Tartalom

[1. Bevezetés 4](#_Toc415459744)

[1.1. Mitől jó a játék 4](#_Toc415459745)

[1.2. Elterjedt játéktípusok 4](#_Toc415459746)

[1.2.1. Többjátékos online harci aréna (MOBA) 5](#_Toc415459747)

[1.2.2. Háborújáték 5](#_Toc415459748)

[1.2.3. Vidd haza a zászlót (CTF) 5](#_Toc415459749)

[1.3. Játék bemutatása 5](#_Toc415459750)

[1.3.1. A saját elképzelés 6](#_Toc415459751)

[1.4. Több játékos, nem csak a pályán 6](#_Toc415459752)

[2. Tervezés 7](#_Toc415459753)

[2.1. Elvárások a játékkal szemben 7](#_Toc415459754)

[2.2. Fejlesztési platform megválasztása 7](#_Toc415459755)

[2.3. Használt technológiák 8](#_Toc415459756)

[2.3.1. Irányelvek 8](#_Toc415459757)

[2.4. Szerver-oldal 8](#_Toc415459758)

[2.4.1. Node.js 8](#_Toc415459759)

[2.4.2. Websocket 9](#_Toc415459760)

[2.5. Kliens-oldal 10](#_Toc415459761)

[2.5.1. HTML5 Canvas 10](#_Toc415459762)

[2.6. Felületi tervek 10](#_Toc415459763)

[3. Megvalósítás előkészületei 11](#_Toc415459764)

[3.1. Futtatókörnyezet telepítése 11](#_Toc415459765)

[3.2. Fejlesztőkörnyezet megválasztása 11](#_Toc415459766)

[3.2.1. A hibakeresés módszere 11](#_Toc415459767)

[3.2.2. Node Package Manager 12](#_Toc415459768)

[3.3. Verziókövetés 12](#_Toc415459769)

[4. Megvalósítási eklépzelés 14](#_Toc415459770)

[4.1. Modulokra bontás 14](#_Toc415459771)

[4.2. Kapcsolatkezelő 15](#_Toc415459772)

[4.3. Parancsértelmező 15](#_Toc415459773)

[4.4. Csoportkezelő 15](#_Toc415459774)

[4.5. Játékvezérlő 16](#_Toc415459775)

[4.6. A vezérlők közti kommunikáció 16](#_Toc415459776)

[5. Megvalósítás menete 17](#_Toc415459777)

[5.1. Websocket kapcsolat felállítása 17](#_Toc415459778)

[5.1.1. A feladathoz megfelelő megvalósítás 17](#_Toc415459779)

[5.1.2. Az kapcsolat tesztelése 18](#_Toc415459780)

[5.2. A kapcsolatok kezelése 19](#_Toc415459781)

[5.2.1. Új kapcsolatok adminisztrálása 19](#_Toc415459782)

[5.2.2. Kapcsolatzárás 19](#_Toc415459783)

[5.2.3. Üzenetfogadás 19](#_Toc415459784)

[5.2.4. Üzenetküldés 20](#_Toc415459785)

[5.3. A parancsértelmező 20](#_Toc415459786)

[5.4. A csoportkezelő 20](#_Toc415459787)

[5.4.1. Csoportok létrehozása 20](#_Toc415459788)

[5.4.2. Csoportba lépés menete 21](#_Toc415459789)

[5.4.3. Csoport elhagyásának menete 21](#_Toc415459790)

[5.4.4. Üres szobák kezelése 22](#_Toc415459791)

[5.4.5. Segédfüggvények 22](#_Toc415459792)

[5.4.6. Hívható függvények 22](#_Toc415459793)

[5.5. A játékkezelő 23](#_Toc415459794)

[5.5.1. Játékba lépés 23](#_Toc415459795)

[5.5.2. Játék elhagyása 24](#_Toc415459796)

[5.5.3. A játék elindítása 24](#_Toc415459797)

[5.5.4. A játék futása 24](#_Toc415459798)

[5.5.5. A játék frissítése 25](#_Toc415459799)

[5.5.6. A játék lezárása 25](#_Toc415459800)

[5.5.7. A játéküzenetek 25](#_Toc415459801)

[5.6. Játékmotor 26](#_Toc415459802)

[5.6.1. A játékmotor célja 26](#_Toc415459803)

[5.6.2. A játék lehetséges elemei 26](#_Toc415459804)

[5.6.3. Megjelenítés a kliens-oldalon 26](#_Toc415459805)

