kikötés van, hogy a játéktér reprezentáló négyzetrács mérete négyzetes legyen. A programban viszont ez az egyszerűség kedvéért le van korlátozva három lehetőségre:

Játéktér mérete	Hajók száma			
	1x1	2x1	3x1	4x1
5x5	3 db	1 db	1 db	0 db
7x7	4 db	3 db	2 db	0 db
9x9	6 db	4 db	3 db	1 db

2.1. táblázat. Egy játékos hajóinak a mennyisége a játéktér méretére és a hajók méretére lebontva.

A **2.1** táblázatban látható, hogy a játéktér méretének függvényében hány hajót kell leraknia a játékosoknak. A hajók száma le van bontva az általuk elfoglalt játékmezők mennyisége alapján. A játéktér mérete a szerver-program konfigurációs fáájisában állítható be.

2.2.2. Játéktér felépítése

A játéktér koordinátarendszere úgy van felépítve, hogy játékostól függetlenül, mindig egy játékos saját csatahajója szemszögéből kezdődik a bal felső sarokból. Tehát például egy "7x7"-es játéktér esetén a **2.2** táblázat mutatja be a koordinátákat.

A1	B1	C1	D1	E1	F1	G1
A2	B2	C2	D2	E2	F2	G2
A3	B3	C3	D3	E3	F3	G3
A4	B4	C4	D4	E4	F4	G4
A5	B5	C5	D5	E5	F5	G5
A6	B6	C6	D6	E6	F6	G6
A7	B7	C7	D7	E7	F7	G7
Csatahajó pozíciója						

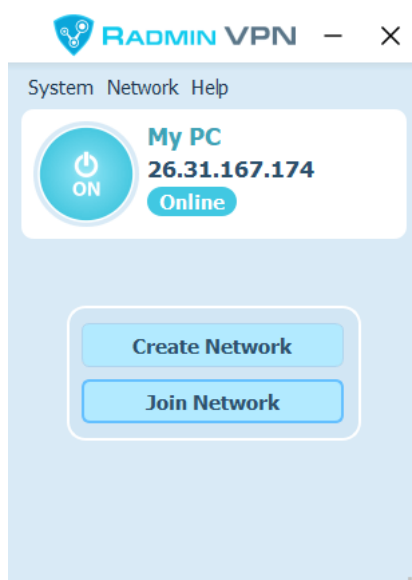
2.2. táblázat. 7x7-es játéktér koordinátarendszere egy játékos csatahajójának szemszögéből.

2.3. Használati útmutató

2.3.1. Virtuális magánhálózat konfigurációja

Ez egy opcionális rész, hogyha **WAN**-on keresztül akarok játszani. Ha egy gépen, vagy helyi hálózaton(**LAN**) tervezek játszani, akkor ez a rész átugorható.

A *Radmin VPN*[4] program telepítését és indítását követően a **2.2** ábrán látható felület fogad. Először is meg kell győződni róla, hogy a nagy gomb kék színű és "ON" állapotban van. Ha nincs, akkor rá kell kattintani. A pontokkal elválasztott 4 számból álló számsor a **VPN**-es internet protokoll(**IP**) cím. Az ábrán ez a "26.31.167.174", de ez mindenkinél más.



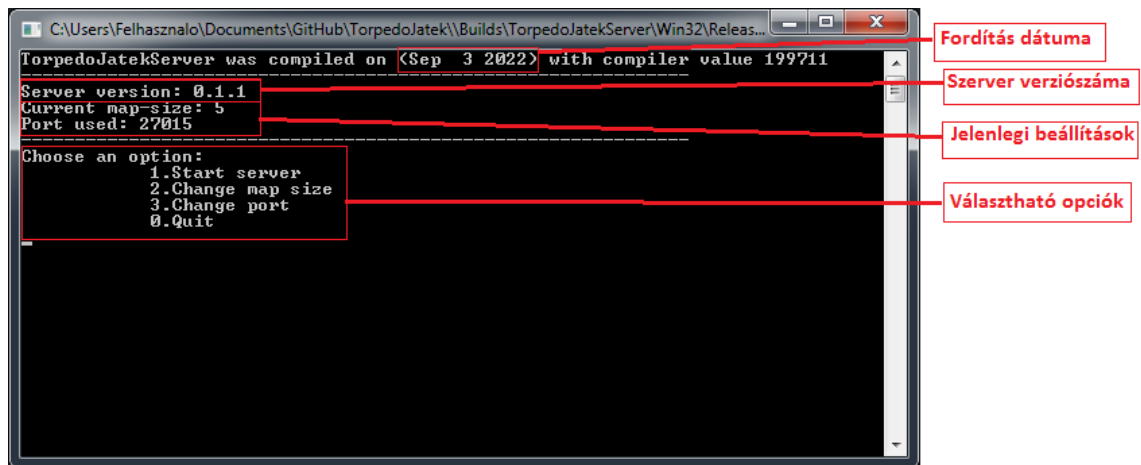
2.2. ábra. Radmin program felülete (https://www.radmin-vpn.com/images/gallery_main_page/en/main_light.png)

Szerver üzemeltetőként a "Create Network" gombra kattintva kell létrehozni egy privát hálózati csoportot. Ennek a csoportnak nevet és jelszót kell adni. Majd tájékoztatni kell a játékosokat ennek a csoportnak a nevééről és jelszaváról. Ezt követően figyelni lehet a *Radmin* felületét, mert az jelzi ha a játékosok sikeresen csatlakoztak a privát hálózathoz. Végül indítható is a Torpedó-játék szerver programja.

Játékosként a "Join Network" gombra kattintva kell csatlakozni a Radmin privát hálózati csoporthoz. A csoport nevét és jelszavát a szervert üzemeltetőtől kell elkérni. Ezeket az adatokat megadva és csatlakozva jelenik meg a csoport a *Radmin* felületén. Itt, a szervert üzemeltető gépének a neve mellett látható a *Radmin* által kiosztott IP címe. Ennek a címnek a segítségével tud a játékos a kliens programban csatlakozni a szerverre.

2.3.2. Szerver

A szerver a "Builds/TorpedoJatekServer/Win32/Release" útvonalon található "TorpedoJatekServer.exe" futtatható állománnyal indítható.



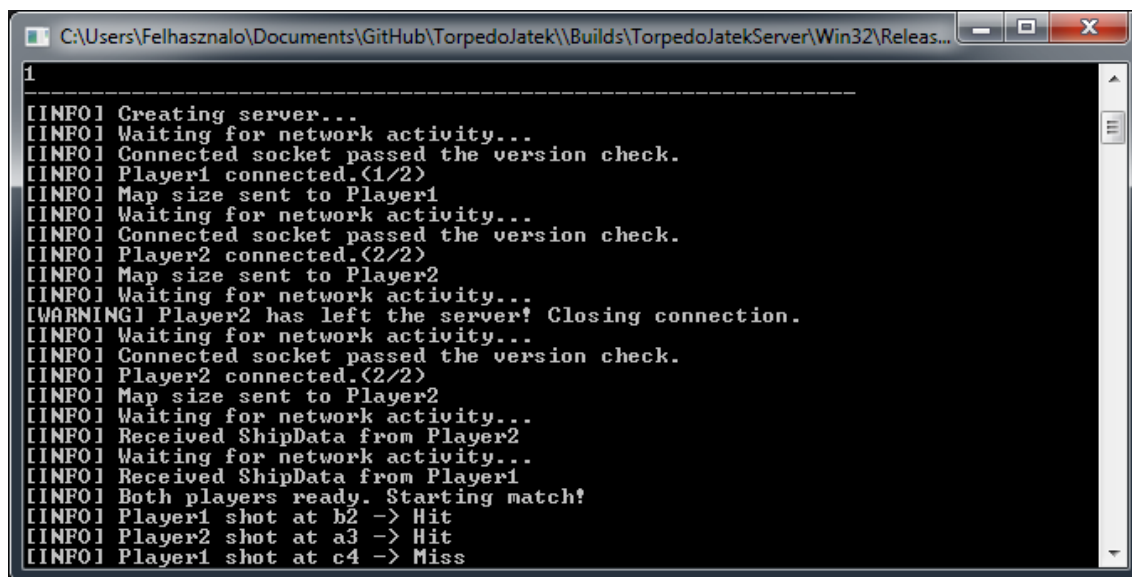
2.3. ábra. A szerver konzolos külalakja a program indítását követően.

A futtatást követően a **2.3** ábrán látható konzolos felület fogad. Ezen látható a fordítás dátuma, amely jelzi, hogy az adott verziójú szerver-program melyik napon lett kész. A szerver verziószáma is megjelenik itt. Ez azért hasznos információ, mert a szerver csak megegyező verziószámmal rendelkező kliens kapcsolatokat fogad el. Továbbá szemügyre vehetőek a jelenlegi beállítások is.

A szerver-program futáskor és újra induláskor is mindig az alapértelmezett beállításokat állítja be. "5x5"-ös méretet állít be a játék-térnek és a "27015"-ös számot állítja be port-számnak.

A futtatást követően a program felhasználói bemenetre vár, ezért választani kell a lehetséges opciók közül. Választani az adott opció számának a beírásával, majd az "Enter" billentyű lenyomásával lehet:

- 1: Szerver felkészítése és elindítása a kapcsolatok fogadására. Ha ezt kiválasztom, akkor egy játékmenet végéig további teendőm nincs. A szerver a 2.4 ábrán látható módon kezeli a kliens kapcsolatokat és szöveges tájékoztatást ad a játékmenet helyzetéről. Egy játékmenet végén a szerver újraindul és visszakerül a program állapota a kezdetleges állapotba.



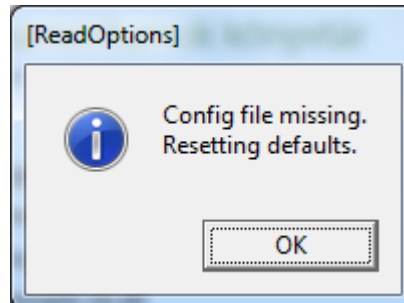
```
1
-----
[INFO] Creating server...
[INFO] Waiting for network activity...
[INFO] Connected socket passed the version check.
[INFO] Player1 connected.<1/2>
[INFO] Map size sent to Player1
[INFO] Waiting for network activity...
[INFO] Connected socket passed the version check.
[INFO] Player2 connected.<2/2>
[INFO] Map size sent to Player2
[INFO] Waiting for network activity...
[WARNING] Player2 has left the server! Closing connection.
[INFO] Waiting for network activity...
[INFO] Connected socket passed the version check.
[INFO] Player2 connected.<2/2>
[INFO] Map size sent to Player2
[INFO] Waiting for network activity...
[INFO] Received ShipData from Player2
[INFO] Waiting for network activity...
[INFO] Received ShipData from Player1
[INFO] Both players ready. Starting match!
[INFO] Player1 shot at b2 -> Hit
[INFO] Player2 shot at a3 -> Hit
[INFO] Player1 shot at c4 -> Miss
```

2.4. ábra. A szerver a kliens kapcsolatok kezelése közben.

- 2: Játéktér méretének módosítása. Alapértelmezetten ez "5x5", vagyis 25 mezőből álló játéktérre jelent. Az opció kiválasztását követően lehetőség van választani az "5x5", "7x7" és "9x9"-es méret között, az 5, 7 vagy 9-es input megadásával.
- 3: Ezzel lehetséges módosítani, hogy mely porton keresztül várja a szerver a kliens kapcsolatokat. Alapértelmezetten ez "27015". Ennek a módosítása csak akkor szükséges, ha egy másik szoftver már használja ezt a portot. Az opció kiválasztását követően megadható az új szám, bemenet formájában.
- 0: A szerver program bezárása.

2.3.3. Kliens - Menü

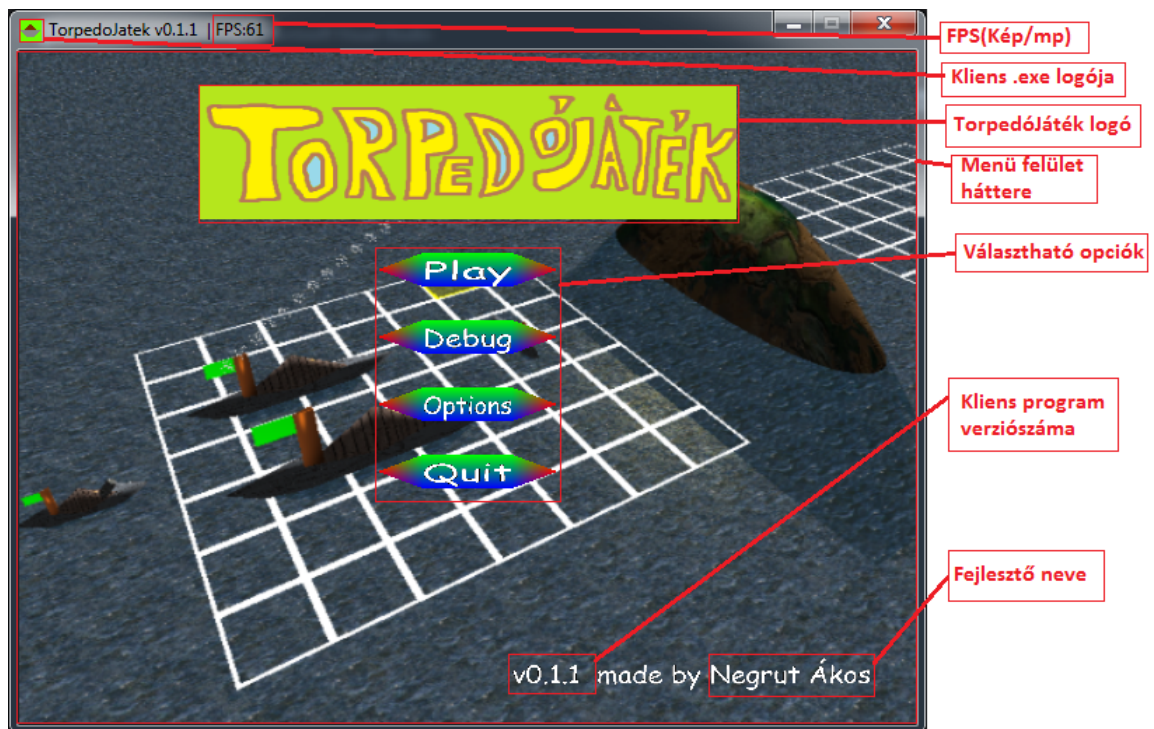
A kliens a "Builds/TorpedoJatekClient/Win32/Release" útvonalon, a "TorpedoJatekClient.exe" indításával futtatható.



2.5. ábra. Hiányzó beállításokat jelző üzenetdoboz.

Első indítás vagy hiányzó beállításokat tartalmazó fájl esetén a 2.5 ábrán lévő üzenetdoboz fogad. Ilyenkor a program elkészíti a hiányzó fájlt, a kódba beégetett, alapértelmezett beállításokkal:

- 800 pixel szélességű és 600 pixel magasságú felbontás.
- Kikapcsolt teljes képernyős üzemmód.
- Bekapcsolt függőleges szinkronizáció.
- Maximumra állított zene- és hang-effekt hangerő.

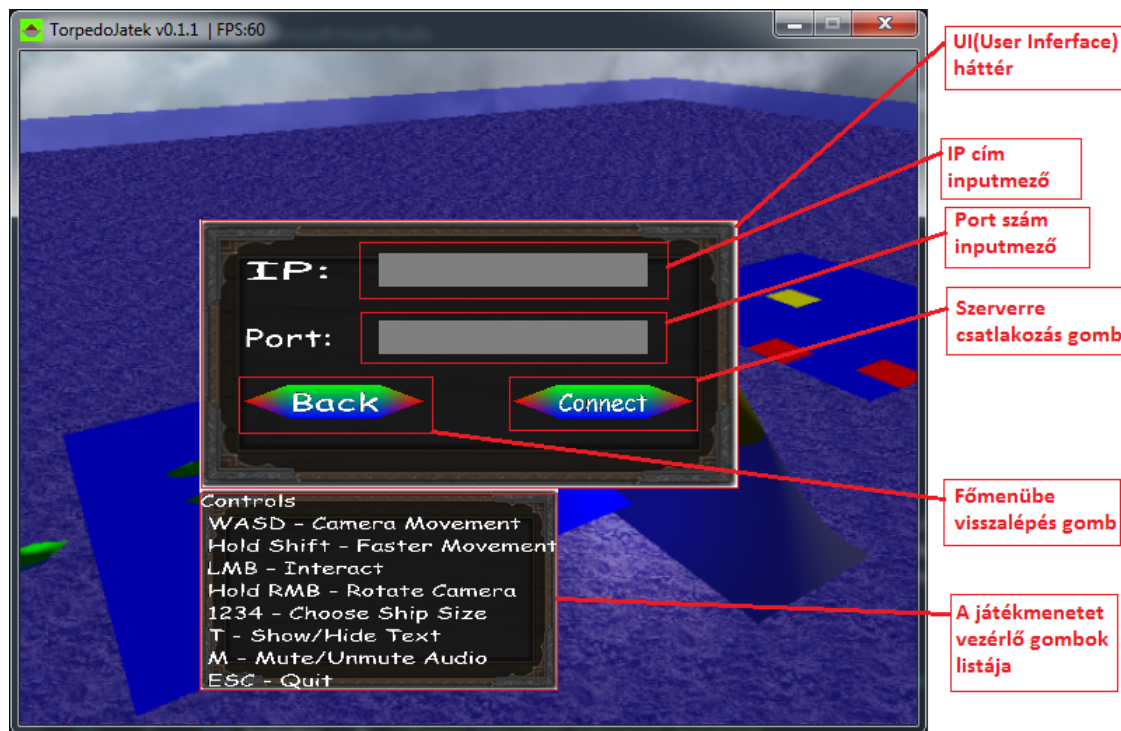


2.6. ábra. A kliens-program futtatását követően a fő menü fogad.

Kezdetben a program egy Windows ablakban indul el, a fő menüt betöltve. A menüt a bal egérgomb segítségével tudom vezérelni. A 2.6 ábrán látható főmenü felület elemei a következők:

- Az ablak címsorában lévő "TorpedoJatekClient.exe" fájl ikonja, illetve a másodpercenként kirajzolt képek mennyisége(FPS).
- A képfájlból betöltött játék logó.
- A menü háttere, amely 4 másodpercenként változik, a betöltött háttérképek között. A képek tartalmazznak a szoftver régebbi verzióiban készült képernyőképeket.
- A főmenü választható opcióinak a gombjai. Bal egérgomb segítségével tudok választani közülük. Hogy melyik mit csinál pontosan, azt az ezt követő alfejezetekben fejtem ki.
- A kliens program verziószáma. Ez felhasználói szempontból azért fontos, mert csak megegyező verziószámmal rendelkező szerverre lehet felcsatlakozni.
- A fejlesztő, tehát az én nevem.

Play opció



2.7. ábra. Szerverre csatlakozás al-menüjének a felülete.

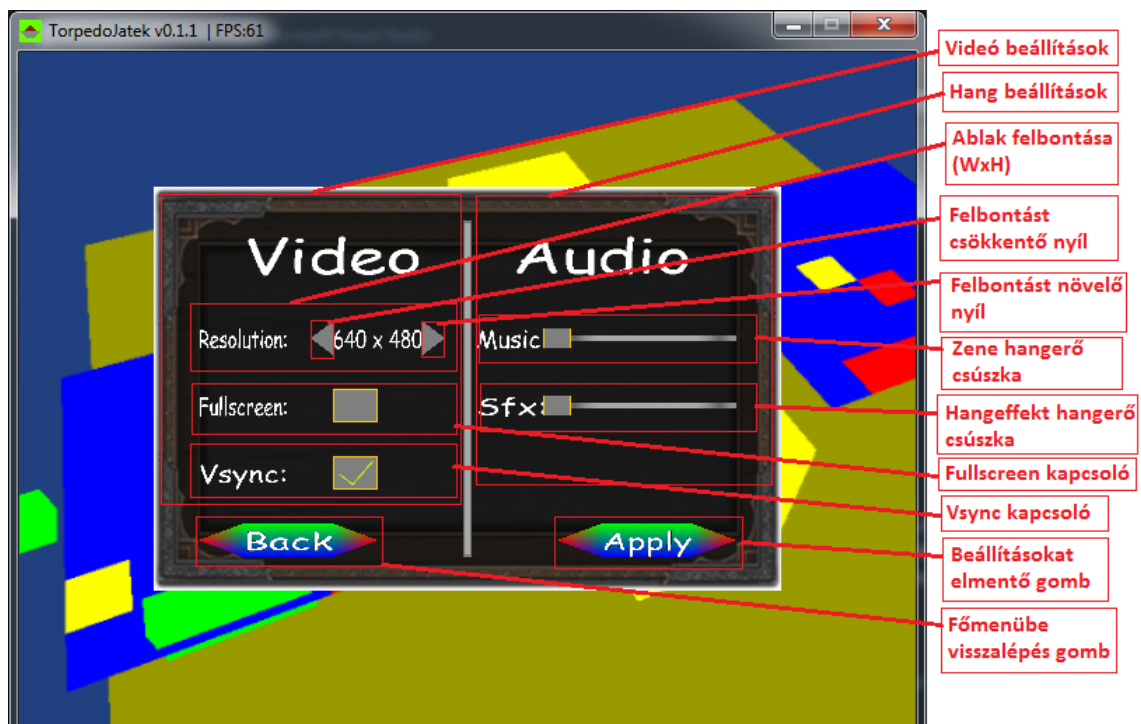
A főmenüben a "Play" gombra kattintva átkerülök a **2.7** ábrán látható almenübe. Itt lehetőség adódik egy aktívan futó és csatlakozásokat váró, Torpedó-szerverre való csatlakozáshoz. A "Play" almenü részei a következők:

- A felhasználói felület(**UI**) képből betöltött háttére.
- **IP** cím input mező. Ebbe, bal egérgombbal való kattintást követően, be tudom írni a szerver **IP** címét. Ha nem akarok tovább gépelni, akkor az "Enter" vagy "Esc" billentyűkkel tudom befejezni a gépelést. Az input mező csak számokat, illetve a "."(pont) karaktert fogadja el bemenetként. Ha ugyanazon a számítógépen fut a szerver, akkor a "127.0.0.1" lokális **IP** címet kell ide beírni. **LAN** keresztüli játék esetén, azon számítógép **IP** címét kell beírni, amelyen a szerver fut. **WAN**-on keresztül történő játék esetében pedig a *Radmin VPN*[4] által, a szervert üzemeltetőnek kiosztott **IP** címet kell beírni.
- Portszám input mező. Alapértelmezetten a "27015"-ös port számon várja egy szerver a kapcsolatokat. Viszont a szervert üzemeltetőnek lehetősége van ettől eltérni. Ebben az esetben szükséges elkérni a módosított port számot tőle. A bemenet megadási módja teljesen azonos az előbb említett, *IP cím input mezőhöz*.
- A szerverre csatlakozási gomb. A megfelelő szerver adatok megadását követően a kliens ezzel próbál kapcsolatot létesíteni a szerver-programmal.
- Főmenübe visszalépés gombja.
- A játékmenetet vezérlő gombok listája. Ezek írják le, hogy sikeres szerverre csatlakozást követően, milyen lehetőségek vannak a játékmenet vezérlésére, illetve mely bemeneti gombokkal tehetem meg ezt. Ezen vezérlők listáját a 2.3.4 alfejezetben fogom kifejteni.

Debug opció

A "Debug" opciót kiválasztva a fő menüből egy olyan játékmenet állapotba kerül a program, ahol szerverre nem csatlakozik és a *Torpedó*[1] játék alapvető logikája sincs jelen. Ez az opció a játék néhány grafikai és hang elemének a tesztelésére szolgál, illetve a **3D**-s szintér megtekintésére.

Options opció



2.8. ábra. Beállításokat kezelő almenü.

Ebben a menüpontban néhány beállítás módosítására és elmentésére van lehetőség. A 2.8 ábrán látszik, hogy a videó és a hang beállítások szét vannak választva. Abban különböznek, hogy a hang beállítások módosításra azonnal érvényesülnek, még a videósok csak a mentést követően.

Resolution: Ablak felbontása. Ezzel a Windows ablak pixelben mérendő "szélesség x magasság" méretét lehet kiválasztani. A bal nyíllal kisebb felbontást, jobb nyíllal pedig nagyobb felbontást lehet beállítani.

Fullscreen: Teljes képernyős és ablakos megjelenítés közti váltogatásra ad lehetőséget.

Vsync: Függőleges szinkronizáció(Vsync) be- és kikapcsolása. Ezt célszerű bekapcsolt állapotban tartani, mivel a kikapcsolt állapot fölöslegesen megterheli a videokártyát.

Music: A zene hangereje a menüben és játékmenetben egyaránt. A négyzet alakú csúszka mozgatásával lehet a hangerőt halkabbra vagy hangosabbra állítani.

Sfx: A játékmenetben lejátszódó hangeffektek és a menü kattintások hangereje. Szintén a négyzet alakú csúszkával állítható a hangerő.

Back: Visszalépés a főmenübe, ezzel együtt elvetve az addig el nem mentett módosításokat. A módosított hangerő ilyenkor visszaáll a mentett értékre.

Apply: Módosítások mentésére és a videó beállítások érvényesítésére szolgál. Ezen felül elmenti az összes beállítást egy merevlemezen lévő fájlba. Ennek köszönhetően ha a jövőben elindítom a kliens programot, akkor az elmentett beállítások alapján fog az betölteni.

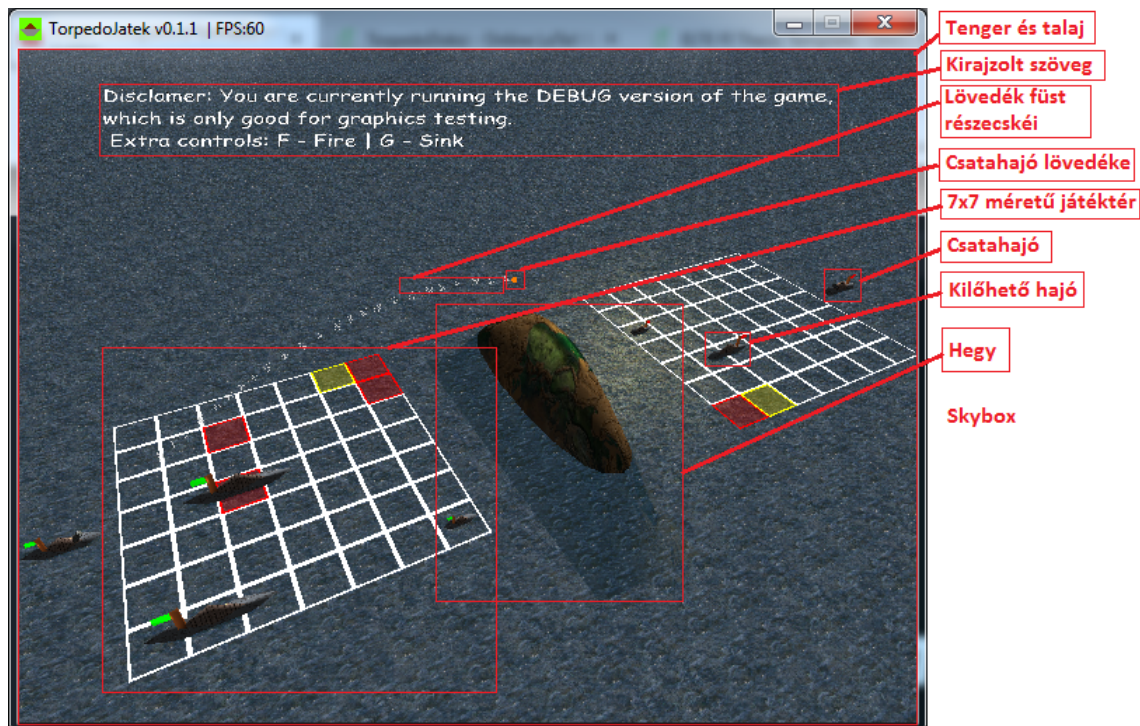
Quit opció

Kilépés a kliens programból.

2.3.4. Kliens - Játékmenet

A kliens program úgynevezett "játékmenet" állapotba tud kerülni a "menü" állapotot követően, ha a fő menüből:

1. A 2.3.3 alfejezetben leírt lehetőségeket követve, a szerverre való csatlakozás sikeres.
2. "Debug" gombra való kattintás történik.



2.9. ábra. Képernyőkép "Debug" módban indított játékmenetről.

A 2.9 ábrán láthatóak a kliens elemei, játékmenet állapotban:

- A tenger és az alatta lévő talaj. Az előbbi animáltan mozog, hogy folyást szimuláljon. Csak a színtér szépítésére szolgál. Nem tartozik az interaktív elemek közé.
- A kirajzolt szöveg. Célja, hogy tájékoztassa a játékost a következő teendőiről, a játék állapotáról. Továbbá jelzi azt is, ha egy hajó lerakása valamilyen okból kifolyólag nem lehetséges egy adott játéktérre.
- Lövedék által eleresztett füst-részecskék. A lövedék pályájának könnyebb követhetőségére szolgál. Nem interaktív.
- Csatahajó lövedéke. Egy játékos csatahajójából távozó, gömb alakú lövedék. Ez egy félkört leíró pályán, animált módon halad a célzott játéktér felé.
- "7x7"-es játéktér. 7 oszlopos és 7 soros játéktér-területből álló tér. Erre a területre tudja egy játékos letenni a hajóit, illetve lőni. Egy adott játéktér színe jelzi annak egy különleges állapotát:

Piros: Találat. Egy játékos általi lövés eltalált ezen a játéktérre egy hajót.

Sárga: Mellé lövés. Egy játékos lőtt már erre a játéktérre, de nem volt ott hajó.

Fehér: Egy éppen kilőtt lövedék célpontja.

Zöld: Egy legalább "2x1"-es méretű hajónak a háta lerakható erre a játékmezőre.

Kék körvonal: Az egér kurzora erre a játékmezőre mutat.

- Csatahajó. Ez a hajó lő az ellenfél játékterének a kiválasztott pozíciójára. A vesztes csatahajója animált módon elsüllyed a játék végén. A zöld színű zászlóval rendelkező csatahajó mindig a játékosé, a piros mindig az ellenfélé.
- Kilőhető hajó a játéktéren. Egy hajó "1x1", "2x1", "3x1" vagy "4x1" játékmezőt foglal el a játéktéren. A **2.1** táblázat mutatja be, hogy a játéktér méretétől függően hány és mekkora hajót tud lerakni egy játékos. A zöld színű zászlóval rendelkező hajó mindig a játékosé, a piros mindig az ellenfélé. Ha egy hajó által elfoglalt összes játék-mezőt találat ér, akkor a hajó animált módon elsüllyed. Egy hajó elhelyezkedhet a játéktéren vízszintes és függőleges alakban, de átlósan nem.
- A hegy. Célja, hogy szépítse a színteret, illetve hogy szimbolizálja ennek a szimmetriai középpontját. Nem interaktív. A csatahajó lövedékek e felett haladnak át egy körpályán.
- *Skybox*, vagyis a színteret körülvevő égbolt és táj. Szintén csak szépíti a színteret.

Játékmenetet vezérlő gombok

A játékmenetben interaktív módon, lehetőség nyílik a program vezérlésére. Ez egér és billentyűzet használatával lehetséges. A lehetséges vezérlő gombok a következők:

- W, A, S vagy D billentyűk - A nézeti kamera mozgatása rendre előre, balra, hátra vagy jobbra.
- Shift billentyű nyomva tartása - Nézeti kamera mozgatási sebességének a növelése.
- Bal egérgomb - Interakció a játéktér mezőivel.
- Jobb egérgomb nyomva tartása - Nézeti kamera mozgatása az ezt követő egérmozgatás irányába.
- 1, 2, 3 vagy 4 billentyűk - A játékmenet hajó lerakási szakaszában a letenni szándékozott hajó mérete.
- T billentyű - Az ablakba kirajzolt szöveg eltüntetése vagy megjelenítése.

- M billentyű - Összes hang némítása vagy a némítás feloldása.
- ESC billentyű - Kilépés a játékmenetből, vissza a fő menübe.

Játékmenet szerverre csatlakozással

A sikeres Torpedó-játék szerverre való csatlakozást követően a kliens ablak felületébe kirajzolt szöveg tájékoztatja mindig a játékost a következő lépésről. Az alapvető sorrend a következő:

1. Hajók lerakása a játékos saját játékterére egészen addig, amíg van lerakandó hajó:
 - (a) Hajóméret kiválasztása az 1, 2, 3 vagy 4 billentyűk segítségével. Itt, a játékos csak akkor jut el a következő fázisba, ha olyan hajóméretet választ, amelyből még van lerakandó hajó.
 - (b) Hajó elejének a lerakása. Bal egérgomb segítségével kell kiválasztani egy játékmezőt a játékos saját játékterén.
 - (c) Ezt követően "2x1", "3x1" és "4x1" méretű hajók esetén a hajó hátát is le kell tenni. Ilyenkor a zöld színű játékmezők jelzik a játékos számára a lehetséges pozíciókat.
2. Esetleges várakozás amíg az ellenfél is végez a hajók lerakásával.
3. Lövés az ellenfél játékterére, illetve várakozás az ellenfél lövésére a játékos saját játékterére. A lövést szintén a bal egérgommbal lehet elvégezni. Ez a lépés folytatódik mindaddig, amíg valamelyik játékos összes hajója el nem fogy vagy még valamelyik ki nem lép a játékmenetből.
4. A vesztes játékos számára kirajzolódnak az ellenfél életben maradt hajói, amelyeket meg tud tekinteni.
5. "Esc" billentyűvel visszalépés a főmenübe. Ezt követően a kliens-program ismét a "menü" állapotába tér vissza.

Játékmenet Debug módban

A "Debug" módban indított játékmenetben lehet bejárni a **3D**-s színteret a kamera mozgatása által, illetve ki- vagy bekapcsolni a hangokat és a szövegkiíratást. Vissza lehet lépni a fő menübe. A 2.3.4 alfejezetben megismert vezérlőkön felül,

további extra vezérlési lehetőségek is vannak. Ezek a csatahajó lövésének és elsüllyedésének a grafikai tesztelésére szolgálnak:

- F billentyű - Csatahajó lövés animációjának a tesztelése.
- G billentyű - Csatahajó elsüllyedési animációjának a tesztelése.

2.4. Hibakezelés

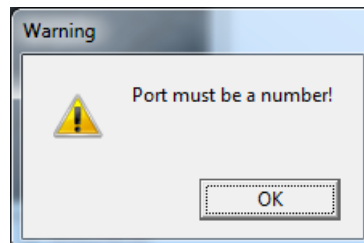
A hibák kezelése megtörténik a szerver és a kliens oldalon is. Viszont az esetek többségében, a hiba jelzését követően, a program bezár.

A hibajelzés a szerver oldalon a standard kimenetre történik, mivel egy konzolos applikációról van szó. Ha a szerver már kapcsolatokat vár vagy kezel, akkor a "WARNING"-al kezdő sorok jelzik a kisebb, nem szokványos eseteket. Ilyen például, ha az egyik kliens kilép a játékmenetből, annak kezdetét követően. Az "ERROR"-al kezdődő sorok pedig a súlyosabb hibákat jelzik, amely következtében a program szokványos működése már nem garantálható.

A kliens oldalon üzenetdobozok segítségével történik a hibák jelzése. Ez egyben a kliens program futási folyamatát is blokkolja, amíg nincs le "OK"-zva.

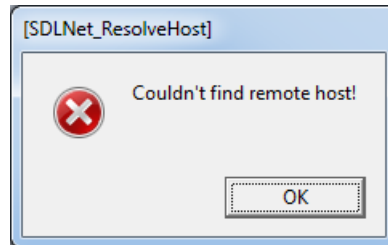
Szerverre való csatlakozáskor ha a program a megadott **IP** cím és portszám alapján képes találni egy olyan szervert, amely kapcsolatokra vár a megadott port számon, akkor a kliens átkerül játékmenet állapotba. Ellenkező esetben az alábbi hibák fordulhatnak elő:

- Üresen hagyott vagy rossz formában megadott port szám. Ekkor a 2.10 ábrán lévő üzenetdoboz ugrik fel. Ilyenkor meg kell nézni, hogy a portszám input-mező nem lett esetleg üresen hagyva, illetve hogy nem tartalmaz-e "."(pont) karaktert.



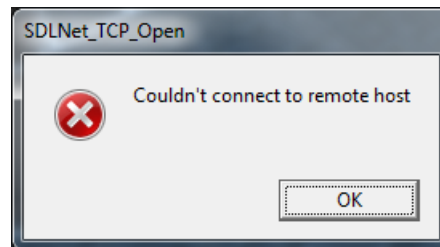
2.10. ábra. Hibás portszám bemenetet jelző üzenetdoboz.

- A megadott **IP** címen keresztül nem elérhető számítógép. Ekkor a 2.11 ábrán lévő üzenetdoboz jön fel, majd kilép a program. Ilyenkor, a kliens újra indítását követően meg kell győződni a beírandó **IP** cím helyességéről. Az 2.3.3 alfejezet "IP cím input mező" részében kitértem ennek a hiba-javítására is.



2.11. ábra. Rossz IP cím megadását jelző üzenetdoboz.

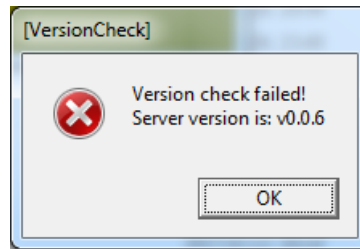
- Rossz a megadott portszám. Ebben az esetben bár a program megtalálja a távoli számítógépet, amelyre csatlakozni kíván, viszont az nem fogad kapcsolatokat a megadott port-számon. Ekkor a 2.12 ábra üzenet-doboza ugrik fel és kilép a program. A hibajavítás azonos az előző pontban írtakhoz, a 2.3.3 alfejezet "Portszám input mező" résznél kitértem, hogy mire kell odafigyelni ezzel kapcsolatban.



2.12. ábra. Rossz portszám megadását jelző üzenetdoboz.

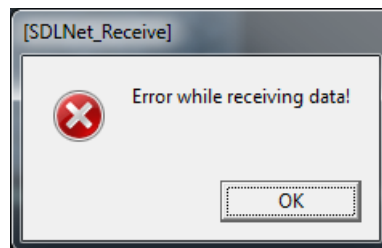
A sikeres csatlakozást követően is előfordulhatnak hibák. Ezek mind a kliens program bezárásához vezetnek:

- A szerver amire csatlakoztam más verziószámmal rendelkezik. Ilyenkor a szerver elküldi a saját verziószámát, majd bontja a klienssel a kapcsolatot. Egy példa a 2.13 ábrán látható, ahol jelzi, hogy a szerver verziószáma "0.0.6". Szerver üzemeltetőként ajánlott mindig a legfrissebb verziójú szerver-programot futtatni. Játékosként pedig másik szerverre kell csatlakozni.



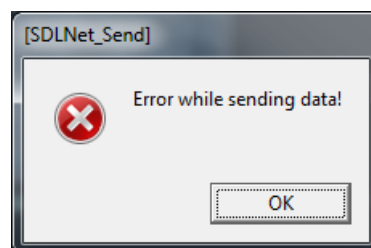
2.13. ábra. Nem egyező verziószámot jelző üzenetdoboz.

- Hiba történt az adatok fogadásakor. Ilyenkor a kliens adatokat vár a szerver felől, de azt már nem éri el. Ez történhet akkor, ha a szerver-program működése valamilyen oknál fogva leállt. Szintén előfordulhat ez a hiba, ha a hálózati kapcsolat a szerverrel megszakad. Ilyenkor sajnos az aktuális játék abbamarad és újra-csatlakozási próbálkozással lehet megbizonyosodni arról, hogy a szerver-program újra lett-e indítva.



2.14. ábra. Adat fogadásakor történő hibát jelző üzenetdoboz.

- Hiba történt az adatok küldésekor. Ilyenkor a kliens adatokat küldene a szerver felé, de azt már nem éri el. A hiba oka és megoldása az előző pontban leírtakkal azonos.



2.15. ábra. Adat küldésekor történő hibát jelző üzenetdoboz.

3. fejezet

Fejlesztői dokumentáció

3.1. Követelmény specifikáció

A szakdolgozat projektjének az elkészítése során a következőket kell megvalósítanom:

- Szerver-kliens architektúra. A közöttük lévő hálózati kapcsolat felépítése. A játékkal kapcsolatos adatok továbbíthatósága. A kapcsolat megszakadásából fakadó hibák kezelése.
- A *Torpedó*[1] háttér-logikája. A játék szempontjából autoritatív szerver. Játékosok közötti hajó-adatok elrejtése.
- A játék valós idejű kirajzolása a kliens programban. Szöveg kirajzolása a grafikus felületre.
- Csatahajók lövedékének az animációja.
- **2D**-s felhasználói felület. Ezt követően a főmenü felépítése. Megjelenítési- és hangerő beállításokat kezelő almenü felépítése. Beállítások elmenthetősége fájlba, illetve azok betöltése a kliens program indulásakor.
- Zenék, illetve hangok lejátszása.
- A **3D**-s színtér felépítése. Ennek részei: tenger, talaj, égbolt, a játékosok hajói, ágyú lövedék, a lövedékhez tartozó füst-részecskék és középen egy hegy.