[5.6.4. Rugalmas ütközés 27](#_Toc415459806)

[5.6.5. Ütközés fallal 27](#_Toc415459807)

[5.6.6. Ütközés játékossal 27](#_Toc415459808)

[5.6.7. Pontszerzés és annak logikája 27](#_Toc415459809)

# Bevezetés

A szórakozás fogalma generációról generációra változik. Modern világunkban számítógépek segítik munkánkat. A mozgalmas hétköznapok közepett el is felejtjük, hogy milyen lehetőségek vannak karnyújtásnyira. Így számítógépeink szórakozató képességeit hajlamosak vagyunk figyelmen kívül hagyni.

A fejlődő technika újabb és újabb vívmányai lehetővé teszik az egyre látványosabb grafikai szoftverek hétköznapi használatát. Ennek a legnagyobb piaca a Játékipar.

De a játékoknak nem kell feltétlenül lélegzetelállító grafikai elemeket mutatnia ahhoz, hogy sikeresek legyenek.

## Mitől jó a játék

Nem a grafika határozza meg a játék élvezeti értékét, ezt napjaink egyik legnépszerűbb játéka a Minecraft is igazolja. Alapvető grafikai elemeket használhatunk ötleteink megépítésére. Akár barátainkkal vagy más játékosokkal közreműködve.

Most eljutottunk mindenidők legsikeresebb játékaihoz. Ezek a játékok versenyszerűek, csapatok harcolnak csapatok ellen. A játék az ellenfél kicselezésével nyerhető meg. Az efféle játékoknak mesterévé válásához rengeteg gyakorlás szükséges. Név szerint a Quake 3, StarCraft 2 és a League of Legends képviselik legjobban ezt a kategóriát.

## Elterjedt játéktípusok

A versenyszerű játékoknak néhány fő fajtája ismert, mindegyiknek rengeteg alváltozata, újraértelmezése létezik. A legjelentősebb tulajdonsága az efféle játékoknak, hogy a játék célja jól ismert és egy ponton a nyertes egyértelműen kihirdethető. Másik közös tényező a csapatokra bontottság. A csapat egységes célért küzd, ennek nevében minden csapattagnak megvan az egyedi szerepe. A játéktól függően a csaptok méretében jelentős különbségek lehetnek. A továbbiakban a leglényegesebb játéktípusokat ismertetem

### Többjátékos online harci aréna (MOBA)

A játékosok csapatokat alkotnak és egy kijelölt területen az „arénában” küzdenek meg. A játék célja lehet egyes pontok elfoglalása, vagy az összes ellenfél megsemmisítése. Leggyakoribba az 5-5 és a 3-3 felállás. A játékban fontos a csapat együttműködése és a választott stratégia.

### Háborújáték

Itt a csapatok helyett seregekről beszélhetünk, melyek nyílt területen mérkőznek meg. A játék során a seregek célja, az ellenséges bázis elfoglalása. Ebben a játékmódban szerepe van a stratégiának, de a csapat toborzás sokkal fontosabb.

### Vidd haza a zászlót (CTF)

Mindkét csapat a bázisán lévő zászlót védi, és az ellenfél zászlóját próbája megszerezni. A játékosok szabadon mozognak a pályán, de ellenfelek akadályozhatják egymást. Az ellenséges zászló könnyen elrabolható, a zászlót csak meg kell érinteni. Amint ez megtörtént a zászló hordozója sérülékennyé válik, a zászló könnyen visszaszerezhető. A rablási kísérlet megállításáért pont jár. De ha ez nem sikerül és az elrabolt zászlót saját bázisáig juttatja egy játékos. Azzal a csapat pontot szerez.

## Játék bemutatása

A bemutatottak közül a zászlós játékmód nyerte el tetszésem. A változatos és gyors játékmenet, melltett a csapatok együttműködésének is szerepe van a játék menetében. Sőt az ellenfél kicselezése a legfőbb cél. A játék viszonylag könnyen megtanulható, nincs szükség bonyolult szabályokra. Így a játék alkalmas baráti játszmák lebonyolítására, jelentős ráfordított tanulási periódus nélkül is, viszont van lehetőség a fejlődésre. A kezdő és a gyakorlott játékos között érezhető különbségek lesznek. A csapatok együttműködése még ennél is fontosabb, és még több teret ad a versenyszerűségnek.