- Fényszámítás a **3D**-s színtér legtöbb elemére. Átlátszóság alkalmazása a tengerre, játékmenet szövegére, a tenger- és talaj által meghatározott pálya szélére.
- Interaktív kliens. Vezérlő gombok létrehozása. A menü gombjaira, illetve a játéktér mezőire való kattintások kezelése.
- Hibák jelzése annak tudatában, hogy a kliens program egy Windows ablakban, a szerver pedig egy konzolos ablakban kell fusson.
- Szerver és kliens közötti verzió egyezés ellenőrzése.
- Vesztes játékos számára, a győztes életben maradt hajó-adatainak átküldése, megjelenítés céljából.

3.2. Felhasznált technológiák

3.2.1. Programozási nyelv

Programozási nyelv tekintetében a *C++*[5] mellett döntök. A *C++* nyelvet felhasználva hatékony és gyors kódot tudok írni, ami valós idejű megjelenítéssel rendelkező programok esetében nagyon fontos szempont. Hátránya viszont, hogy nincs automatikus memória-felszabadítás a *heap*-en, a pointerok könnyen mutathatnak olyan memóriaterületre, amelyet az operációs rendszer nem(vagy már nem) a program számára tart fenn. Az írandó program kódjának mérete és az egyediesíthetőség arányát tekintve a programozási nyelveket csoportosítani szokás. Ebben a csoportosításban a *C++* középszintű nyelvnek számít és a *C* nyelv felett, illetve a *JAVA* nyelv alatt helyezkedik el.



3.1. ábra. C++ logója (https://isocpp.org/assets/images/cpp_logo.png)

Alternatívaként felmerül a *JAVA* programozási nyelv, mivel a *C++*[5] előbb említett hátrányait ez orvosolja. *JAVA*-nál fontosabb szempont hogy fusson a

program, mint az hogy milyen gyorsan fut. Ebből eredően a nyelv fordítói is szigorúbbak és több a futás idejű hiba-ellenőrzés. Ezek viszont a program futását értelemszerűen lassítják, ezért is nem találom a *JAVA*-t jó alternatívának játékfejlesztés esetén.

3.2.2. Fejlesztői környezet

A program elkészítésére a *Visual Studio 2015*[6] és 2019-et használok fel. Ez saját, *Microsoft Visual C++(MSVC)* néven ismert fordítóval rendelkezik. Egy ekkora terjedelmű projekt elkészítésére szükségem van egy olyan fejlesztői környezetre, amely összefoglalja és kényelmessé teszi egy program elkészítését. A *Visual Studio*-t ilyennek találom, mivel be lehet benne állítani az **MSVC** fordítót, rendelkezik saját kódszerkesztő felülettel és kiírja a fordító hibáit. Továbbá, futásidőben fellépő hiba esetén blokkolja a program folyamatát, hogy mielőtt az bezárulna, végig tudjam nézni a függvény-hívási stack-et. Ezen felül, egy kód-fájlban lévő függvénynek meg tudom keresni az egész projektben lévő előfordulását. Hátránya, hogy elég sokat foglal, illetve hogy ha nem tanulmányi célokra használok, akkor fizetős. Szerencsére, a telepítéskor kiválasztható hogy pontosan mely részeit tervezem felrakni, viszont a sok telepítendő csomag között nem igazán könnyű eligazodni.

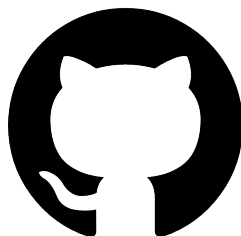


3.2. ábra. Visual Studio logója (<https://visualstudio.microsoft.com/wp-content/uploads/2021/10/Product-Icon.svg>)

Alternatívaként felmerül a *Notepad++* vagy a *Visual Studio Code* szövegszerkesztők *MinGW* fordítóval való használata. Ezek mivel csak szövegszerkesztők, ezért nem lehet bennük a fordítót konfigurálni, ami már önmagában egy elég nagy probléma. Másik alternatíva az *Eclipse C++*, ami már ténylegesen hívható fejlesztői környezetnek. Ehhez is kell külön telepíteni a *MinGW* fordítót is, hogy használható legyen. Viszont ha ez megvan, akkor már megközelíti a *Visual Studio 2015*[6] által nyújtott lehetőségeket, de még nem éri el.

3.2.3. Verzió-kezelés

Verzió-kezeléshez a *GitHub*[7] nevezetű internetes hosting szolgáltatást használom, illetve a hozzá tartozó *GitHub Desktop* szoftvert. Ennek segítségével verziókra bontva elmenthető a kód és az ahhoz tartozó külsős fájlok online. Ez történhet privát- vagy nyilvános adattárba. Megtekinthetők a kódfájlokban történő módosítások két verzió között. Továbbá elvethetők a módosítások. Ha nincs internetes kapcsolat, akkor is elő lehet készíteni a módosítások feltöltését későbbre. Az asztali szoftverjük előnye, hogy nem kell a weboldalt vagy a parancssoros *Git* programot használni. Az előbbinél hiányosak a funkciók, az utóbbinak megkörülményes a használata.



3.3. ábra. GitHub logója (<https://cdn-icons-png.flaticon.com/512/25/25231.png>)

3.2.4. Rajzoló API

A kliens ablak felületének a megrajzolásához az *OpenGL*[8] 4.6 alkalmazásprogramozási interfészt(**API**) választom. Ez közvetlenül a videokártya illesztőprogramjával áll kapcsolatban. Ezáltal képes vagyok programozottan, közvetlen rajzolósi hívásokat kérni a videokártya feldolgozó egysége(**GPU**) felé, ezáltal *hardveres gyorsítással*[9] történik a kívánt kép kirajzolása. Előnye, hogy platform független és sok programozási nyelvvel használható. Hátránya, hogy a 4.6-os verzió 2017-ben jött ki és nem tervezik a fejlesztését a továbbiakban.



3.4. ábra. OpenGL logója (https://www.opengl.org/img/opengl_logo.jpg)

Alternatívaként felmerül a *Vulkan*- vagy a *DirectX* **API** használata. Az előbbi modernebb, mint az *OpenGL*[8], de nehezebb a használata. Az utóbbi pedig csak Windows operációs rendszeren működik.

3.2.5. Ablak- és input-kezelés

Az *SDL*[10] 2.0.22 multimédiás fejlesztői könyvtárat választom a Torpedó-játék kliens ablakának a lekéréséhez az operációs rendszertől. Továbbá, ez végzi az ablak-, illetve a bemenettel kapcsolatos események kezelését. Például, a menüben való kattintást nem lehetne megvalósítani egyszerű *C++*[5] standard által biztosított lehetőségekkel. Az *SDL* könyvtár "*SDL_ShowSimpleMessageBox*" függvényének segítségével könnyen le lehet kérni egy kis üzenetdoboz ablakot is az operációs rendszertől.



3.5. ábra. SDL logója (https://www.libsdl.org/media/SDL_logo.png)

Alternatívaként a *Windows GDI API* merül fel. Ez a Windows operációs rendszer megjelenítője. Ennek hátránya, hogy alacsony szinten, lényegében az operációs rendszer függvényeit kell meghívni. Továbbá az esetleges platform függetlenség is elvész a használatával.

3.2.6. Hang lejátszás

Hangok betöltéséhez és lejátszásához az *SDL_mixer*[11] 2.0.4 könyvtárat választom, amely kiegészíti az *SDL*[10] könyvtárt, tehát csak annak meglétével használható. Ez képes egy hosszabb zene fájl lejátszására és több hang lejátszására is. A "mixer" névből fakad, hogy képes összekeverni több csatornán lévő hangsávok adatait egybe. Ezen felül a könyvtárban van néhány használható hang módosító effekt. Például, a távolság-alapú hangerő interpolációt a könyvtár "*Mix_SetDistance*" függvényével könnyen implementálni tudom.

Alternatívaként felmerül az *SDL_audio*. Ez túlságosan alacsony szinten van implementálva. Ebből fakadó probléma, hogy nincsenek definiálva csatornák, így csak egyetlen hangsávon lehet hangot lejátszani. Ennél nagyobb probléma vele, hogy nem párhuzamosítja önmagát. Tehát vagy beletörődne, hogy a hang lejátszása alatt blokkolódik a program futása vagy rám maradna a párhuzamosítás implementálása.

Másik alternatíva az *OpenAL*. Ennek érdekessége, hogy ez nem egy könyvtár, hanem egy **API**. Előnye, hogy ezzel lehet belenyúlni a hang lejátszási folyamatába a legjobban, ezáltal nagyon jó hangminőséget előállítva. Hátránya, hogy ez is alacsony szinten történik, még az **API** működésre bírása se triviális.

3.2.7. Szöveg kirajzolása

A szöveg kiírása a Windowsos ablak felületére nem egy triviális probléma, ezért is hívom "szöveg kirajzolás"-nak. Ennek oka, hogy a Windows képes kirajzolni és kezelni a betűket a saját megjelenítő **API**-ja által kezelt ablakokba, viszont itt nekem csak egy vászonra hasonlítható felületem van, amit az *SDL*[10] lekér az operációs rendszertől. Tehát, célszerű ilyenkor egy szöveg-kirajzolásra alkalmas kiegészítő használata. Én erre a *SDL_ttf*[12] 2.20-t választom.

Alternatívaként felmerül az *ImGui* könyvtár használata, amellyel nem csak a szöveg kirajzolást, hanem egy egész **2D**-s felhasználói felületet könnyebben össze lehet dobni. Viszont az előállítható felület kinézete nekem nem tetszik és tudomásom szerint nincs is mód rajta szépíteni.

3.2.8. Hálózati kapcsolat

A szerver és kliensek közötti kapcsolatokat az *SDL_net*[13] 2.0.1 könyvtár segítségével valósítom meg. Hogy pontosan hogyan, azt a 3.4.5 alfejezetben fejtem ki. Ez a könyvtár egy kiegészítője az *SDL*[10] könyvtárnak, önállóan nem működik.

3.2.9. Nvidia Nsight

A grafikai kirajzolás elemzésére az *NVIDIA Nsight*[14] nevezetű fejlesztői eszközt használom. Ez képes valamilyen grafikus **API**-t használó, grafikai megjelenítéssel rendelkező szoftverbe beleszúrni magát. Így, megjelenik a program felületén az *Nsight* képernyő fedvénye és használhatóvá válnak a funkciói.

Ez a program képes egy pillanatképet csinálni a megjelenítésről és egyben blokkolni a program további futását. Majd a pillanatkép kirajzolását végig lehet követni rajzolási hívásonként. Tehát, minden kirajzolt objektum rajzolási folyamatát egyesével végig lehet követni. A szakdolgozat fejlesztése alatt én pont erre a célra használom.

3.3. Elméleti háttér

3.3.1. Vertex Buffer

Inkrementális képszintézisnek[15] hívják azt a rajzolási formát, amelyet a Torpedó-játék kliens programjában alkalmaztam. Ennek az a lényege, hogy a **3D**-s térben lévő modelleket kisebb poligonokból építem fel. Ennek a poligonnak érdemes a háromszöget választani, mivel a 3 csúcspontja által definiált háromszög mindig meghatároz egy egyedi síkot(lásd [16] hivatkozás 5. oldala). 4 csúcspont által definiált téglalapra ez az állítás viszont nem mindig igaz.

Tehát a háromszög csúcspontjait az XYZ tengelyeken meghatározott pozíciójuk alapján definiálom. Ezeket a csúcspontokat szokás *Vertex*-nek[17] hívni. Az *OpenGL*[8] segítségével, ha a háromszög 3 vertexét órával ellentétes sorrendbe átadom a **GPU** memóriájába, akkor az képes lesz elvégezni a megfelelő rajzolást. Az *OpenGL* "glFrontFace" függvényével meg lehet fordítani a megadási sorrendet, de én maradok az alapértelmezettnél.

Egy komplex modell általában nagyon sok vertexből áll. Mivel a **GPU** memóriájába relatív lassú az adatok átküldése, ezért célszerű nem egyesével át küldeni a vertexeket. Jobb megoldás csoportosítani őket egy tömbbe és átküldeni az egészet egyben. A **GPU**-n erre a célra szolgáló tömböt hívják *Vertex Buffer*-nek[18].

A *Vertex Buffer* két részből áll: *Vertex Array Object*(**VAO**) és *Vertex Buffer Object*(**VBO**). A **VBO** maga az adat tömb. A **VAO** pedig meghatározza, hogy pontosan hogyan helyezkednek el az adatok a **VBO**-ban.

3.3.2. Shader program

Shader program-nak[19] nevezik azt a programot, amely a **GPU**-n fordul le, ott linkelődik és ott is fut le a bejövő input adatokra, párhuzamosított módon. Az *OpenGL*[8] segítségével lehet elvégezni a fordítást(a "glCompileShader" függvénnyel) és a linkelést(a "glLinkProgram" függvénnyel). A *shader program* értelme, hogy a **GPU** rajzolási folyamatába programozott módon bele lehessen szólni. Az *OpenGL*-hez tartozó shader programozási nyelv neve *OpenGL Shading Language*(GLSL)[20].

A *GLSL* nyelv leginkább a *C* programozási nyelvre hasonlít. Viszont, ki van egészítve számítógépes grafikához köthető típusokkal. Ilyen például, a "vec3", ami a 3 pontból álló vektor, vagy a "mat4", ami a "4x4"-es mátrix típusa. Tartalmaz továbbá "in" és "out" kulcsszavakat, amelyek rendre jelzik egy változóra, hogy az az adott shader fázisban bemeneti vagy kimeneti adat. Tartalmaz egy úgynevezett "uniform" kulcsszót, amely lehetővé teszi a shader program futása előtt a változó értékadását vagy módosítását. Tartalmaz textúra-mintavételezéshez köthető típusokat, ilyen például a "sampler2D".

A Torpedó-játék kliensében két fajta shader kódot használok: *Vertex shader*[21]("vert" kiterjesztéssel rendelkező fájlok) és *Fragmens shader*[22]("frag" kiterjesztés). A vertex shader input adatai a vertex bufferből bejövő adatok. Ennek a kódja fut le az összes vertexre párhuzamosított módon. A számításokat követően, a "gl_Position" beégetett változó végső értéke határozza meg az adott vertex végső pozícióját. Ez egy 4 elemű vektor, melynek első 3 eleme az XYZ tengely által meghatározott pozíció, a 4. elem pedig a *homogén koordináta*[23]. Ennek segítségével készül el a **3D** vetítése **2D**-be. Az "out" kimeneti értékek pedig a fragmens shader bemeneti értékei lesznek. A kimeneti vertex értékek itt *normalizált eszközkoordinátákban*[24] kell hogy legyenek, -1 és 1 között, mert az ezen kívüli értékek levágódnak vagy nem lesznek a megjelenítés részei. Mire a fragmens shader lefutási fázisába ér, addigra már elkészül egy **2D**-s kép a **3D**-s színtérről és itt már ennek a képnek a fragmenseivel dolgozok. Egy fragmens értéke általában ekvivalens egy pixelével, kivéve mikor a megrajzolt felület mérete különbözik a megjelenítési felület méretétől. A fragmens shader kódja párhuzamosított módon fut le annyiszor, ahány fragmens van. Kimeneti értéke egy 4 elemű vektor(a "fs_out_col" változó), amely tartalmazza a fragmens RGB(piros-zöld-kék) színét és az alfa értékét 0 és 1 közötti tartományban. Az alfa értéket általában átlátszóság számításához használják, de lehet másra is. Például a *3D Picking*[25] módszer megvalósításához ki lehet menteni egy objektum indexét az alfa értékbe.

A shader kódok párhuzamosított lefutása jelentősen gyorsít a kirajzolás folyamatán. A **GPU** felhasználásával való kirajzolást hívják *hardveresen gyorsított rajzolás*-nak[9]. Ez persze a **GPU** működési jellegéből fakadóan csak akkor gyors igazán, hogyha kerülöm a túlzott elágazások vagy ciklusokból való kitörések használatát a shader kódban.

A Torpedó-játékban használt különböző shader programok célja a következő:

default: Az alap *kép-buffer*-hez[26] egy alap shader. Ez az egyedi kép-bufferhez megrajzolt textúrát rajzolja ki az ablak felületére.

dirLight: Textúrákat rak az objektumokra, majd kiszámolja a *Phong*[27] típusú fényszámítást egy irány fényforrásra és egy megvilágítandó objektumra. A programban ilyen objektumok például a talaj, hajók és hegy. Továbbá a színtér szélén lévő talajra átlátszóságot számít az égbolttal.

menu: A menü által használt shader program. Előrajzolási fázisban az alfa értékbe kieszedi az objektum indexét a *3D Picking*-hez[25]. Tényleges rajzolási fázisban pedig megfelelően feltextúrázza az adott menü elemét. Ezen felül ez a shader is színez át egy gombot, ha felette tartom az egérmutatót.

options: A beállításokat tartalmazó almenü shader programja. Nagybán hasonlít a "menu" shader programhoz. Kivétel, hogy mivel ez az almenü tartalmaz textúra nélküli elemeket, ezért az itt le van kezelve. Továbbá ez nem használható előrajzoláshoz.

projectile: Csak egy egyszerű alap shader a csatahajó lövedékének a rajzolásához. Külön shader program, mivel a lövedéknek nincs textúrája.

ptOutline: A játék-mezőhöz köthető. Ez végzi el az előrajzolást, a játékmező objektumokhoz indexet rendelve. Továbbá ez végzi el a játékmezők körvonalainak a megfelelő színezését, az állapotuktól függően.

seatile: A tenger- és játékmezők rajzolásához van használva. *Phong*[27] típusú, de spekuláris fénytől mentes fényszámítást alkalmaz. Továbbá a színtér szélén lévő tengerre átlátszóságot számít a égbolttal.

skybox: Itt végzem el a égbolt "végtelenbe" való kitolását. Ez annyit jelent, hogy mindig megjelenítem, de mindig a többi **3D**-s objektum mögött. Ehhez a vertexek pozíciói a Z tengelyen a *homogén koordináta*[23] értékét kapják meg. Továbbá a fragmens shaderbe egy több lapból álló *Cubemap*[28] textúrájából mintavételezek.

text: A játékmenetben lévő szöveg-mezőhöz köthető shader program. A vertex shader kód a szöveg szélessége és a sorok száma alapján határozza meg az szövegdoboz csúcspontjainak a pozícióját. A fragmens shader kód pedig először

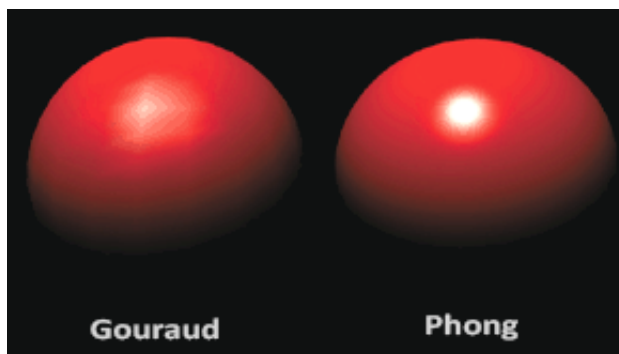
is a szöveget tartalmazó textúrát illeszti rá. Ezt követően átlátszóvá teszi teszi a szövegdoboz azon részét ahol nincs szöveg.

3.3.3. Phong megvilágítási modell irány fényforrással

A kliens program játékmenetében *Phong megvilágítási modell*-t[27] használok, irány fényforrással, így szimulálva a nap világítását a **3D**-s objektumokra. A modell 3 részből áll össze: ambiens-, diffúz- és spekuláris megvilágítás. Az ambiens egy minimális megvilágítási érték, hogy teljes sötétségben se legyenek a tárgyak teljesen feketék. Diffúz megvilágításnál minél kisebb a fény iránya és az objektum vertexének a normálvektora között bezárt szög, annál erősebb lesz a fény. A spekuláris fény egy objektum csillogását szimulálja. Itt a fényforrás objektumon való visszaverődése és a kamera nézeti iránya között bezárt szög függvényében nő a fényerő. Tartalmaz egy fényességi erőt is, amelynek növekedésének függvényében minél kisebb pontban gyűlik össze a fény.

Az irány fényforrás a legegyszerűbben implementálható fényforrási típus. Ennek nincs pozíciója, csak iránya. Kifejezetten jó az irány fényforrásnak a szimulációja olyan esetben, amikor valami végtelenül távoli pontot feltételezünk fényforrásnak. Ilyen például a Torpedó-játék esetében a nap.

A *Phong megvilágítási modell*-t a fragmens shaderben kell leimplementálni, mivel minden egyes fragmensre számol megvilágítást. A vertex shaderben implementálva gyorsabb eredményt kapnánk, viszont az interpoláció megszűnésével csúnyább eredmény születne. Az olyan megvalósítást *Gouraud*[29] megvilágítási modellnek nevezik. A különbség látható a 3.6 ábrán.



3.6. ábra. Gouraud és Phong megvilágítási modell közti különbség (https://learnopengl.com/img/lighting/basic_lighting_gouraud.png)

3.4. Megvalósítás

3.4.1. 3D Picking

3D Picking[25] módszernek nevezik azt, amikor a **2D**-be le vetített **3D**-s szintéren meghatározom, hogy melyik objektumra mutatok az egérrel. Mivel a rajzolási folyamat végén csak egy színes kép marad, ezért a megoldás nem triviális. A Torpedó-játék kliens programja ezt a módszert használja például a menü elemekre vagy a játékmezőkre való kattintáskor.

A hivatkozásban leírtak alapján, a módszer egy előrajzolási fázissal kezdődik. Ebben a fázisban a rajzolás végére indexeket rendelek a kirajzolt objektumoknak a fragmens shaderben. Ez után az eredmény képéből kiolvasom azt a pixel adatot, amire az egér mutat. A pixel adat alfa értékében lesz az objektum indexe. Ez alapján már tudom is, hogy melyik objektumra kell egy esetleges kattintási esemény-kezelést végrehajtani.

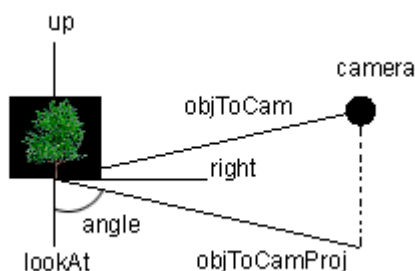
További lehetőség, hogy végül a végső kirajzolási fázisban átadom ezt az indexet a fragmens shadernek, amely egyenlőséget fog ellenőrizni az objektumok indexével. Ha egyenlőséget talál, akkor lehetőségem van másképpen kirajzolni az adott objektumot. A Torpedó-játékban például más színű az a menü gombja, amire éppen mutatok az egérrel.

3.4.2. Cylindrical Billboard

Cylindrical Billboard[30] módszernek nevezik azt, amikor egy függőlegesen álló **2D**-s téglalapot mindig úgy forgatok az Y tengely mentén, hogy az a kamera irányába nézzen. Ezzel spórolni lehet gépi erőforrást, mivel nem egy **3D**-s modellel dolgozok, viszont nem fog eltűnni a téglalap, csak ha alulról vagy felülről tekintek rá. A Torpedó-játék kliens programjának a játékmenet részében ezzel a módszerrel fordulnak a lövedék részecskéi mindig a kamera irányába.

A hivatkozásban leírtak alapján, először is meghatározom a téglalap pozíciójától a kamera irányába néző irányvektort az XZ tengely által definiált síkra vetítve. Ennek eredményét skalárisan összeszorozom a téglalap nézeti irányával, hogy megkapjam a bezárt szöget. Kereszt szorzatot is végzek, hogy megkapjam a téglalap megfelelő forgási tengelyét. Végül a bezárt szög arcus koszinuszával elvégzem a

forgatást a kiszámolt forgástengely alapján. A 3.7 ábrán látható, hogyan kell ezt elképzelni.



3.7. ábra. Cylindrical Billboarding módszer szemléltetve
(<http://www.lighthouse3d.com/opengl/billboarding/image1.gif>)

3.4.3. Átlátszóság

A Torpedó-játék kliensében kétféle átlátszóság van megvalósítva. Az első a teljes átlátszóság, amikor is egy kirajzolandó objektum bizonyos részeit teljesen átlátszóvá teszem. Ez a *fragments shader*-ben történik. Ott, egy bizonyos ellenőrzés alapján eldönthető, hogy mely *fragmente*-ket tartom meg, illetve dobom el. Az eldobott *fragmente*-k helyén ott marad a mögötte lévő objektum *fragmente*-s, ezáltal megvalósul a teljes átlátszóság. Ilyen történik például a játékmenetben kirajzolt szövegdoboz esetén. Ott, az előre megadott háttérszínnel rendelkező *fragmente*-ket eldobom és csak a fehér színű szöveget tartalmazó részt tartom meg.

A másik típusú átlátszóság a részleges átlátszóság, amikor valamilyen arányban keverem két egymást fedő objektum színeit. Ilyen történik például a tenger esetében. Szerencsére, az *OpenGL*[8] ezt képes automatikusan elvégeztetni számomra, ha bekapcsolom a "GL_BLENDING" beállítását. Ekkor, a kirajzolt objektum alfa értékét veszi alapul, ami egy 0 és 1 közötti szám és az alapján arányosítja az objektum és az el fedett objektumok *fragmente*-inek színét.

3.4.4. Fájlkezelés

A Torpedó-játék kliens programja tartalmaz külső erőforrás fájlokat. Ezek a "*TorpedoJatekClient/Resources*" mappában találhatóak.

Képek: A programban a textúrák képek segítségével készülnek. Ezen "bmp" és "jpg" kiterjesztésű kép-fájlok betöltését az *SDL_image*[31] 2.0.1 könyvtár kezeli.

Zenék és hangok: Az "ogg" formátumú zenéket és "wav" formátumú hangeffektek betöltését és lejátszását az *SDL_mixer*[11] könyvtár kezeli.

Betűtípusok: A "ttf" kiterjesztésű betűtípus-fájlokat az *SDL_ttf*[12] könyvtár tölti be és használja fel a későbbi szövegek rajzolásához.

Beállítások: Az "options.cfg" nevű, kliens program beállításait tartalmazó fájl a program a *C++*[5] standard szövegkezelő függvényeivel hozza létre, írja felül vagy olvassa be.

- Ha a program indulásakor nem létezik, akkor a programba beégetett alapértékekkel létrehoz egyet.
- Ha már létezik, akkor beolvassa, de a megfelelő változók értékadása előtt még ellenőrzi, hogy nem lett-e külsőleg korrumpálva. Ha igen, akkor szintén felülírja a fájlt az alap-értékekkel.
- A beállítások almenüben, mentés esetén kimentí a megfelelő formában az adatokat a fájlba.

3.4.5. Hálózati adatfolyam

A szerver és kliens programok között *Transmission Control Protocol*(**TCP**) típusú hálózati kapcsolat jön létre és úgynevezett "socket"-eken folyik a szerver-kliens kommunikáció. Ez az *SDL_net*[13] könyvtár segítségével valósul meg. **TCP** protokoll mellett döntöttem az **UDP**-vel szemben, mivel a projekt szemszögéből fontosabbnak tartom a garantált adatátvitelt, mint a gyorsat.

A szerver a helyi végpont **IP** címén létrehoz egy szerver típusú socketet és azt hozzá kapcsolja a megadott port számhoz. Ezt a socketet beteszi egy "socket set"-be, amin aztán hálózati aktivitást tud ellenőrizni anélkül, hogy a program futása blokkolódna. Itt a hálózati aktivitás csak egy kliens csatlakozása lehet, így az adott

klienshez egy külön socketet hoz létre, majd ezt is beteszi a "socket set"-be. Ez így biztosít egy párhuzamosított kiszolgálást is, mivel a kliensekhez el tárolja azt is, hogy mely állapotában tart a kiszolgálásuk. Ezért akár egyszerre képes a szerver lekezelni a második kliens csatlakozását és az első klienssel történő adatátvitelt. Ha a szerver már kapcsolatban áll két klienssel, de egy harmadik is csatlakozna, akkor vele létrehoz egy ideiglenes kapcsolatot csak addig, amíg tájékoztató üzenetet küld, hogy tele a szerver.

A szerver és kliens közötti adatátvitel az "SDLNet_TCP_Send" és "SDLNet_TCP_Recv" függvények segítségével történik. A "SDLNet_TCP_Send" paramétere egy kapcsolati socket, egy küldésre szánt adatra mutató pointer és egy szám érték, amellyel megadom hogy hány bájtnyi adatot akarok küldeni. A "SDLNet_TCP_Recv" paramétere egy socket, egy mutató a memóriaterületre, ahova elmentem az adatot, illetve a maximálisan tárolható bájtok száma ezen a területen.

A kliens oldalon a kapcsolat létrehozása hasonlóan történik a szerveréhez. Annyi a különbség, hogy itt nem egy szerver socketet hozok létre, hanem egy olyat amely szerver sockethez csatlakozik. Viszont ezt is ugyanúgy beteszem egy "socket set"-be, hogy ellenőrizni tudja a hálózati aktivitást a program folyamatának blokkolása nélkül.

3.4.6. Verzió egyezés ellenőrzése

Szerverre való csatlakozást követően, a szerver bontja a kapcsolatot egy klienssel, hogy ha a kliens verziószáma nem egyezik a szerverével. Ilyen esetben a szerver standard kimenetén is megjelenik egy tájékoztatás, illetve a kliens oldalon is, egy üzenet-doboz formájában. A szerver elküldi ilyenkor a kliens felé a saját verziószámát és ezt az információt tartalmazza az üzenet-doboz.

Ennek a megvalósítására készítettem a *TorpedoJatekVersion* osztályt, amely tartalmazza a verzió-számozási konvenciót. Új verzió elkészítésekor csak ennek az osztálynak az értékein kell módosítsak. Ez által beégett a verziószám a kliens és a szerver programba is egyaránt. A kapcsolódáskor, a szerver oldalon meghívódik a "CheckClientVersion" függvény, amely elvégzi a verzió ellenőrzést. Ez küldi el a választ is az eredményről a kliensnek.

Fontosnak tartom, hogy a verziókezelés folyamata lehetőleg verzió független legyen és hogy ez a folyamat induljon el legelőször a csatlakozást követően. Ezért a folyamat megvalósítását követően törekszem arra, hogy csak esetleges hibák esetén módosítsam az ezt kezelő függvényt. Ugyanis, ha túl gyakran változik egy ilyen függvény, akkor már képtelenné válhat az eredeti céljának az ellátására. Ha pedig nem ez a folyamat indulna el először, akkor az nagy eséllyel idézne elő inkonzisztens adat-átvitelt.

3.5. Felhasználói interakciók

A Torpedó-játék egy interaktív szoftver, tehát felhasználó inputokat vár el a megfelelő futás érdekében. Ezért is tartom fontosnak részletezni a program által nyújtott lehetőségeket.

Az interakciók két csoportra bonthatók:

- Azon interakciók, melyeknél egy aktor van és elvégezhetőek a szerver program használata nélkül. Ilyen például a fő menüből átlépés egy almenübe, vagy a kliens program beállításainak a módosítása.
- Azon interakciók, melyeknél több aktor van, szükséges 2 kliens program és a szerver program futtatása, illetve szükséges, hogy a kliensek sikeresen fel csatlakozzanak egy aktívan futó szerverre. Ebből fakadóan, egy ilyen interakció például egy játékos lövése az ellenfél játéktérnek egy pozíciójára.

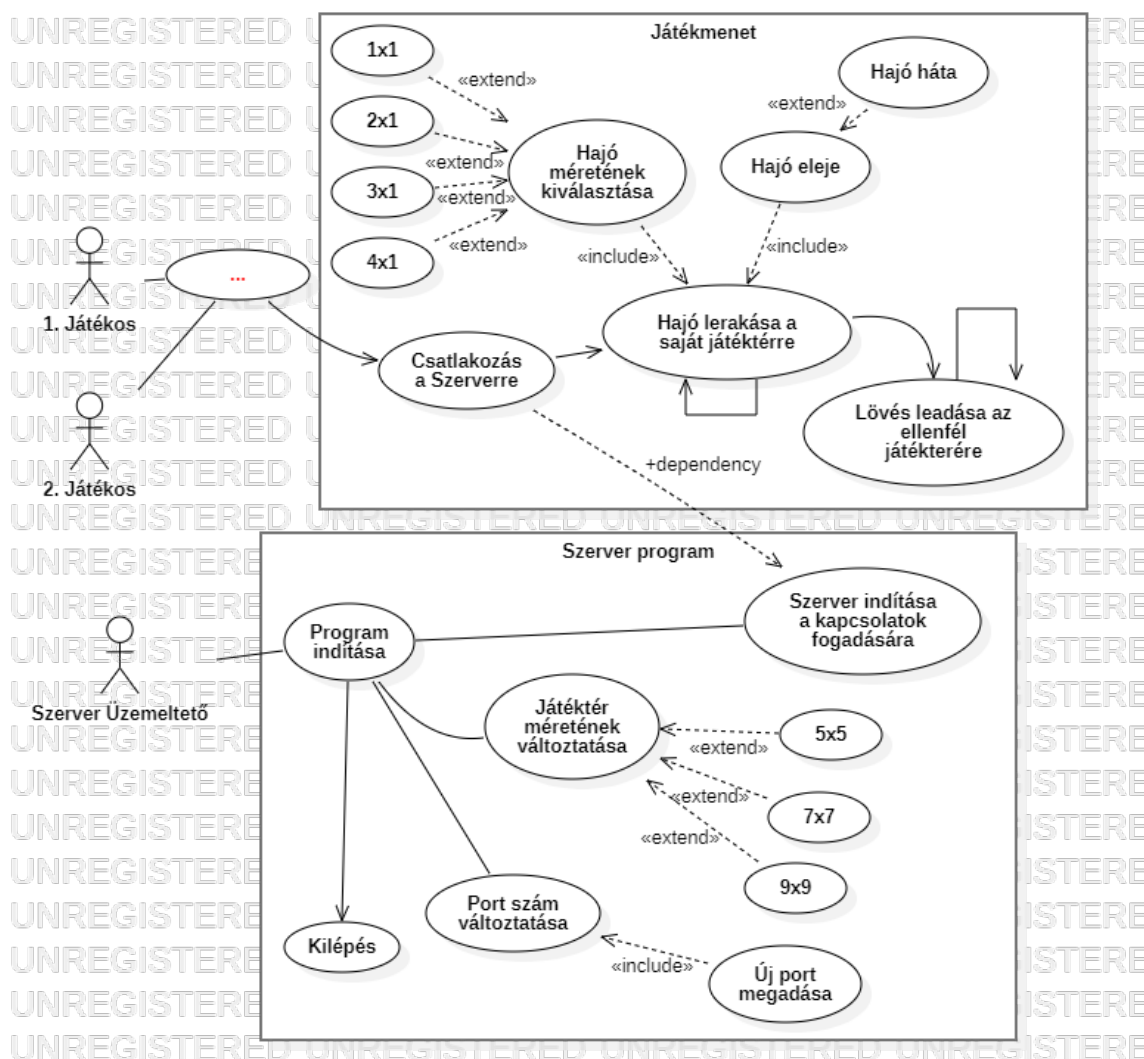
A következő két használati-eset diagram segítségével részletesebben bemutatom a lehetőségeket:

A 3.8 ábrán látható, hogy milyen lehetőségek vannak a kliens program kipróbálására, szerver megléte nélkül. Az ábrán pirossal jelölt "..." csomópont jelöli azt a részt, ahol már kell a szerver. A kliens szétbontható menü és játékmenet részekre.

- Lehetőség van felcsatlakozni egy játék-szerverre, beírva a szerver **IP** címét és a portszám adatait. Aktív kapcsolatokat váró szerver nélkül persze ez egy hibaiüzenetet követően megghiúsul. Csatlakozási próbálkozást megelőzően lehetőség van visszalépni a fő menübe.
- Elindítható a játékmenet "Debug" módban, amikor is a kliens játékmenet részese elindul, de aktív szerverre kapcsolódás nem történik. Ilyenkor a *Torpedó*[1] játék háttér-logikája sincs jelen, tehát játszani se lehet.
- Megtekinthetők és módosíthatóak a fájlba mentett beállítások, melyek a kliens program működését befolyásolják. Itt csak a zene és hangeffektek változtatása lép érvénybe azonnal, a többi beállítás érvénybe lépéséhez el kell menteni a változtatásokat a módosítást követően. Ha végeztem, akkor vissza tudok lépni a fő menübe.

- Kiléphetek.

A "Debug" módban indított játékmenetben be lehet járni a **3D-s** színteret a kamera mozgatása által és ki- vagy bekapcsolni a hangokat és a szövegkiíratást. Vissza lehet lépni a fő menübe. További extra lehetőségek is vannak, ezek a csatahajó lövésének és elsüllyedésének a grafikai tesztelése. Részletesebben kitérek a játékmenet irányíthatóságára a 2.3.4 alfejezet "Játékmenetet vezérlő gombok" részében.



3.9. ábra. Használati-eset diagram, ami a többszemélyes játékmenet interakcióit mutatja be.

A 3.9 ábrán már egy tényleges, többszemélyes játékmenet interakciói láthatóak. A pirossal jelölt "..." csomópont azt jelzi, hogy egy játékos értelemszerűen képes azon interakciók végrehajtására abban az esetben is ha amúgy létezik egy aktív

kapcsolatokat váró szerver. Fontos megjegyezni, hogy a 3.9 ábrán lévő 3 aktor szerepét akár egy személy, egy számítógépen is el tudja végezni.

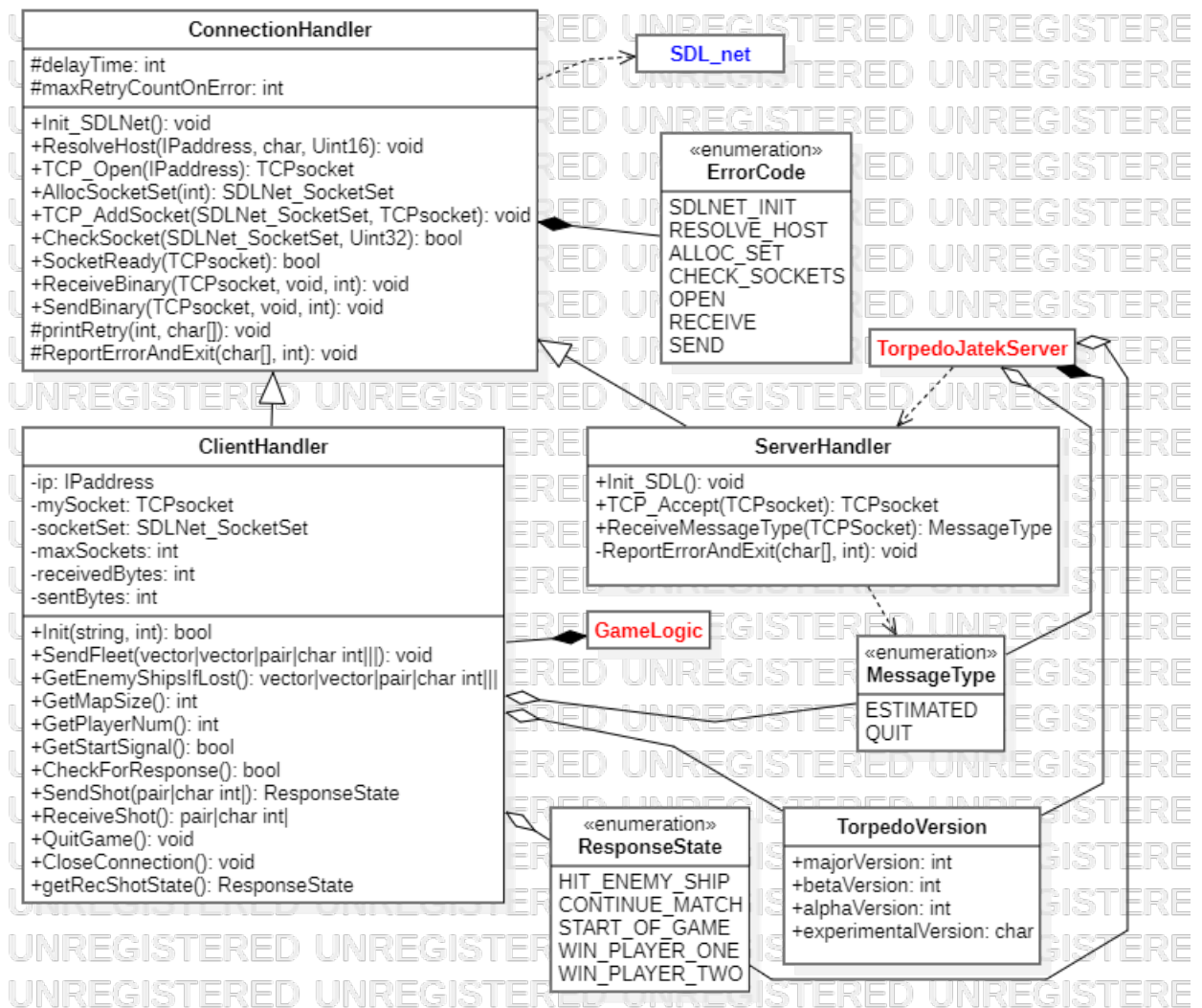
Szerver üzemeltetőként a következő lehetőségeim vannak a szerver program indítását követően:

- Elindíthatom a szervert, hogy aktív kliens kapcsolatokra várjon. Ilyenkor a program futásába további beleszólásom nem lehet és meg kell várni amíg egy játékmenet 2 játékos között véget ér. Ekkor ugyanis a program újraindul az alapértelmezett beállításokkal.
- Módosíthatom a játéktér méretét. A lehetséges méretekről és az abból fakadó hajó mennyiségekről írtam a 2.2.1 alfejezetben.
- Változtathatok, hogy mely portszámon várja a program a kliens csatlakozásokat. Ezt nem ajánlott módosítani, csak ha emiatt nem működik rendesen a kliensek csatlakozása. Viszont minden esetben a játékosok tudtára kell adjam, hogy végül melyik a használt port szám.
- Kiléphetek.