### A saját elképzelés

A megvalósítani kívánt játék, egy négyzetes pályán játszható. A pályán megtalálható a csapatok bázisa, ezt a csapat zászlója jelöli. A játékosok csatlakozás után egyből a küzdőtéren találják magukat. A csapatok tagjai a csapat bázisa környékén kezdik a játékot. Innentől a fő cél a pontszerzés, adott pontszám elérésével pedig a meccs megnyerése. Pontszerzésre két mód van. Egyik az ellenfél zászlójának megszerzése, majd a csapat bázisra juttatása és a saját zászló megérintése. Ez tíz pontot jelent a csapatnak. Ennek kivitelezése közel sem egyszerű, hiszen a másik csapat ezt mindenáron próbálja megakadályozni. A csapattagok az elrabolt zászlót visszajuttathatják azt a helyére a rabló játékos megérintésével. A rablási kísérlet megállítása egy pontot ér. Tehát ha sikerül is megszereznünk az ellenséges zászlót vigyáznunk kell arra, hogy ne érintkezzünk ellenféllel. De ez még mindig nem elég a pontszerzéshez. A saját zászlónknak a helyén kell lennie. Ha azt az ellenfél időközben elrabolta meg kell várnunk, hogy csapatunk visszaszerezze. Ha ez megtörténik, már megkaphatjuk a jól kiérdemelt tíz pontot.

A játék megvalósításánál nem a látványra fókuszálok, inkább a játszhatóságra és a versenyszerűségre. Az egyszerű formák nem terelik el a játékos figyelmét. A minimalista összeállítás segít a gyors döntéshozásban. Minden játékos saját csapata színét viseli. A csapatok bázisát és zászlóját szintén a csapatszín jelöli. A csapatszínek jellemzően kiegészítő színpárok, tehát jól elkülönülnek. A játéktér lehetőleg szimmetrikus, így egyik csapat sem jut semmiféle előnyhöz.

## Több játékos, nem csak a pályán

A felvázolt játékot egy időben több játékos játssza, a játéktípus magában, legalább 4 játékost feltételez. A játékosok helyi hálózaton vagy az interneten keresztül kapcsolódnak egy központi kiszolgálóhoz. A játék megvalósítása mellett egyéb funkciókra is szükség lesz. A játék megszervezésére és a játékot megelőző egyeztetésére is alkalmasnak kell lennie a felületnek. A játék tehát önállóan nem használható, egyéb szolgáltatások veszik körül, így egy a szobák kezelésére alkalmas felület és a felhasználó közi kommunikációra alkalmas.

# Tervezés

## Elvárások a játékkal szemben

A célom, hogy a játékot minél több felhasználó kipróbálhassa, élvezhesse, a már általa megszokott környezetben. Elsősorban asztali számítógépekre készül, de a felsőkategóriás hordozható eszközökön is elvárható a megfelelő működés. Elsődleges a felhasználó kényelme. Telepítés nem szükséges. Egy modern böngésző képes a szükséges rajzolási és hálózati teljesítmény kezelésére. A böngészős megvalósítás nagy előnye, hogy a szabványos megoldásoknak köszönhetően különböző eszközökön, különböző operációs rendszereken is ugyanarra a végeredményre számíthatunk.

Másfelől megközelítve más elvárásokkal találkozunk. Egy technikai szempontból fontosnak tekinthetjük a játék rajzolási teljesítményét vagy hálózati átvitelkésési tényezőjét. Egy közvetlenül az operációs rendszerre írt program nagyobb teljesítményt biztosít, így például csökken a minimális hardverkövetelmény. Ezért viszont kényelmi pontokat veszítünk, hiszen az ilyen programot telepíteni kell a böngészős változattal szemben, ahol csak egy weboldalt kell felkeresnünk. A modern böngészők és az elérhető áru számítógépek világában a legtöbb felhasználónak nem lesz gondja a teljesítménnyel.

## Fejlesztési platform megválasztása

A böngészős alkalmazásunk programozásához több megoldás közül választhatunk. Lehetőségünk van kiegészítőket használni például Adobe Flash, vagy Microsoft Silverlight. Ezeket viszont nem a böngésző tartalmazza, hanem előre telepítendőek. Ezek elavulttá váltak a böngészőkbe épített JavaScript futtatókörnyezet fejlődésével. Ez azt jelenti, hogy csakis egy tetszőleges modern böngészőt például a Mozilla Firefox-ot vagy a Google Chrome-ot kell beszereznünk, ha még nem tettük. Ez tehát a felhasználói oldal.