3.6. Osztályok

3.6.1. Kapcsolatokat kezelő osztályok

A szerver és kliens programok között kapcsolatokat kell létrehozni és az ezen kapcsolatokon történő adatátvitelt felügyelni.



3.10. ábra. Osztálydiagram a kapcsolatokat kezelő osztályokról.

A 3.10 ábrán látható diagramon szemléltetem az ezzel foglalkozó osztályokat. Az *SDLnet*-et[13] kék színnel jelöltem meg, mivel az egy külső header fájl, amelyet fel használok. A kliens a saját *ClientHandler* osztályán keresztül használja fel a *ConnectionHandler* osztályt, még a szerver program a *ServerHandler*-t használja fel. Viszont mivel ezen osztályok is összefüggnek, ezért tartom fontosnak ezeket kiemelni.

ConnectionHandler

Ez az osztály egy úgynevezett "wrapper"-je az *SDL_net*-nek[13]. Ez azt jelenti, hogy az *SDL_net*-ben lévő, hasonló nevű, statikus függvényeket tartalmaz. Viszont annyiban különbözik a felülírástól, hogy én ezeket kibővítem úgy, hogy

tartalmazzanak extra funkciókat. Ilyen funkciók például a kapcsolatokból fakadó hibák kezelése és kiírása. A programban ezzel foglalkozik a "printRetry" és a "ReportErrorAndExit" függvény.

A *ConnectionHandler* képes létrehozni egy socketet a "TCP_Open" függvénnyel, amely kapcsolatban áll egy szerver-sockettel, majd figyelni ezen a socketen, hogy történt-e bármilyen hálózati aktivitás, a "CheckSocket" függvénnyel. Továbbá képes egy ilyen socketen keresztül bináris adatok küldésére("SendBinary" függvény) és fogadására("ReceiveBinary" függvény).

Az osztályhoz tartozik egy *ErrorCode* nevezetű felsoroló, amelynek célja, hogy szövegesen érthetőbbé tegye a függvényekből származó hibakódokat.

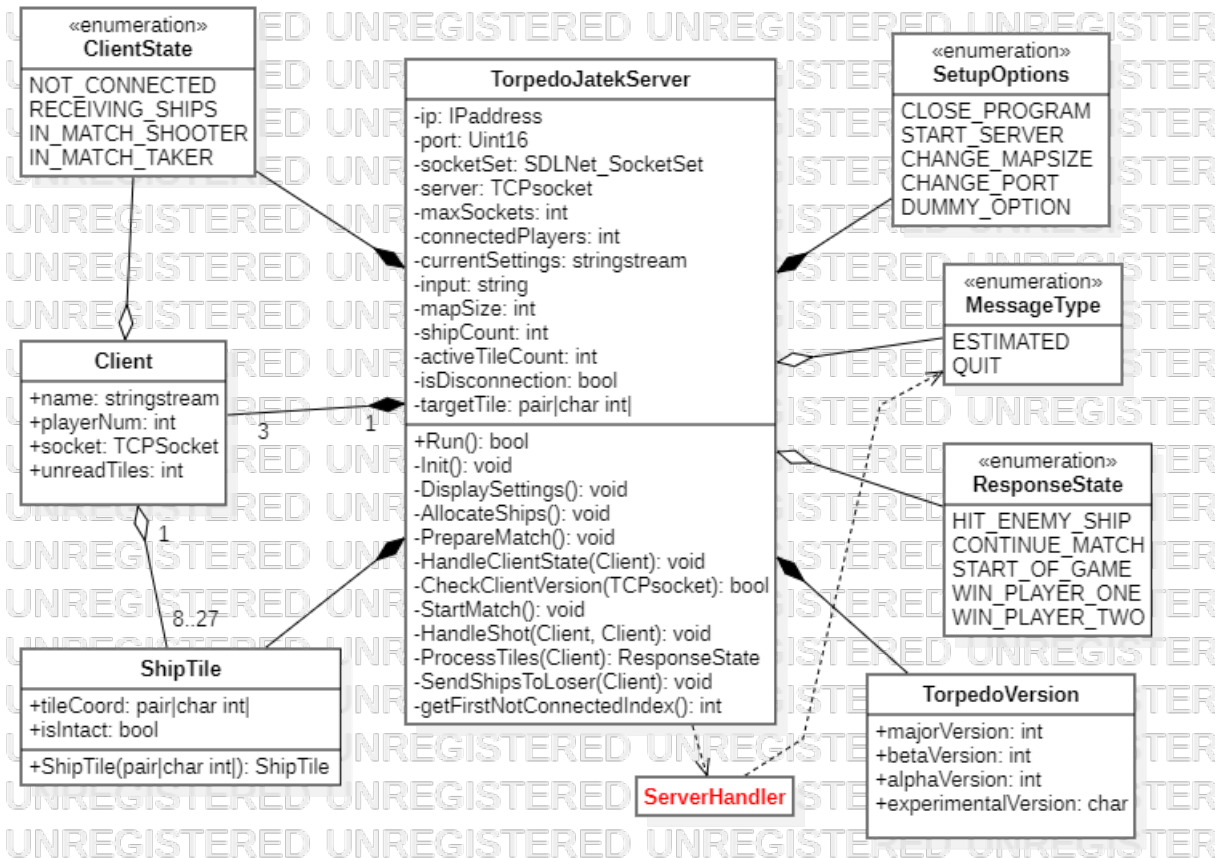
ServerHandler

Ez az osztály a *ConnectionHandler* osztály leszármazottja, amelyet a szerver program használ. Tartalmaz olyan függvényt, amely kifejezetten szerver-socketet hoz létre. Ilyen a "TCP_Accept". Ezen kívül, mivel a szerver program egy konzolban fut, a hiba-kezelésből ki kell szedni a *MessageBox*-ok(üzenetdobozok) használatát, ehhez pedig felül kell írni a "ReportErrorAndExit" függvényt.

A "ReceiveMessageType" függ egy másik fájlban lévő felsorolótól. A függvény célja, hogy ellenőrizze a kientől jövő adatot. Ez az adat lehet elvárt adat, mely a további szokványos működéshez kell vagy kilépési szándék, mely esetben a szerver programnak erre reagálni kell.

3.6.2. Szerverrel kapcsolatos osztályok

Az ellátott feladatokból fakadóan a szerver program jelentősen kevesebb osztályból épül fel. A 3.11 ábrán láthatóak ezek és a hozzájuk kapcsolódó felsorolók. Pirossal van jelölve a *ServerHandler*, mivel azt a 3.10 ábrán már kifejtettem. A szerver felhasználja annak a statikus függvényeit.



3.11. ábra. Osztálydiagram a szerver program osztályairól.

TorpedoVersion

A Torpedó-játék projekt verziószámát nyilvántartó osztály.

ShipTile

Egy hajóhoz tartozó játékmező nyilvántartó struktúra. Mely koordinátán van("tileCoord"), illetve hogy nem lett-e már kilőve("isIntact").

Client

Egy klienshez tartozó adatokat nyilvántartó struktúra a szerver oldalon. Legfontosabb elemei közé tartozik a kliens socket, amelyen el tudjuk érni. Továbbá a játéktér méretétől függően 8-tól 27-ig terjedő hajómező. Az "unreadTiles" tartja nyilván, hogy hány hajómező adatait nem kapta még meg az adott klienstől, mivel a játék csak akkor indulhat el, ha mindkét játékostól megkapta az összes szükséges hajómezőt.

TorpedoJatekServer

A szerver program fő osztálya. Fontosabb függvényei:

DisplaySettings: A program konfigurációs fázisát kezeli a megjelenítéssel, opciók módosításával és input fogadással együtt.

AllocateShips: A játéktér mérete alapján("mapSize") lefoglalja a hajómezőknek a területet a memóriában.

HandleClientState: A hajó lerakási fázisban kliens kapcsolatokra és hajómező adatokra vár. A játék kezdetét követően pedig lövési adatokra vagy kilépési szándéokra. Több kapcsolatot is képes nyilvántartani párhuzamosan a *ClientState* felsoroló miatt.

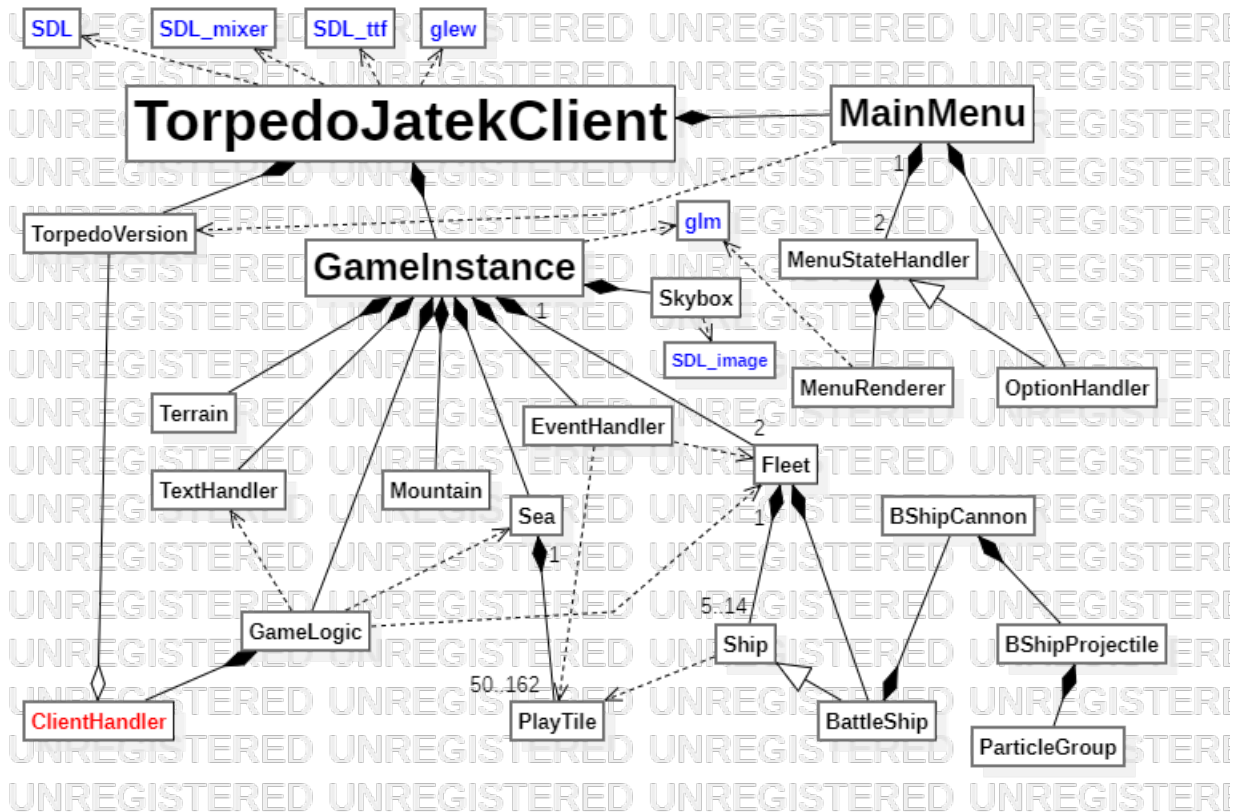
CheckClientVersion: Ellenőrzi, hogy a kliens és a szerver verziószáma azonos-e. Ha nem akkor ez tájékoztatja a klienst a szerver verziójáról("TorpedoVersion"), majd bontja vele a kapcsolatot.

HandleShot: Egy kliens lövését kezeli le a játékmenetben. Frissíti a lövést kapott kliens hajó-mezőjének az "isIntact" állapotát.

SendShipsToLoser: Egy játékmenet végén ez küldi át a vesztes kliensnek a győztes még életben maradt hajóinak az adatait, hogy az meg tudja jeleníteni azokat. Végül bontja vele is a kapcsolatot.

Része egy *SetupOptions* felsoroló osztály, amelynek objektuma a szerver konfigurációs fázisában tartja nyilván a szerver állapotát. Felhasználja a *MessageType* és *ResponseState* felsoroló osztályokat. Utóbbival egy lövést követően jelzi a kliensek felé, hogy a Torpedó játékmenete mely állapotban folytatódik tovább. Továbbá része 3 *Client* struktúrának az objektuma. A 3. kliens az ideiglenes kliens. Ennek célja, hogy ha már két játékos van a serveren és egy 3. csatlakozni próbál, akkor jelezze felé, hogy nincs már hely számára, majd bontja is a kapcsolatot.

3.6.3. Kliens osztály hierarchiája



3.12. ábra. A kliens program osztály hierarchiája.

A 3.12 ábrán látható, hogy a kliens program osztályai milyen kapcsolatban állnak egymással. A kék színnel jelzett téglalapok mutatják a külső könyvtárakkal való összefüggéseket. A piros *ClientHandler* osztály kezeli a hálózati kapcsolatot és adatforgalmat a szerverrel. A kliens tartalmaz kisegítő osztályokat is, viszont mivel azok több osztállyal is kapcsolatban állnak, ezért az ezekkel való kapcsolatokat a kifejtett osztály-diagramokon jelzem. A hely szűkösége miatt az egyes osztályok adottságait és függvényeit is alább fejtem ki.

A kliens főosztálya a *TorpedoJatekClient*. Ebből válik ki a *MainMenu*(főmenü), illetve a *GameInstance*(Játékmenet) rész. A menü megfelelő futásához nem szükséges egy szerverre való csatlakozás, ezért is van a *ClientHandler* osztály a játékmenet ágon.

A *MainMenu* osztály része két *MenuStateHandler* objektum. Ebből az egyik maga a főmenü állapota, másik a szerverre való csatlakozás almenüje.

A *GameInstance* osztályhoz két *Fleet*(flotta) objektum tartozik. Egyik a klienst indító játékosé, másik az ellenfélé. Ezek a játéktér méretétől függően tartalmaznak 5, 9, vagy 14 darab *Ship*(hajó) objektumot. Szintén e méret függvényében tartalmaz a *Sea*(Tenger) osztály 50, 98 vagy 162 játékmezőt. Itt egy osztály tartja nyilván mindkét oldal játék-mezőit külön véve, ezért is kell számolni duplán.

3.6.4. Kliens - Kisegítő osztályok

A kliens tartalmaz néhány osztályt, amelyből a kliens osztályok nagy része vagy származtat egy vagy több objektumot, vagy függnek tőlük. Az alábbiakban kifejtem melyek ezek az osztályok.

gCamera

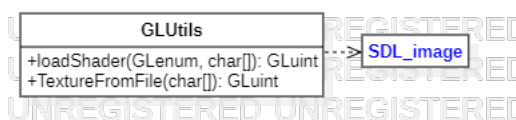


3.13. ábra. gCamera osztály-diagramja.

A *gCamera* osztály egy virtuális kamerát valósít meg, amelyen keresztül megtekinthető a **3D**-s színtér. A kliens program felhasználója képes ezt a kamerát egy adott sebességgel mozgatni, illetve forgatni. A kamera mozgástere jelenleg be van korlátozva az XZ síkon a talaj által lefedett terület hatodáig. Az Y sík pozitív irányába a hegy magasságának négyszereséig, negatív irányba a talaj szintjéig.

Az osztály a *GameInstance* részeként jön létre, mivel csak a játékmenetben van jelentősége. A **3D**-s színtérbe tartozó objektumok mind függenek a *gCamera* objektumától, mivel számít, hogy milyen szemszögből tekintünk az objektumra. Rendelkezik egy "Update" függvénnyel, amely a fő ciklus minden lefutásakor frissíti a kamera pozícióját és nézeti irányát a felhasználói interakció függvényében. E frissítés előtt a "BoundaryCheckNextFrame" függvény ellenőrzi, hogy a frissítést követően még a mozgástér belül maradna-e a kamera. A "SetProj" függvény valósítja meg a perspektívikus vetítést a **3D**-s színtérre. A *KeyboardDown*, *KeyboardUp*, *MouseMove* pedig a leszármaztatott input kezelő függvények.

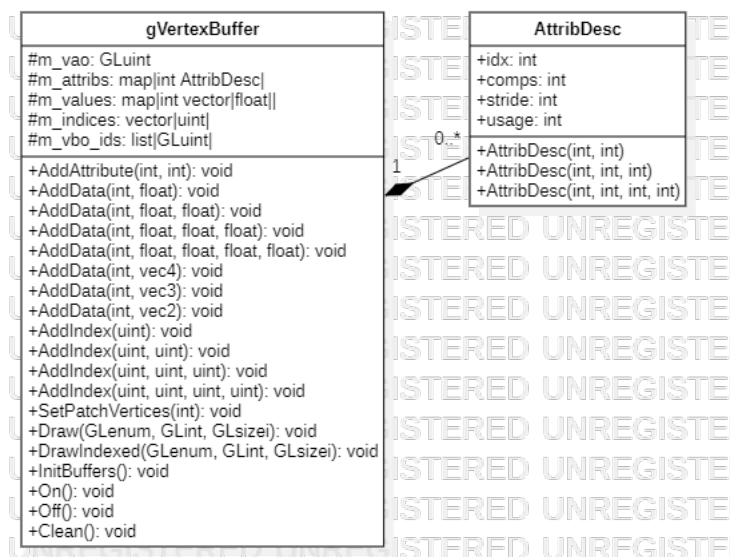
GLUtils



3.14. ábra. GLUtils osztály-diagramja.

A *GLUtils* két statikus függvényt tartalmaz. A "loadShader" shader kódot képes beolvasni és lefordítani. A "TextureFromFile" függvény egy kép-fájlból képes *OpenGL*-es[8] textúrát létrehozni. Ez a függvény emiatt függ az *SDL_image*[31] könyvtártól, mivel annak segítségével olvassa be a fájlt a kiterjesztésnek megfelelő pixel-formátumban. Minden olyan osztály függ a *GLUtils*-től, amely fájlból beolvasott textúrát használ.

gVertexBuffer



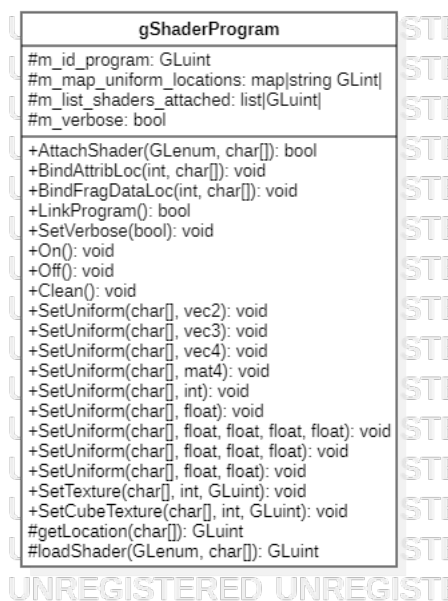
3.15. ábra. gVertexBuffer osztály-diagramja.

A *gVertexBuffer* egy **3D**-s vagy **2D**-s modell csúcspontjainak az adatait gyűjti össze egy pufferbe, hogy azt egyben át lehessen küldeni a **GPU** memóriájába a rajzoláshoz, ezzel felgyorsítva a folyamatot.

Az "AddAttribute" függvénnyel lehet megadni, hogy egy csúcspont milyen adatokból áll. Ez általában a pozíció-, szín-, normálvektor és textúra koordináta adat szokott lenni. Majd az "AddData"-val lehet hozzáadni ezeket az adatokat külön-külön. Végül az "InitBuffers" segítségével lehet inicializálni a vertex buffert, átküldve az adatokat a **GPU**-ra.

Az "On" és "Off" függvényekkel lehet váltogatni, hogy melyik buffert használom rajzoláshoz, mivel egyszerre csak egy lehet aktív. A "Draw" kezdeményezi a rajzolósi hívást a **GPU** felé, még a "DrawIndexed" az indexelt rajzolást. Az utóbbi csak akkor megy, ha index adatokat is tartalmaz a vertex buffer. Indexek segítségével kiküszöbölhetőek a duplikált vertex adatok, ezzel sokszor helyet spórolva.

gShaderProgram

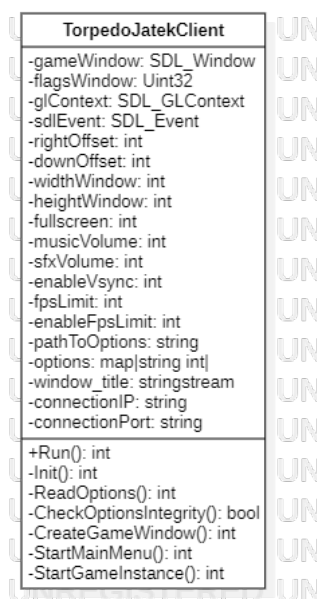


3.16. ábra. gShaderProgram osztálydiagramja.

A *gShaderProgram* a vertex- és fragmens shader kódok beolvasásával és a **GPU**-ra való lefordításával ("loadShader" függvény) és linkelésével ("LinkProgram") foglalkozik. A "SetVerbose"-al állítható, hogy hiba esetén írja-e ki a program a standard kimenetre a hibaüzenetet. Az "On" és "Off" függvényekkel lehet váltogatni több shader program között, mivel egyszerre csak egy lehet aktív. A "SetUniform" és "SetTexture" segítségével lehet rendre a shader program uniform változóinak, illetve textúra minta-vételezőinek értéket átadni a **GPU** megfelelő memória területére.

3.6.5. Klienssel kapcsolatos osztályok

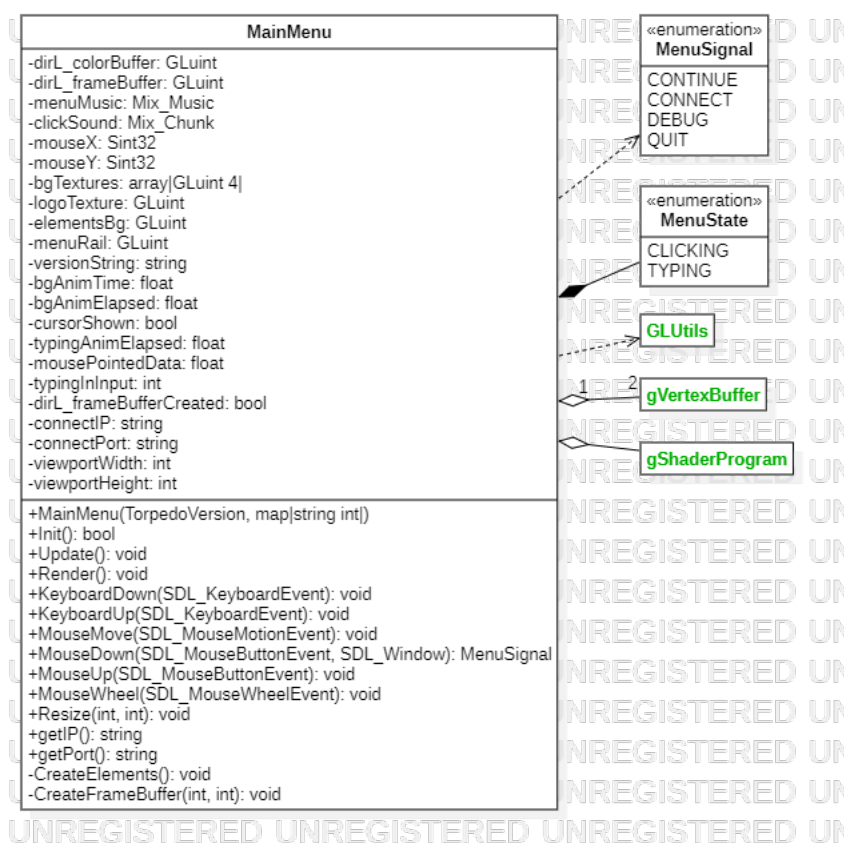
TorpedoJatekClient



3.17. ábra. TorpedoJatekClient osztálydiagramja.

A kliens program fő osztálya. Ez inicializálja a legtöbb külső könyvtárat, az *SDL*-t[10] és *OpenGL*-t[8]. Megpróbálja betölteni a beállításokat tartalmazó fájlt("ReadOptions" függvény). Ha nem létezik vagy hibás, akkor létrehoz egyet. *SDL* segítségével lekér egy ablakot az operációs rendszertől a "CreateGameWindow" függvényben. Majd felváltva hívja meg rendre a fő menüt("StartMainMenu") és a játékmenetet("StartGameInstance") kezelő osztályokat egészen a kilépésig. Szintén kezeli a főmenü és játékmenet fő ciklusát, amely meghívja az adatok frissítésével, rajzolással és input kezeléssel foglalkozó függvényeket. Szerverre való csatlakozási szándék esetén átvezeti a "connectionIP" és "connectionPort" adatokat a menüből a játékmenet részhez.

MainMenu



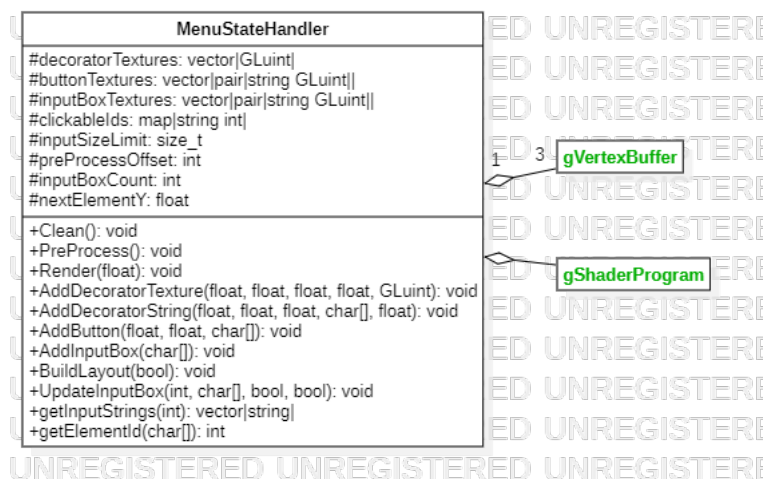
3.18. ábra. MainMenu osztály-diagramja.

A fő menüt kezelő osztály. Az "Init" függvénybe betölti a hang fájlokat, illetve elkezd lejátszani a menü zenéjét. Létrehozza a "CreateElements" függvény segítségével a főmenü és összes almenü grafikus felületeinek az elemeit. Vezérli a menü input kezeléseit azon függvényekkel, melyek *SDL-es*[10] eseményeket várnak. Kezeli a megfelelő menü kirajzolását("Render"), illetve egy előrajzolás fázisban itt valósítja meg a *3D Picking-et*[25] a menü kattintható elemeire. Az "Update" függvény számlálók segítségével figyeli, hogy mikor kell változtatni a menü háttérképén, illetve hogy a csatlakozási almenüben az input-mezőbe íráskor éppen meg kell-e jeleníteni a kurzor karaktert vagy sem.

Az osztály függ egy "MenuSignal" felsorolótól, amely nyilvántartja, hogy a menüből való kilépést követően hogyan fusson tovább a program. Tartalmaz egy "MenuState" felsorolót, amely alapján meg lehet különböztetni, hogy éppen kattintásra vár-e a menü vagy karaktereket írunk-e be egy input-mezőbe.

Mivel textúrákra van szükség, ezért függ a *GLUtils*-tól. Tartalmaz egy alap shader programot, illetve a két vertex puffert. Ebből az egyik magába foglalja az egész ablak területét a kép-bufferes rajzoláshoz, a másik a háttérképekhez kötődik.

MenuStateHandler

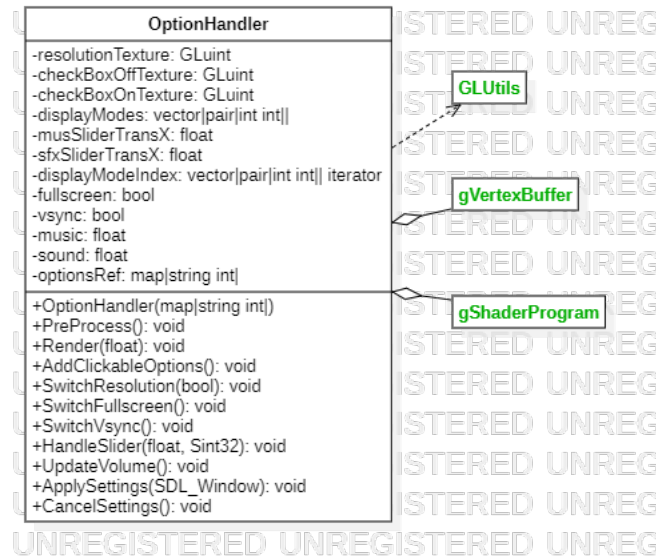


3.19. ábra. MenuStateHandler osztálydiagramja.

A főmenü és almenük állapotát kezelő osztály. Tartalmaz olyan függvényeket, amelyekkel díszítő(nem kattintható) elemeket lehet hozzáadni a menü-állapothoz. Az "AddDecoratorTexture" fájlból betöltött textúrát, az "AddDecoratorString" pedig szövegből felépített textúrát. Az "AddButton" hozza létre a kattintható gombokat, illetve az "AddInputBox" a szöveges bemeneti mezőket. Az elemek hozzáadásának végeztével a "BuildLayout" építi fel a díszítő elemek, gombok és input-mezők vertex buffereit.

Az "UpdateInputBox" függvény újra-rajzolással frissíti az input-mezőket annak függvényében, hogy milyen bemenet érkezett a felhasználtól. A "getInputString" segítségével szedhetők ki az input-mezőkbe beírt szövegek.

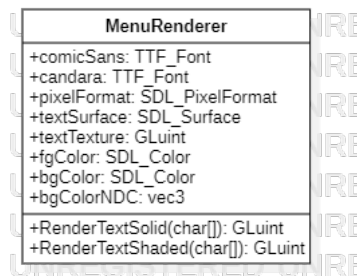
OptionHandler



3.20. ábra. OptionHandler osztály-diagramja.

A beállításokat kezelő almenü osztálya. Az "AddClickableOptions" adja hozzá az almenüben lévő extra interaktív elemeket, mint például a nyilak és a csúszkák. A "Switch-" függvények kezelik rendre a felbontás változtatás, teljes képernyő, illetve **Vsync** beállításokat. A "HandleSlider" kezeli a hangerőket állító csúszkákat, még az "UpdateVolume" frissíti a hangerőt a csúszkák új pozíciója alapján. Az "ApplySettings" érvényesíti a videó-beállításokat is, illetve elmenti az összes beállítást egy fájlba. A "CancelSettings" elveti a módosításokat és visszaállítja a hangerőket a módosítást megelőző értékekre.

MenuRenderer



3.21. ábra. MenuRenderer osztály-diagramja.

A menüben lévő szövegeket rajzoló osztály. A konstruktor a "Comic Sans" és "Candara" betűtípusokat fájlból beolvassa, "TTF_Font" típusú változókbá. Ez alapján a "Comic Sans"-t és a **CPU**-t felhasználva kirajzol szöveget egy "SDL_Surface" felületre. Majd pixel-formátumot konvertálva elkészít ebből egy *OpenGL*-es[8] textúrát a **GPU**-ra.

A "RenderTextSolid" függvény fekete háttérre rajzol fehér szöveget és bilineáris filterezést alkalmaz az eredmény textúra fragmens értékeire. A bilineáris filterezés átlagolja a fragmens és szomszéd fragmens értékeit, így meghatározva egy eredményt. A "RenderTextShaded" esetében jelenleg szürke háttérre rajzol fehér szöveget és "legközelebbi szomszéd" filterezést alkalmaz a fragmensekre. Ez a filterezés a fragmens képbeli pozíciójához legközelebbi értéket veszi eredményül.

GameInstance



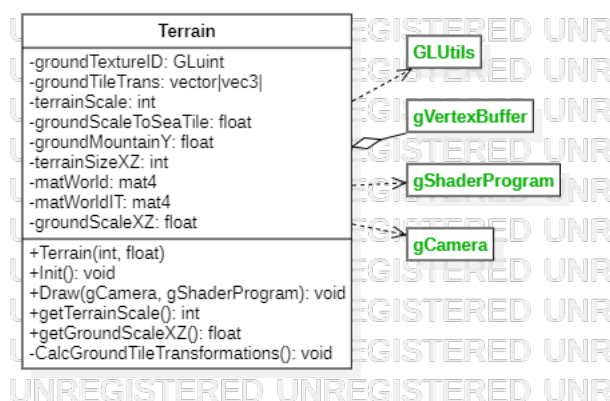
3.22. ábra. GameInstance osztálydiagramja.

A játék-menethez tartozó osztályok vezérője. Tartalmazza a billentyűzet és egérhez tartozó input kezelő függvényeket. Az "Init"-ben inicializálja a frontend-hez kapcsolódó főosztályok objektumait. Az "Update" felügyeli az

animációk lejátszási fázisát és frontend objektumok állapotváltozásait. Innen hívódik meg a "HandleGameState" is, amely a játékmenet állapotát felügyeli és változáskor esetlegesen szöveget ír ki. A "Render" intézi a teljes rajzolást, meghívva a megfelelő alosztályok függvényeit.

Tartalmaz egy *GameState* felsorolót, amely a játékmenet lehetséges állapotait jelzi. Része egy *gCamera* objektum, amelynek irányításával bejárható a **3D**-s színtér. Két shader programot is tartalmaz: az egyik az egyedi kép-pufferhez köthető("sh_default"), még a másik az olyan frontend elemek rajzolásánál van használva, amelyekre alkalmazunk *Phong*[27] típusú fényszámítást. Tartalmaz egy *gVertexBuffer* objektumot is, amely az egyedi kép-buffer kirajzolási területét fedi le, vagyis a teljes ablak vásznat.

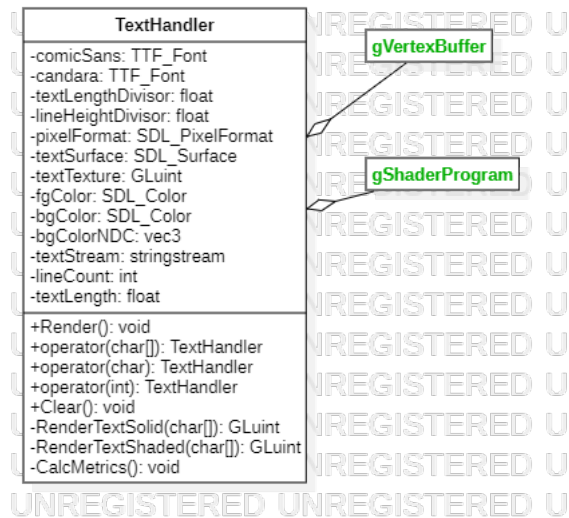
Terrain



3.23. ábra. Terrain osztály-diagramja.

A talaj osztálya. A talaj modelle "50x50" darab kis négyzet alakú földmezőből épül fel. Az "Init" függvényben van felépítve egy vertex bufferbe. A "CalcGroundTileTransformations" segítségével számítja ki, hogy az egyes földmezők milyen pozíció értéket kapnak a bufferben. A "Draw" függvény rajzolja ki végül a talajt, csak egy rajzolási hívást felhasználva. Ez így gyorsabb, mintha egyesével kéne a földmezőket elhelyezni és kirajzolni. Ezen felül helytakarékosabb is ahhoz képest, ha minden egyes földmezőnek saját vertex buffere lenne.

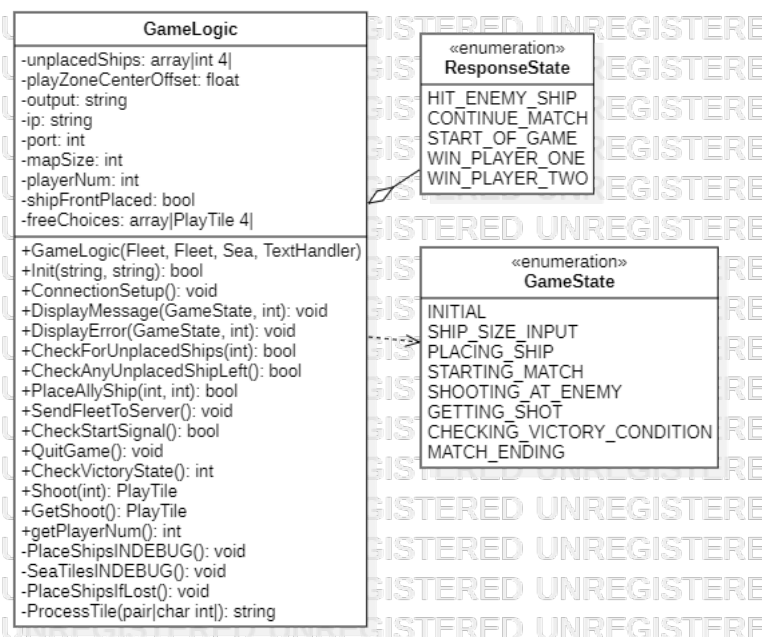
TextHandler



3.24. ábra. TextHandler osztálydiagramja.

A játékmenet állapotában kirajzolódó szöveg kezelésével foglalkozó osztály. Nagy mértékben hasonlít a *MenuRenderer* osztály működésére. Kivétel, hogy itt egy előre meghatározott pozícióra lehet kirajzolni a szöveget, illetve hogy a sortörés is meg van valósítva benne. Ezen felül még a "<" és "«" operátorok felül vannak definiálva, hogy az "std::cout"-hoz hasonló kiírást biztosítson. A "«" operátor felüldefiniálása hozzáfűzi az addig a "textStream" változóban tárolt szöveghez a paraméterben lévő szöveget. A "<" felüldefiniálása viszont teljesen felülírja az eltárolt értéket.

GameLogic



3.25. ábra. GameLogic osztály-diagramja.

A Torpedó-játék háttér-logikáját valósítja meg a kliens oldalon, összekötve ezáltal a backendet a frontendel. Ha "Debug" módban van elindítva a játékmenet, akkor a "PlaceShipsINDEBUG" függvény hívásával lerak néhány hajót a játéktérre, a kódba beégetett koordinátákra, illetve a "SeaTilesINDEBUG" átállítja néhány játéktér állapotát. Szintén a beégetett kód alapján "7x7"-es játéktérre határoz meg.

Ha szerverre kapcsolódási szándékkal indul a játékmenet, akkor kezdeményezi a szerverre való csatlakozást. Sikeres csatlakozás esetén kezdeményezi a játéktér méretének lekérdezését a szervertől, majd e méret alapján inicializálja a *Sea* és a két *Fleet* objektumot.

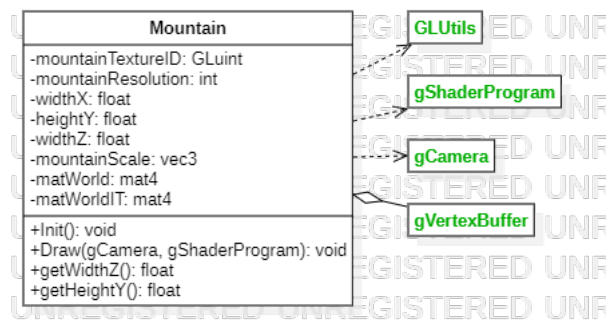
Ezt követően elindul a hajó lerakási fázis és a "DisplayMessage" segítségével jelzi a játékos felé, hogy a program felhasználói bemenetre vár. A "PlaceAllyShip" kezeli a hajók lerakását nagyrészt a *Fleet* objektum hívásain keresztül. A "CheckAnyUnplacedShipLeft" függvény ellenőrzi minden hajó lerakás után, hogy kell-e még hajót lerakni, illetve a "CheckForUnplacedShips" ellenőrzi kifejezett hajóméretre, hogy abból kell-e még lerakni. Ha a játékos az összes hajót lerakta, akkor a "SendFleetToServer" kezdeményezi a hajó adatok küldését a szerver felé.

Ezt követően a "CheckStartSignal" függvény figyeli a kliens oldali **TCP** socketet, hogy jelezte-e már a szerver a játék elindulását.