A kiszolgáló környezet megválasztása már nehezebb kérdés. Sok szempontot kell figyelembe venni és a választék is igen széles. A kiszolgáló oldal tervezésekor lényeges hogy hatékony megoldásokat válasszunk, hiszen a kiszolgáló központi csomópont, minden felhasználó ide csatlakozik, így ha ez nem képes tartani az iramot, azt minden játékos megérzi. Célszerű elterjedt megoldásokat használni a széleskörű támogatottság miatt.

## Használt technológiák

### Irányelvek

A JavaScript nyelvben lehetőség van az objektumorientált programozásra. Ez nagyon ajánlott, de nem kötelezettség. A probléma részekre bontása segíti a megértést. Egymással szoros kapcsolatban lévő blokkokat alakíthatunk ki. Egy probléma objektumorientált megoldásénak elkészítése jellemzően több munkát és tervezés igényel, de ez a befektetés könnyen megtérül hosszútávon. A később eszközölt változtatások viszont nagyságrendekkel egyszerűbben elvégezhetőek.

## Szerver-oldal

### Node.js

A Node.js egy szerver-oldali JavaScript alapú futtatókörnyezet. A Chrome V8 JavaScript motorjára épült, amely már évek óta a legjobban teljesítő JavaScript-motor. Sikerét jórészt annak köszönheti, hogy a programot közvetlenül gépi kódra fordítja. Így nagyságrendekkel javul a teljesítmény az értelmezett, vagy a bájt kódra fordított megoldásokkal összehasonlítva.

Ezen kívül esemény vezérelt, ez annyit jelent, hogy eseményeket készíthetünk és ezek bekövetkezéséhez logikát vagy agy további eseményeket rendelhetünk. A Node.js jelenlegi verziója már C++ kiegészítéseket is kezel. Így akár C++ könyvtárakat is használhatunk vagy esetleg a közvetlen memóriaelérés is megoldható a még nagyobb teljesítmény érdekében. A Node.js híres még a méretezhetőségéről, különösen jól teljesít kis, független feladatok elvégzésében. Több nagy cég is alkalmazza, így a Yahoo a PayPal és az eBay is. Mindez bizonyítja, hogy alkalmas valós idejű kiszolgálók megvalósítására. Fontos jellemző még, hogy mind Windows, Linux és Macintosh rendszereken is használható.

### Websocket

Valósidejű alkalmazásokhoz elengedhetetlen a gyors kétirányú adatkapcsolat. A böngészők eredetileg, http kéréseket küldtek, majd az erre kapott válasz után lezárták a kapcsolatot. Ez dokumentumok betöltésére hatékony, de folyamatos oda-vissza kommunikációra kényelmetlen és lassú.

Kapcsolódás a szerverhez

A websocket technológia használható gyors kétirányú adatküldésre. A protokoll célja, a http korlátozott kérés-válasz struktúrájából származó korlátozottságok és nagymennyiségű fejlécadat elkerülése. A websocket kapcsolat a kiépüléstől a lezárás pillanatáig végig nyitva van, és készen áll adatok küldésére és fogadására. A modern böngészők mindegyike támogatja a technológiát.

A kapcsolat létrejötte

A websocket szerver megadott TCP porton hallgat, és várja a kapcsolatokat. A kapcsolatot tehát a kliens kezdeményezi. Méghozzá kérést küld a szervernek, melyben kéri a kapcsolat felállítását. Ez egy http get upgrade kérés formájában történik. Ha ezt a szerver ezt elfogadja, egy megfelelő http választ küld. Így kiépül a kapcsolat, amelyen már szabadon folyhat az adat.

A kapcsolat lezárása

A kapcsolat lezárása több féle képen történhet. A kapcsolatot egyaránt zárhatja a szerver és kliens is. A kapcsolat zárása egyszerűbb min annak felállítása. A kapcsolatot záró fél elküldi a megfelelő kapcsolatzáró kódot. Erre a kódot fogadó fél is kapcsolatzáró kódot küld. Így biztosítva, hogy a kapcsolat zárása után már nincs forgalom. A kapcsolat záródhat egy harmadik módon is. Amennyiben a két fél hálózati összeköttetése megszakad, akkor a felek külön-külön időtúllépés miatt zárják a kapcsolatot.

## Kliens-oldal

A felhasználó a számítógépére telepített böngészővel csatlakozhat a kiszolgálóhoz. A kapcsolat során a kiszolgáló gondoskodik a megfelelő adatok szolgáltatásáról. Azonban a tiszta adatfolyam és a vezérlőutasításokat a felhasználó nem képes értelmezni. Ezért szükséges az adatok megfelelő megjelenítése. A megjelenítés mára egy szabványosított folyamattá vált. Minden böngésző ismeri ezeket a szabványokat, így a fejlesztők munkája jelentősen könnyebb. Azonban vigyáznunk kell, mivel egyes böngészők közt is vannak eltérések. Ezeket a legkönnyebb folyamatos teszteléssel kiszűrni. A tesztelést pedig többféle böngészővel végezni.

### HTML5 Canvas

A böngészők régóta képesek grafikai elemek megjelenítésére. Ez az elmúlt években ezen a téren jelentős előrelépések voltak mind funkcionalitásban mind teljesítményben. A HTML5-ös szabványos vászonra JavaScript-el rajzolhatunk alakzatokat, így téglalapot, kört, szöveget vagy akár raszter, sőt vektorgrafikus képeket is. A canvas alapú megjelenítéssel együtt jár annak könnyű beillesztése a felületbe, hiszen a böngésző saját stílusbeállításait használhatjuk.

## Felületi tervek

# Megvalósítás előkészületei

## Futtatókörnyezet telepítése

A választott futtatókörnyezet Node.js, ezt Windowson telepíthetjük a nodejs.org-ról letölthető Windows Installer-el. Linux rendszereken legkönnyebben parancssorból telepíthetjük az „*apt-get install nodejs*” paranccsal. De akár portable (hordozható) változatot is beszerezhetünk, ebben az esetben nincs szükség rendszergazda jóváhagyására, de számolnunk kell apróbb kellemetlenségekkel.

A teszteléshez, és a felület megvalósításához Mozilla Firefox-ot és Google Chrome-ot is használok Az internetezők több mint 60%-a e két böngésző egyikét használja. Így biztosítható, hogy mindkét böngészőben minden zökkenőmentesen működik.

## Fejlesztőkörnyezet megválasztása

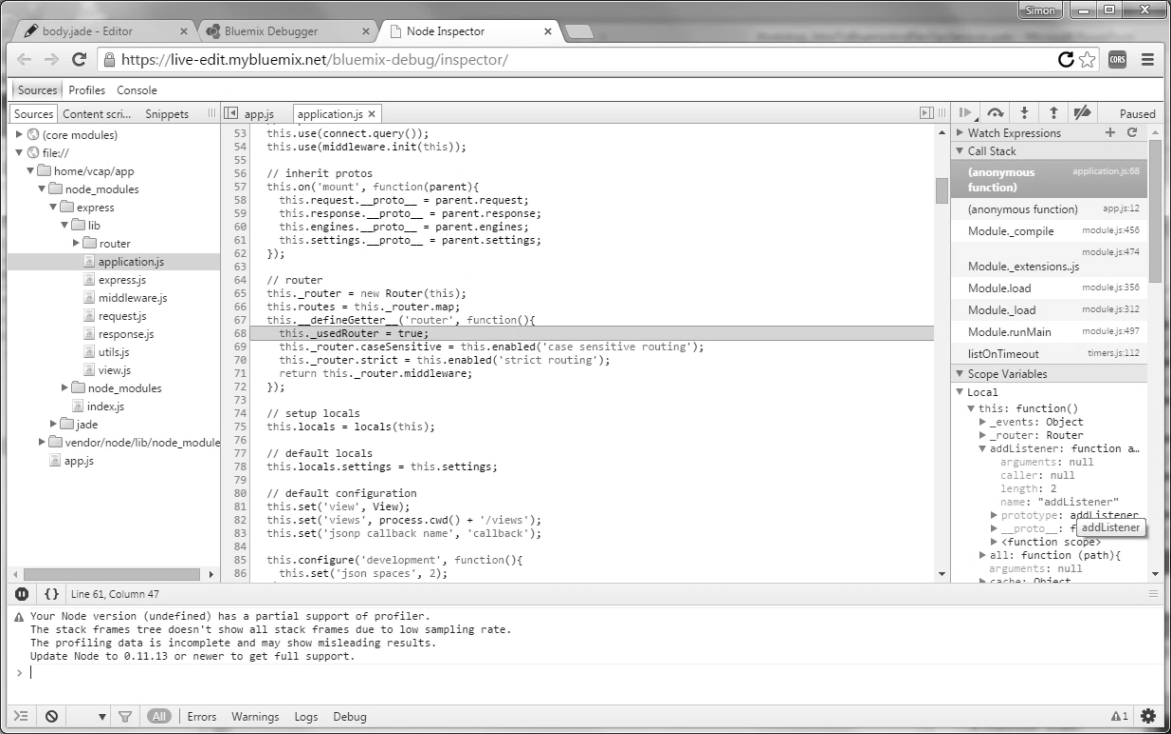
A web fejlesztők többsége egyszerű szövegszerkesztőt használ a fejlesztőmunka során. Windowson a legelterjedtebb a könnyen használható Notepad++. Támogatja a szintakszis kiemelés és a forrásfájlok automatikus formázására is van lehetőség. Viszont csak kezdetleges szókiegészítésre képes Természetesen a feladat megoldására teljesen alkalmas.