A következő fázis a lövés fázis. Itt felváltva hívódik meg a "Shoot", amely a játékos lövését kezeli, illetve a "GetShoot", amely az ellenfél lövésének a hatását. Minden lövés után a "CheckVictoryState" függvény nézi, hogy a szervertől jövő adat alapján nem nyert-e valaki. Ha az adott játékos veszít, akkor a "PlaceShipsIfLost" kezdeményezi még a szervertől a győztes játékos életben maradt hajóit és lerakja azokat az ellenfél játék-terére.

Az osztály tartalmaz egy *ResponseState* felsorolást, amely a lehetséges szerver válaszok olvashatóbb formáját jelzi. Ezen kívül függ egy *GameState* felsorolótól, amely a játékmenet lehetséges állapotait jelzi.

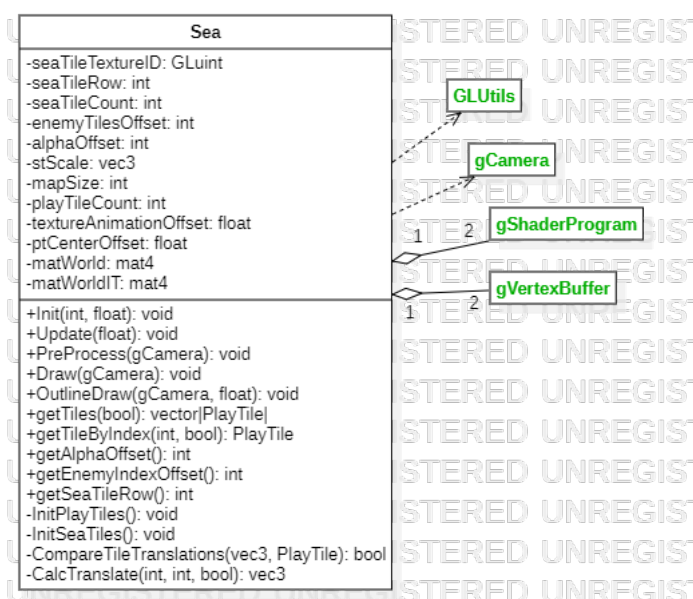
Mountain



3.26. ábra. Mountain osztály-diagramja.

A hegy osztálya. Az "Init"-ben a modell egy adott tartományra leszűkített kétismeretlenes függvény alapján van felépítve a vertex pufferbe. Majd ez kerül kirajzolásra a "Draw" segítségével.

Sea



3.27. ábra. Sea osztály-diagramja.

A tengert és a játék-tereket kezelő osztály. Az "InitPlayTiles" függvény inicializálja a játékos és az ellenfél játékmezőit egyaránt. Az "InitSeaTiles" pedig a többi tengermezőt hozza létre egy vertex bufferbe úgy, hogy ne legyen átfedés a játékterek és a tengermezők között. Ezt az ellenőrzést végzi el a "CalcTranslate" függvény az elmozdítási pozíciókat összehasonlítva. Így az egész tenger 4000 darab kis, játékmező méretű négyzetből van felépítve.

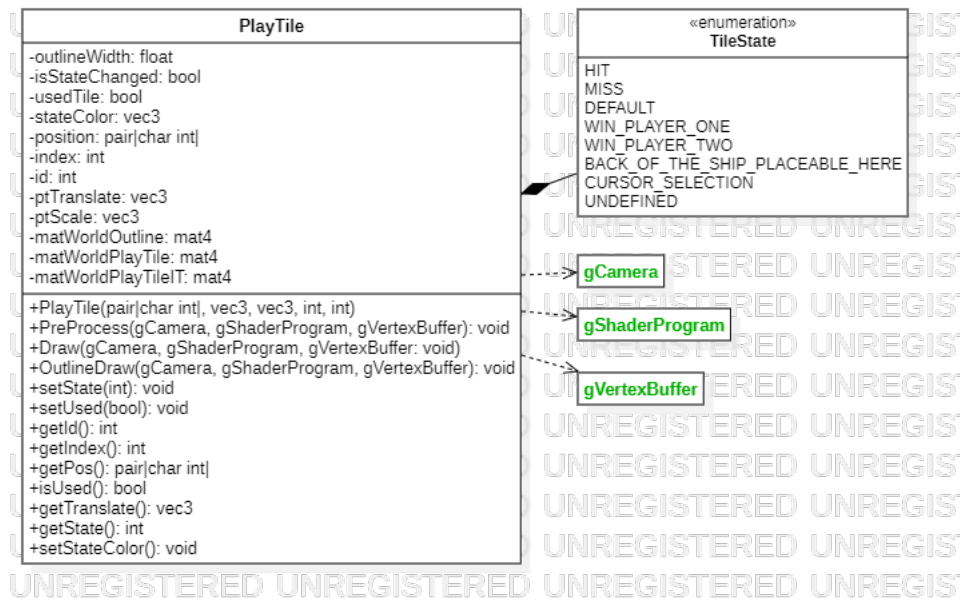
Az "Update" függvény frissíti a változót, amely nyilvántartja, hogy mennyivel kell mozgatni a tengermező textúráját a következő kirajzolásra. A kirajzolás 3 fázisban történik:

PreProcess: Megvalósítja a *3D Picking*[25] módszer előzetes lépését a játékmezőkre. Tehát szám típusú indexet rendel minden éppen látható játékmezőhöz.

Draw: Kirajzolja a tenger- és játékmezőket egyaránt.

OutlineDraw: Megrajzolja a körvonalakat a játékmezők köré. E körvonalaknak a színét az adott játékmező állapota határozza meg.

PlayTile



3.28. ábra. PlayTile osztálydiagramja.

A játékmező kezelő osztály. A "position" jelöli a játékmező koordinátáját(például "A3"). Az "index", hogy hányadik elem a *Sea* által nyilvántartott vektorban. Az "id" pedig hogy a *3D Picking*[25] megvalósításához milyen indexe lesz a játék-mezőnek. A "usedTile" értékben van tárolva, hogy egy hajó helyet foglal-e a játékmezőn vagy sem.

Az osztály része egy *TileState* felsoroló, amely a játékmező lehetséges állapotait tartalmazza. Ezen állapotok alapján állítja át a "SetStateColor" a játékmező színét a backend oldalon.

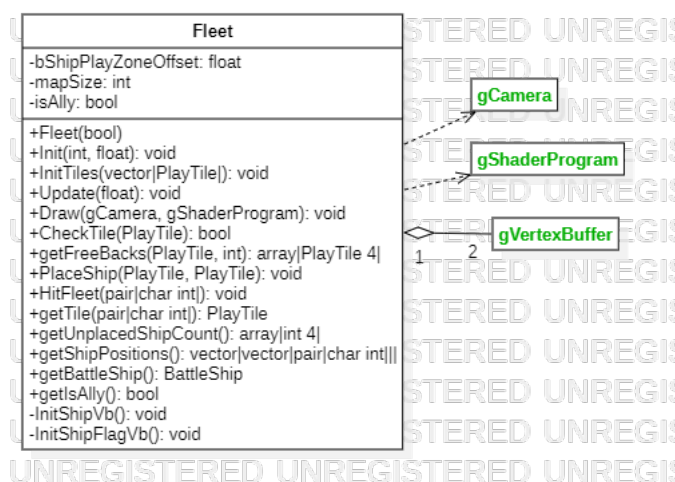
EventHandler



3.29. ábra. EventHandler osztálydiagramja.

A játékmenet esemény-kezelésével foglalkozik. A konstruktorban betölti az összes hangot és elindítja a zene lejátszást. Az "Update" kezeli a csatahajó lövésének az animációját. Figyeli, hogy ha vége van, akkor találat vagy mellé lövés hangot kell lejátszani. A "FireProjectile" játsza le az ágyú-lövés hangot és kezdeményezi a lövedék és részecskéinek animációját. A "SwitchVolume" valósítja meg az összes hang némítását. Az "ApplySoundDistEffect" pedig kiszámolja a megfelelő hangerőt a kamera és hangforrás közötti távolság függvényében.

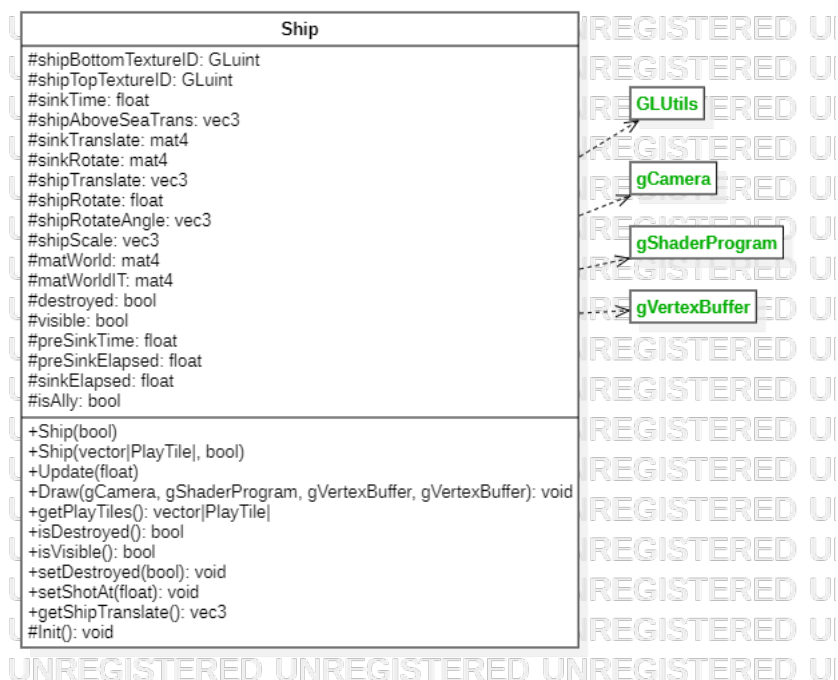
Fleet



3.30. ábra. Fleet osztály-diagramja.

A játékos flottáját kezeli. A "CheckTile" függvény ellenőrzi, hogy egy játékmező szabad-e. A "getFreeBacks" adja vissza azokat a koordinátákat, amelyek egy legalább "2x1" méretű hajó hátának megfelelőek lennének. A "PlaceShip" rak le egy hajót a hajó pozíciójának eleje és háta ismeretében. A "HitFleet" kezeli az ellenféltől jövő lövésnek a hatását a játékos flottájára.

Ship



3.31. ábra. Ship osztály-diagramja.

A hajó osztálya. A paraméteres konstruktor segítségével jön létre a hajó, a megfelelő pozícióra és a megfelelő méretre skálázva. Az "Update" függvény kezeli azokat a számításokat, amelyek a hajó elsüllyedésének az animációjával kapcsolatosak. A "destroyed" változó tartja nyilván, hogy ki van-e már löve a hajó. A "visible" az elsüllyedés animáció végeztével vált igazra és ezt követően már nem rajzolódik ki többet a hajót.

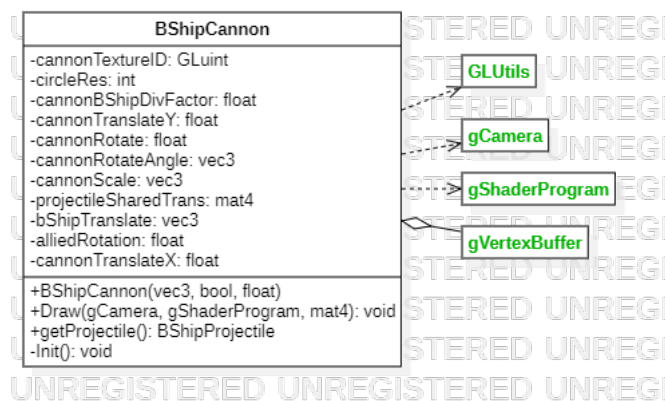
BattleShip



3.32. ábra. BattleShip osztály-diagramja.

A csatahajó osztálya, amely a *Ship* osztály gyereke. Része egy *BShipCannon* objektum, vagyis az ágyú. A "ResetForDEBUG" függvény "Debug" módban indított játékmenet esetén visszaállítja a csatahajó pozícióját az eredeti értékre, hogy ismét tesztelni lehessen a süllyedés animációját.

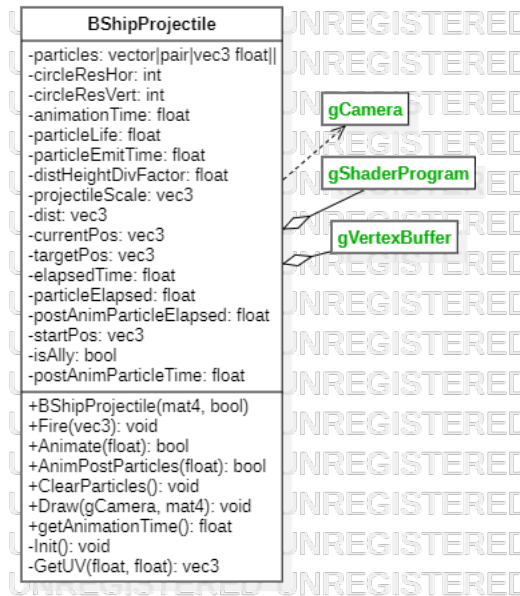
BShipCannon



3.33. ábra. BShipCannon osztálydiagramja.

A csatahajó ágyújának osztálya. Az ágyú henger alakú és modellje az "Init" függvényben van felépítve a vertex bufferbe. Része egy *BShipProjectile* objektum, vagyis a lövedék.

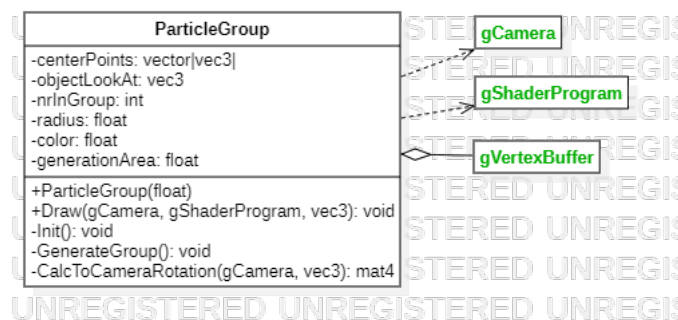
BShipProjectile



3.34. ábra. BShipProjectile osztálydiagramja.

A csatahajó ágyújának lövedéke. Ezt egy kicsi narancssárga színű gömb ábrázolja, melynek modellje az "Init" függvényben épül fel. A "Fire" és az "Animate" függvényekkel valósul meg a lövedék és a részecskéinek az animációja egy körív mentén. Az "AnimPostParticles" valósítja meg a lövést követő, még életben maradt részecskék animációját. Része egy *ParticleGroup* objektum.

ParticleGroup

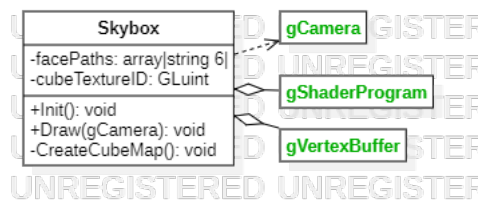


3.35. ábra. ParticleGroup osztálydiagramja.

A részecske-csoport osztálya. A "GenerateGroup" függvény alakítja ki véletlenszerűen, a "generationArea" értéke alapján, hogy hol legyenek egy kicsi

3D-s térfogaton belül a kicsi részecskék pozíciói. A "nrInGroup" határozza meg, hogy hány kicsi részecske tartozik egy csoportba, ami jelenleg 64. A "radius" határozza meg egy kicsi részecske-négyzet csúcsainak távolságát a középponttól. Ezek alapján az "Init" függvényben épül fel a részecske-csoport vertex buffere. A rajzolás során a "CalcToCameraRotation" számolja ki, hogy mennyire kell elfordítani a részecske-csoportot az Y tengely mentén, hogy az mindig a kamera felé nézzen.

Skybox



3.36. ábra. Skybox osztály-diagramja.

A **3D**-s színteret körülvevő égbolt osztálya. A "facePaths" tömbbe töltődnek be az égbolt egyes oldalainak a képei. A "CreateCubeMap" függvény hozza létre ezekből a *Cubemap*-es[28], többoldalú textúrát. A *Skybox* modelle és a shader programja pedig az "Init"-ben jön létre.

3.7. Kódolási konvenciók

A programok kódolása során egy általános kódolási konvenciót állítok fel, amely a kódot szebbé és egységessé teszi. Nem garantálom, hogy ezt minden esetben be sikerül tartani. Az alábbi listába a "{}" között lévő elemek a sablonnevek. A sablon nevek első betűi jelzik, hogy kis- vagy nagybetűvel kezdem-e a nevüket:

- Függvény paraméter: $\{függvénynév\}(\{paraméterszó\}_{paraméterszó})$
- Pointer újonnan lefoglalt memóriaterületre: $\{típus\}^* \{pointernév\}$
- Azon pointer amely egy másik pointer adataira mutat: $\{típus\}^* p\{Pointernév\}$
- Referencia: $\{típus\} \& \{név\}$
- Osztály adattag neve: $\{névszó\}\{Névszó\}$
- Frontend elem transzformációjának a neve: $\{elemnév\}Translate$
- Vertex buffer neve: $vb_ \{név\}$

- Shader program neve: $sh_ \{név\}$
- Getter-ek visszatérési értéke:
 - Primitív típus esetén - $const \{típus\} get\{Osztálytag\}()$
 - Fejlesztő által definiált típus esetén - $const \{típus\} \& get\{Osztálytag\}()$
- Osztály privát függvényeinek és adattagjainak a deklarálási sorrendje:
 1. Felsorolók
 2. Struktúrák
 3. Függvények
 4. Fejlesztő által definiált típussal rendelkező adattagok
 5. Konstans, osztály működését beállító változók
 6. Transzformációk
 7. Nem konstans változók
 8. Konstans változók, amelyeket a konstruktor tag-inicializál
- Osztály adattag-ként szereplő vektor konténerek: $std::vector<\{típus\}^*> \{név\}$
- Kisegítő osztályok: $g\{Név\}$

A fenti konvenciók a megfelelő működést biztosító kódolásban is elősegítenek:

- Ha egy pointert másolok, ami már meglévő adatra mutat, akkor a "p" karakterrel kezdem a nevét, hogy elkerüljem a többszörös heapen lévő adat törlését.
- Primitív adattípusok esetén engedélyezem a visszatérési adatok másolását, mivel azok mérete elenyésző a fejlesztő által definiáltakhoz képest.
- Figyelek rá, hogy a konstruktor által tag-inicializált változók az osztálydefiníció végére kerüljenek. Így a tag-inicializáláskor fel tudja használni a program a többi adattagok, mert azoknak már lesz értéke.

3.8. Továbbfejlesztési lehetőségek

- A hegyet meg lehetne valósítani valami grafikailag naprakészebb módszerrel, például *Displacement Mapping*[32] módszerrel, ahol egy magassági térkép lenne a bemenő adat. Ezt a magassági térképet is lehetne procedurálisan generálni, például *Perlin-zaj*[33] módszer segítségével.

- Meg lehetne oldani egy tényleges vissza-kapcsolódási lehetőséget a klienseknek a szerverre. Ilyenkor várna a szerver például 60 másodpercet, mielőtt kijelentené a másik játékost nyertesnek. Ennek érdekében a szerveren jobban le lehetne menteni a játék állapotát több adat nyilvántartásával.
- A lövedék részecskék a kamera irányába való forgása jelenleg *Cylindrical Billboarding*[30] módszerrel történik. Ezt tovább lehetne fejleszteni *Spherical Billboarding*[34]-ra. Ebben az esetben a részecskék az X tengely mentén is forognának.
- A lövedék részecskéinek lehetne idő alapú mozgás animációja, szín- és átlátszóság változása.
- Ködöt lehetne szimulálni abban az esetben, ha a kamera a tengerszint alatt és talaj felett van az Y tengelyen. Ezen felül külön hangot lehetne lejátszani, amikor a kamera tengerszint alá merül vagy mikor felszínre jön.
- A hanglejátszás jelenleg távolság alapú, ezt nevezik *Distance-based audio*-nak. Ezt tovább lehetne fejleszteni, hogy a kamera nézőpontja és a tárgy pozíciója által bezárt szög arányában ossza szét a hangot a program a két hangszóró között. Ezt hívják úgy, hogy *Positional Audio*.
- Meg lehetne oldani, hogy a program mindig a "main" függvényben érjen véget, elkerülve az "std::exit" függvények használatát.