A szoftverfejlesztés során viszont nem csak a forrásfájlokat szerkesztünk. A program logikájának megvalósítását menet közben találjuk ki. Az így megírt kódblokkok tesztelése természetes és szüksége. Előfordulnak azonban összetett részfeladatok, amelyek elkészítése nem sikerül egyből. A hibásan megírt program javítását hibakeresésnek vagy debugolásnak hívjuk.

### A hibakeresés módszere

A hibakeresés során speciális hibakereső szoftvert használunk. Amely lehetővé teszi a program lépésenkénti futtatását. Így képet kaphatunk a program futásának bármely pillanatáról. Vizsgálhatjuk a változók értékei, és a programot soronként léptethetjük, ezzel a hibák sokkal könnyebben felderíthetőek, mintha csak egy hibás eredmény vagy hibaüzenetet látnánk a program lefutása után.

Az általam választott platform is rendelkezik ilyen eszközökkel. Legelterjedtebb a node-inspector. Ez webes felületen engedi programunk vizsgálatát. A mellékelt ábrán látható a node-inspector webes felületének felépítése.



1. ábra – Node inspector webes felület

### Node Package Manager

A Node.js egy tartozéka az „npm”, ami a Node.js csomagok telepítésére és publikálására alkalmas. A sokféle probléma és az ezekre adott sokféle megoldás kezelésére alkalmas ez. Egy csomag telepítése a „npm install <csomagnév>” paranccsal végezhető. A csomagok telepítése történhet a központi adatbázisból, az internetről, vagy helyi forrásból.

## Verziókövetés

A szakdolgozat készítési folyamat egésze alatt verziókövetést használok. Erre a legalkalmasabb a Git nevű verzókövető szoftver. Egyaránt képes kis és nagy projektek kezelésére. A verziókövetés segít átlátni a fejlesztések haladását, sorrendjét. A verziókövetés néhány egyszerű műveletből áll. A szerkesztett fájlokat először kiválasztjuk szakszóval stage-eljük majd amint elvégeztük a kívánt módosításokat és úgy gondoljuk, hogy érdemes rögzíteni haladásunkat, véglegesítjük más szóval commit-oljuk az új verziót. Lehetőség van még új haladási szálak létrehozására. Az így keletkezett ágakat a változtatások összesítésével egyesíthetjük, így új verzió alakítható. A kényelmi funkcionalitás sem elhanyagolható. Egy parancs kiadásával továbbíthatjuk a helyi változtatásokat egy központi tárolónak. A verziókezelés egyszemélyes projekteknél nem kulcsfontosságú, de felgyorsítja és átláthatóvá teszi a programfejlesztést.

# Megvalósítási eklépzelés

## Modulokra bontás

A program írása közben Igyekeztem előre gondolkodni, és minél szakszerűbb, könnyen megérthető megoldásokat produkálni. Ennek a tervezési folyamatnak során határoztam el, hogy külön fájlokat fogok létrehozni a programegységeknek. Ez segíti az átláthatóságot és így a szerkesztés folyamatát.

A moduláris programozás módszere szerint a programot a különböző függvények bontjuk. Ezek mindegyike egy pontosan meghatározott feladatot lát el. Emellett az egyes programrészek a lehető legkevésbé befolyásolják a program egészének működését és a programrészek egymással való kapcsolatát. Ennek értelmében nem használunk vezérlőváltozókat. Megfigyelhető még hogy a függvények két nagy csoportot alkotnak. Egyik a „vezérlők” csoportja, a másik a „dolgozók” csoportja. A vezérlők a dolgozókat fogják össze, a dolgozók pedig kisebb részfeladatokat látnak el.

A Node.js támogatja a moduláris programozást, tehát tartalmaz egy modul betöltő rendszert. A modulok készítése egyszerű csak az exports változó egy attribútumába kell írnunk tetszőleges néven az elérni kívánt függvényt, vagy értéket. Amennyiben egyetlen objektumot szeretnénk exportálni, használhatjuk a module.exports változót. Így nem kell jellemzőnkként felépíteni az objektumot. Modulok betöltésére a require függvényt használhatjuk, melynek első paramétereként a betölteni kívánt fájl nevét kell megadnunk. Ez mind a Node.js jellegzetessége, tehát a szabványos JavaScriptnek így a V8-nek sem szerves része.

Az én megvalósításom is ezekre alapul. A meghívott start program tölti be a többi programrészt. Ez elegáns megoldás, hiszen könnyen kezelhetjük, hogy mi töltődik be. Esetlegesen is csak a forrásfájlok elejét kell szerkesztenünk, ha több vagy kevesebb modult szeretnénk betölteni.

## Kapcsolatkezelő

A kiszolgálóarhiterktúra szélső eleme. Minden beérkező csomag ezen halad át. Folyamatosan hallgat a megadott TCP porton. Új kapcsolat esetén foglalkozik a kapcsolat felállításával, és a felhasználó mielőbbi kiszolgálásával. Üzenet esetén gondoskodik arról, hogy a kiszolgáló egyéb érintett részei tudjanak a bejövő információról. Minden bejövő kapcsolatot egy felhasználóhoz rendel. A kiszolgáló a továbbiakban ez alapján könnyen tudja kezelni a kapcsolatokat. Fontos szerepe van még kapcsolatok lezárásában is.

## Parancsértelmező

A parancsértelmező a központi kiszolgáló és a kapcsolatkezelő közt helyezkedik el. Minden olyan üzenet, ami a kapcsolatkezelőn túljut, a parancsértelmezőbe ér. A parancsértelmező feladata az üzenetek feldolgozása, majd a döntés meghozatala. Parancsnak számít minden olyan üzenet, ami per (/) jellel kezdődik. Ezek értelmezve vannak. Minden egyéb üzenet felhasználói üzenetként jelenik meg.

Ez a modul a kiszolgáló vezérlője. Minden bekövetkező változás ide vezethető vissza. Ha újabb feladatokat szeretnénk a kiszolgálóhoz rendelni. Itt kell gondoskodnunk arról, hogy a megfelelő parancs a kívánt programrészek lefutását eredményezze.

## Csoportkezelő

A csoportvezérlő a kiszolgáló magját képezi. A csatlakozó felhasználók mindegyikéről tudomása van. Alapértelmezetten mindenki egy speciális csoportba kerül. Innen indulva készíthetnek saját csoportot, vagy csatlakozhatnak egy már létezőhöz.

A csoportok jelszóval védhetőek. Így csak a jelszót ismerő játékosok tudnak csatlakozni. A jelszóval védett csoport ideális lehet például ismerőseinkkel való beszélgetésre, mert így egyéb játékos biztosan nem fognak zavarni.

Minden csoportnak van egy tulajdonosa, aki jogosult azt átnevezni vagy annak jelszavát megváltoztatni. Csoportot viszont a csoport tulajdonosa sem szüntetheti meg. A csoport csak abban az esetben törlődik, ha azt minden felhasználó elhagyta. Amennyiben a csoportot tulajdonosa elhagyja, a csoport új tulajdonosa az első csatlakozott játékos lesz.

## Játékvezérlő

A Játékvezérlő a programnak azon része, amely az egyes Játékok lebonyolítását végzi. Egy-egy játékvezérlő rendelhető a mindegyik csoporthoz. A játékvezérlő már valós idejű technológiát alkalmaz. Így például képes követni minden egyes játékos nyomva tartott billentyűit, és erre közvetlen választ is képes küldeni. A játékvezérlő szintén modulokból áll. A leglényegesebb a csatlakozó vagy távozó játékosok kezelője, a játéklogika, és a kliensek kiszolgálásáért felelős részek.

## A vezérlők közti kommunikáció

A vezérlők közti kommunikáció megtervezése igazi kihívásnak bizonyult. Szem előtt tartottam a modulok hierarchiáját, és ehhez illesztettem a kapcsolati sémát is. Viszont bizonyos esetekben megspóroltam egy-egy felesleges lépést. Ennek értelmében a vezérlők bizonyos műveleteket egy közös objektumon keresztül bonyolíthatják le. Ez sok extra munkát spórol meg.