3.9. Tesztelés

3.9.1. Szerver futása

Esemény	Elvárt eredmény
Szerver indítása a kapcsolatok fogadására	A szerver kliens kapcsolatokra vár.
Játéktér méretét változtató opció kiválasztása	A szerver kéri az 5, 7 vagy 9-es játéktér-méret beírását.
Játéktér méret megadása	A szerver beállítja az új játéktér méretet, majd visszalép a főmenübe.

Esemény	Elvárt eredmény
Port számot változtató opció kiválasztása	A szerver kéri az új port szám beírását.
Új port szám megadása	A szerver beállítja az új port számot, majd visszalép a főmenübe.
Kilépés opció kiválasztása	A szerver program bezárul.

3.1. táblázat. Szerver manuális tesztelése.

3.9.2. Kliens futása - Menü

Esemény	Elvárt eredmény
Az egér kurzorának a menü gombjára való húzása	Az adott gomb színe megváltozik.
<i>Play</i> gombra kattintás	A kliens belép a szerverre csatlakozási almenübe.
<i>Back</i> gombra kattintás a csatlakozási almenüben	A kliens visszalép a főmenübe.
Input mezőre kattintás	Az input mezőbe számok és '.' karakter begépelése elérhetővé válik. Továbbá a gombok nem reagálnak kattintásra.
Input mezőbe való gépelés közben az <i>Enter</i> vagy <i>Esc</i> billentyű lenyomása	A input mezőbe való gépelés véget ér. A menü gombok újra reagálnak kattintásra.
<i>Connect</i> gombra kattintás	A kliens csatlakozni próbál a szerverre az input-mezőkbe megadott adatok alapján.
<i>Debug</i> gombra kattintás	A kliens <i>Debug</i> módban indított játékmenet állapotra vált át.
<i>Options</i> gombra kattintás	A kliens átlép a beállításokat tartalmazó almenübe.
<i>Back</i> gombra kattintás a beállítások almenüjében	A kliens a hangerőt visszaállítja a mentett állapotra, majd visszalép a főmenübe.

Esemény	Elvárt eredmény
Jobbra mutató nyílra való kattintás a beállítások almenüjében	Eggyel nagyobb felbontás értéke kerül kiválasztásra, ha eredetileg nem a maximális volt.
Balra mutató nyílra való kattintás a beállítások almenüjében	Eggyel kisebb felbontás értéke kerül kiválasztásra, ha eredetileg nem a minimális volt.
A "Fullscreen:" szöveg melletti jelölőnégyzetre kattintás	A teljes képernyőhöz köthető beállítás átkapcsol az eredetivel ellentétesbe.
A "Vsync:" szöveg melletti jelölőnégyzetre kattintás	A függőleges szinkronizációhoz köthető beállítás átkapcsol az eredetivel ellentétesbe.
A "Music:" szöveg melletti csúszka mozgatása az egérrel balra	A zene hangereje csökken a csúszka zónáján belüli pozíciójának függvényében.
A "Music:" szöveg melletti csúszka mozgatása az egérrel jobbra	A zene hangereje nő a csúszka zónáján belüli pozíciójának függvényében.
Az "Sfx:" szöveg melletti csúszka mozgatása az egérrel balra	A hang effektek hangereje csökken a csúszka zónáján belüli pozíciójának függvényében.
Az "Sfx:" szöveg melletti csúszka mozgatása az egérrel jobbra	A hang effektek hangereje nő a csúszka zónáján belüli pozíciójának függvényében.
<i>Apply</i> gombra való kattintás	A kiválasztott beállítások elmentődnek és a videó beállítások érvényesülnek.
<i>Quit</i> gombra való kattintás	A kliens program bezárul.

3.2. táblázat. Kliens menüjének manuális tesztelése.

3.9.3. Kliens futása - Játékmenet

Esemény	Elvárt eredmény
Az egér kurzorának egy játékmezőre húzása	Az adott játémező körvonalának színe megváltozik.
<i>Esc</i> billentyű lenyomása	A kliens visszalép a menübe.

Esemény	Elvárt eredmény
<i>W, A, S vagy D</i> billentyű lenyomása	A kamera elindul rendre: előre, balra, hátra vagy jobbra.
<i>Shift</i> billentyű nyomva tartása kamera mozgása közben	A kamera gyorsabb sebességgel mozog.
Jobb egérgomb nyomva tartása és az egér mozgatása	A kamera forogni kezd az egér mozgatás irányába-
<i>T</i> billentyű lenyomása	Ki- vagy bekapcsol a szöveg megjelenítése.
<i>M</i> billentyű lenyomása	Hangok némítása be- vagy kikapcsol.
<i>F</i> billentyű lenyomása DEBUG módban indított játékmenetben	Elindul a csatahajó lövésének animációja.
<i>G</i> billentyű lenyomása DEBUG módban indított játékmenetben	Elindul vagy újraindul a csatahajó elsüllyedésének animációja.
<i>1, 2, 3 vagy 4</i> billentyű lenyomása a játék hajóméret kiválasztási szakaszában.	Ha van még az adott mérettel jelzett hajó, amit még le kell rakni, akkor átlép a hajó lerakási fázisba.
Bal egérgommbal a saját játéktér játék-mezőjére kattintása, hajó lerakási fázisban.	1x1 -es méretű hajó esetén a hajó megjelenik. Nagyobb méret esetén átlép a játék a hajó hátának lerakási fázisába, ha a hajónak létezik megfelelő hátsó pozíciója.
Bal egérgommbal a saját játéktér zöld játék-mezőjére kattintása, hajó hátának a lerakási fázisában.	Megjelenik a legalább 2x1 -es méretű hajó.
Bal egérgommbal az ellenfél játéktérének játék-mezőjére kattintása	Ha a lövéssel mi vagyunk a soron, akkor elindul a lövés folyamata az adott játékmező felé.

3.3. táblázat. Kliens játék-menetének manuális tesztelése.

3.9.4. Kliens erőforrás-használata

Az elkészített verziók során nyomon követem a kliens program memória igényét és gyorsaságát. A program gyorsaságát nem időmérés segítségével tesztelem, hanem a programba írt **FPS** számlálóval. Mivel a program egy főciklus belsején halad végig egészen a kilépési szándékig, ezért ezt megfelelőnek tartottam erre a célra. Az erőforrás tesztelés mindig "Debug" módban indított játékmenettel, "800x600"-as felbontásban és kikapcsolt **Vsync**-el történt.

Verziószám	RAM	VRAM	FPS
0.0.1	35 MB	N/A	N/A
0.0.2	23 MB	N/A	3100
0.0.3	149 MB	N/A	52
0.0.4	157 MB	N/A	38
0.0.5	175 MB	100 MB	38
0.0.6	82 MB	64 MB	277
0.0.7	80 MB	88 MB	248
0.1.0	85 MB	88 MB	258
0.1.1	87 MB	92 MB	256

3.4. táblázat. A program memória igénye és sebessége verziókra bontva.

A 3.4. táblázatban látható a program memória igénye és sebessége verziókra lebontva. Az "N/A"-val jelölt celláknál nincs mérés. A "0.0.5"-ös verzióig növekszik a **RAM** használat és csökken az **FPS** szám, mivel addig naív módon valósítom meg az algoritmusokat és a kirajzolást, illetve a program is bővül. Például a tenger- és talajmezőket ott egyesével tárolom el a memóriában és egyesével történik a kirajzolásuk is. A "0.0.6"-os verzióban viszont egy teljes kódrefaktorálást végzek el és ezáltal csökken a **RAM** és **VRAM** igény, illetve nő az **FPS** szám. Az ezt követő **RAM** és **VRAM** igény növekedés, illetve **FPS** csökkenés már csak a program bővülésének az okából fakad.

4. fejezet

Összegzés

Úgy vélem, hogy a 3.1 alfejezetben leírtakat sikeresen megvalósítottam. Létrehoztam egy interaktív, számítógépes játékot, amely a Torpedó-játék[1] háttér-logikájára alapul. Implementáltam a **3D**-s megjelenítést, hang- és zene lejátszást, szöveg kirajzolást és szerver-kliens architektúrát. Mindezt úgy tettem, hogy az alapokról indultam ki, szándékosan nem használva játék-motort[2]. Ezért úgy vélem, hogy az eredeti célom is elértem.

A szakdolgozat elkészítése során elsajátítottam a felhasznált külsős könyvtárak használatát. Ezáltal a jövőbe könnyebb lesz ezeknek, vagy az ellátott funkciókban hasonló könyvtáraknak a használata. A *C++*[5] nyelvvel való programozásban is meglepően sokat fejlődtem. Megtanultam például, hogy az inicializálatlan pointerok nem lesznek garantáltan null-pointerok.

A Torpedó-játék projekt fejlesztése során sok tapasztalatot szereztem a játékfejlesztéssel kapcsolatban. A jövőben valószínűleg át fogok térni az okos pointerok használatára. Velük könnyebb és biztonságosabb a heap memória kezelése. Ha hasonló projektbe kívánok belevágni, akkor törekedni fogok a minél általánosabb absztrakcióra ezáltal biztosítva a stabil alapokat.

A dokumentáció alapján kijelenthető, hogy az alapokról kezdett játékfejlesztés, játék-motor[2] használata nélkül egy sokkal lassabb folyamat, mint azok használatával. Viszont, mint minden nagyobb programozási projekt esetén, minél több függvény készül el, annál többet is lehet utána újrahasznosítani. Ez is az objektum-orientált programozás lényege.

Irodalomjegyzék

- [1] *Battleship(Game)*. Utolsó elérés dátuma: 2022.08.23. URL: [https://en.wikipedia.org/wiki/Battleship_\(game\)](https://en.wikipedia.org/wiki/Battleship_(game)).
- [2] *Játék-motor*. Utolsó elérés dátuma: 2022.08.26. URL: <https://hu.wikipedia.org/wiki/Vide%C3%B3j%C3%A1t%C3%A9k-motor>.
- [3] *Visual Studio 2015,2017,2019,2022 Redistributable letöltése*. Utolsó elérés dátuma: 2022.08.29. URL: <https://docs.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>.
- [4] *Radmin VPN weboldala*. Utolsó elérés dátuma: 2022.08.29. URL: <https://www.radmin-vpn.com/>.
- [5] *A C++ referenciákat tartalmazó weboldal*. Utolsó elérés dátuma: 2022.09.09. URL: <https://en.cppreference.com/w/>.
- [6] *Visual Studio hivatalos oldala*. Utolsó elérés dátuma: 2022.09.09. URL: <https://visualstudio.microsoft.com/>.
- [7] *Github hivatalos weboldala*. Utolsó elérés dátuma: 2022.09.23. URL: <https://github.com/>.
- [8] *OpenGL weboldala a fejlesztő honlapján*. Utolsó elérés dátuma: 2022.09.09. URL: <https://www.khronos.org/opengl/>.
- [9] *Hardveres gyorsítás*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/Hardware_acceleration.
- [10] *SDL könyvtár hivatalos weboldala*. Utolsó elérés dátuma: 2022.09.09. URL: <https://www.libsdl.org/>.
- [11] *SDL_mixer GitHub adattára*. Utolsó elérés dátuma: 2022.09.23. URL: https://github.com/libsdl-org/SDL_mixer.
- [12] *SDL_ttf GitHub adattára*. Utolsó elérés dátuma: 2022.09.09. URL: https://github.com/libsdl-org/SDL_ttf.

- [13] *SDL_net GitHub adattára*. Utolsó elérés dátuma: 2022.09.23. URL: https://github.com/libSDL-org/SDL_net.
- [14] *NVIDIA Nsight weboldala*. Utolsó elérés dátuma: 2022.09.23. URL: <https://developer.nvidia.com/nsight-graphics>.
- [15] *Inkrementális képszintézis*. Utolsó elérés dátuma: 2022.09.23. URL: <http://cg.iit.bme.hu/portal/sites/default/files/oktatott%20t%C3%A1rgyak/sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pek%20grafika/inkrement%C3%A1lis%203d%20k%C3%A9pszint%C3%A9zis/bmeincr.pdf>.
- [16] *Illeszkedési axiómák*. Utolsó elérés dátuma: 2022.09.23. URL: <https://tavoktatas.mnt.org.rs/sites/default/files/2020-10/Geometria%2020-%20Illeszked%C3%A9si%20axiomak.pdf>.
- [17] *Vertex fogalma a számítógépes grafikában*. Utolsó elérés dátuma: 2022.09.23. URL: [https://en.wikipedia.org/wiki/Vertex_\(computer_graphics\)](https://en.wikipedia.org/wiki/Vertex_(computer_graphics)).
- [18] *Vertex puffer*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/Vertex_buffer_object.
- [19] *Shader program*. Utolsó elérés dátuma: 2022.09.23. URL: <https://en.wikipedia.org/wiki/Shader>.
- [20] *OpenGL shading language(GLSL)*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/OpenGL_Shading_Language.
- [21] *Vertex Shader Wiki oldala*. Utolsó elérés dátuma: 2022.09.25. URL: https://www.khronos.org/opengl/wiki/Vertex_Shader.
- [22] *Fragmens Shader Wiki oldala*. Utolsó elérés dátuma: 2022.09.25. URL: https://www.khronos.org/opengl/wiki/Fragment_Shader.
- [23] *Homogén koordináta*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/Homogeneous_coordinates.
- [24] *Normalizált eszközkordináta*. Utolsó elérés dátuma: 2022.09.23. URL: <https://learnopengl.com/Getting-started/Coordinate-Systems>.
- [25] *3D Picking módszere részletesebben*. Utolsó elérés dátuma: 2022.09.23. URL: <https://ogldev.org/www/tutorial29/tutorial29.html>.
- [26] *Képpuffer leírása*. Utolsó elérés dátuma: 2022.09.23. URL: <https://learnopengl.com/Advanced-OpenGL/Framebuffers>.
- [27] *Phong megvilágítási modell*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/Phong_reflection_model.

- [28] *Cubemap leírása*. Utolsó elérés dátuma: 2022.09.23. URL: <https://learnopengl.com/Advanced-OpenGL/Cubemaps>.
- [29] *Gouraud megvilágítási modell*. Utolsó elérés dátuma: 2022.09.23. URL: https://en.wikipedia.org/wiki/Gouraud_shading.
- [30] *Cylindrical Billboarding módszere részletesebben*. Utolsó elérés dátuma: 2022.09.23. URL: <http://www.lighthouse3d.com/opengl/billboarding/index.php?billCyl>.
- [31] *SDL_image GitHub adattára*. Utolsó elérés dátuma: 2022.09.24. URL: https://github.com/libsdl-org/SDL_image.
- [32] *Displacement Mapping módszer Wiki oldala*. Utolsó elérés dátuma: 2022.09.25. URL: https://en.wikipedia.org/wiki/Displacement_mapping.
- [33] *Perlin zaj Wikipédia oldala*. Utolsó elérés dátuma: 2022.09.25. URL: https://en.wikipedia.org/wiki/Perlin_noise.
- [34] *Spherical Billboarding módszerről részletesebben*. Utolsó elérés dátuma: 2022.09.25. URL: <http://www.lighthouse3d.com/opengl/billboarding/index.php?billSphe>.

Ábrák jegyzéke

2.1. Radmin VPN Logója (https://www.radmin-vpn.com/img/logo-03.png)	6
2.2. Radmin program felülete (https://www.radmin-vpn.com/images/gallery_main_page/en/main_light.png)	8
2.3. A szerver konzolos külalakja a program indítását követően.	9
2.4. A szerver a kliens kapcsolatok kezelése közben.	10
2.5. Hiányzó beállításokat jelző üzenetdoboz.	11
2.6. A kliens-program futtatását követően a fő menü fogad.	11
2.7. Szerverre csatlakozás al-menüjének a felülete.	12
2.8. Beállításokat kezelő almenü.	14
2.9. Képernyőkép "Debug" módban indított játékmenetről.	16
2.10. Hibás portszám bemenetet jelző üzenetdoboz.	19
2.11. Rossz IP cím megadását jelző üzenetdoboz.	20
2.12. Rossz portszám megadását jelző üzenetdoboz.	20
2.13. Nem egyező verziószámot jelző üzenetdoboz.	21
2.14. Adat fogadásakor történő hibát jelző üzenetdoboz.	21
2.15. Adat küldésekor történő hibát jelző üzenetdoboz.	21
3.1. C++ logója (https://isocpp.org/assets/images/cpp_logo.png)	23
3.2. Visual Studio logója (https://visualstudio.microsoft.com/wp-content/uploads/2021/10/Product-Icon.svg)	24
3.3. GitHub logója (https://cdn-icons-png.flaticon.com/512/25/25231.png)	25
3.4. OpenGL logója (https://www.opengl.org/img/opengl_logo.jpg) . . .	25
3.5. SDL logója (https://www.libsdl.org/media/SDL_logo.png)	26
3.6. Gouraud és Phong megvilágítási modell közti különbség (https://learnopengl.com/img/lighting/basic_lighting_gouraud.png)	31
3.7. Cylindrical Billboarding módszer szemléltetve (http://www.lighthouse3d.com/opengl/billboarding/image1.gif) . . .	33

3.8. Használati-eset diagram, ami az egy aktor által elvégezhető interakciókat mutatja be.	37
3.9. Használati-eset diagram, ami a többszemélyes játékmenet interakcióit mutatja be.	38
3.10. Osztálydiagram a kapcsolatokat kezelő osztályokról.	40
3.11. Osztálydiagram a szerver program osztályairól.	42
3.12. A kliens program osztály hierarchiája.	44
3.13. gCamera osztály-diagramja.	45
3.14. GLUtils osztály-diagramja.	46
3.15. gVertexBuffer osztály-diagramja.	47
3.16. gShaderProgram osztálydiagramja.	48
3.17. TorpedoJatekClient osztálydiagramja.	49
3.18. MainMenu osztály-diagramja.	50
3.19. MenuStateHandler osztálydiagramja.	51
3.20. OptionHandler osztály-diagramja.	52
3.21. MenuRenderer osztály-diagramja.	52
3.22. GameInstance osztálydiagramja.	53
3.23. Terrain osztály-diagramja.	54
3.24. TextHandler osztálydiagramja.	55
3.25. GameLogic osztály-diagramja.	56
3.26. Mountain osztály-diagramja.	57
3.27. Sea osztály-diagramja.	58
3.28. PlayTile osztálydiagramja.	59
3.29. EventHandler osztálydiagramja.	60
3.30. Fleet osztály-diagramja.	60
3.31. Ship osztály-diagramja.	61
3.32. BattleShip osztály-diagramja.	62
3.33. BShipCannon osztálydiagramja.	62
3.34. BShipProjectile osztálydiagramja.	63
3.35. ParticleGroup osztálydiagramja.	63
3.36. Skybox osztály-diagramja.	64

Táblázatok jegyzéke

2.1. Egy játékos hajóinak a mennyisége a játéktér méretére és a hajók méretére lebontva.	7
2.2. Játéktér Koordinátarendszere	8
3.1. Szerver manuális tesztelése.	67
3.2. Kliens menüjének manuális tesztelése.	68
3.3. Kliens játék-menetének manuális tesztelése.	69
3.4. A program memória igénye és sebessége verziókra bontva.	70

Rövidítésjegyzék

- 2D = 2 Dimenziós - XY tengelyekkel megvalósított sík, 4
- 3D = 3 Dimenziós - X,Y,Z tengelyekkel megvalósított tér, 4
- API = Application Programming Interface. Egy program függvényei, melyeket más programok felhasználhatnak, 25
- CPU = Central Processing Unit. A számítógép processzora, 53
- FPS = Frames Per Second - Másodpercenként hány képet rajzol ki a program., 12
- GPU = Graphics Processing Unit. A videokártya processzora és feldolgozó egysége, 25
- IP = Internet protokoll. Egy számítógép hálózati azonosítója., 8
- LAN = Local Area Network - Helyi hálózat., 8
- MSVC = Microsoft Visual C++, 24
- RAM = Random Access Memory. Rendszermemória, 70
- TCP = Transmission Control Protocol. Hálózaton keresztüli adatátviteli protokoll, 34
- UDP = User Datagram Protocol. Hálózaton keresztüli adatátviteli protokoll, 34
- UI = User Interface - Felhasználói kezelőfelület., 13
- VAO = Vertex Array Object, 28
- VBO = Vertex Buffer Object, 28
- VPN = Virtual Private Network - Virtuális magán hálózat. Kiterjeszti a helyi hálózatot úgy, hogy az adatok titkosításával garantálja a biztonságot, 6
- VRAM = Video Random Access Memory. Videó memória, 70
- Vsync = Vertical synchronization. A monitor frissítése frekvenciájában maximalizálja a másodpercenként kirajzolt képek mennyiségét, 14
- WAN = Wide Area Network - Nagy kiterjedésű hálózat. A helyi hálózaton kívüli területet is tartalmazza, 6