A vezérlők betöltésének sorrendje is kérdéses volt. Mivel a vezérlők egymásra hivatkoznak, előfordulhat, hogy az egyik vezérlő a másik betöltése előtt már szeretné azt használni. Erre megoldás, ha az egyes modulok betöltése után megvárjuk, hogy az utolsó modul is betöltődjön.

# A megvalósítás menete

A megvalósítást a kiszolgáló oldalról közelítettem meg. A kiszolgáló a központ egység. Minden körülménytől függetlenül működőképesnek kell lennie. Tehát a kiszolgáló üzemel akkor is, ha nincs semmiféle kapcsolat. A kezdeti fázisban pedig nincs is mi kapcsolódjon. Így a kliens-oldal fejlesztése inkább a kiszolgáló teszteléseként alakul.

A fejlesztés egy sor testprogram megírásával kezdődött. A választott technológiákat csak részben ismertem. Némi tapasztalatszerzésre volt szükségem.

## Websocket kapcsolat felállítása

A program kulcsfontosságú eleme a hálózati kapcsolat. A játék működéséhez valós idejű kapcsolat szükséges. A választás a legtöbb böngésző által támogatott websocketre esett. Ez viszont nem minden. A kiszolgáló megvalósításánál megválaszthatjuk, hogy melyik megvalósítást használjuk. A választék széles, mindegyiknek megvannak a sajátosságai.

### A feladathoz megfelelő megvalósítás

A legelterjedtebbek a „ws” és a „socket.io” de említést érdemel még a „faye” a „socketcluster”. Az utóbbira jellemző, hogy jelentősen nagyobb teljesítményre képes társaihoz viszonyítva. Ezt az összes processzormag kihasználásával éri el. Érdekes még a „primus” megvalósítás. Ez az elérhető megvalósítások egy gyűjteménye, tartalmazza az eddig említetteket és még néhány kevésbé ismertet is. Egyszerűen válthatunk a megvalósítások között. Ezen kívül magában foglal néhány sajátos fejlesztést is, így nagyobb stabilitást nyújt.

A választás ismét nehéz volt. A két kiemelkedő megvalósítás egyike a ” ws”, ami a leggyorsabb megvalósítás. Nagyságrendekkel gyorsabb, mint a többi. A másik a „socket.io”, ami régebbi alternatív technológiákat is támogat, így a régebbi böngészővel rendelkező felhasználók is kapcsolódhatnak. A fejlesztési feladat szempontjából mindkettő fontos, de én fontosabbnak tartottam, hogy a gyorsabb megoldást alkalmazzam, még ha ez néhány felhasználónak kényelmetlenséget is okoz. Ezeket figyelembe véve tehát a „ws”-el kezdtem kísérletezni.

### Az kapcsolat tesztelése

Az első kihívás a kapcsolat felállítása. Szerencsére példaprogram bőven akad és a feladat sem nehéz. A „ws” modul betöltése után példányosítjuk azt. A kapott objektumhoz egy eseménykezelőt rendelünk. Ez a programrész új kapcsolat fogadása esetén hívódik meg. Ezután a kapcsolathoz rendelünk eseménykezelőt. Ez az egyes üzenetek érkezésekor hívódik meg. Paraméterként megkapjuk az üzenetet is. A mellékelt példaprogram a fogadott üzenetet kiírja, és válaszként az üzenetet csupa nagybetűvel visszaküldi.

**var** WebSocketServer = require('ws').Server;

**var** wss = **new** WebSocketServer({ port: 8080 });

wss.on('connection', **function** connection(ws)

{

ws.on('message', **function** (msg)

{

console.log(msg);

ws.send(msg. toUpperCase());

});

});

2. ábra – Websocket szerver példa

Ezt a programot könnyen tesztelhetjük is. Csak egy böngészőre van szükségünk. A programrészt akár a fejlesztői parancssorba is másolhatjuk.

**var** connection = **new** ebSocket('ws://127.0.0.1:80');

connection.onopen = **function** ()

{

connection.send("teszt üzenet");

};

connection.onmessage = **function** (msg)

{

alert(msg.data)

};

3. ábra – Kapcsolódás és üzenetküldés

Ha a kiszolgáló fut, akkor a második ábrán mutatott program sikeresen kapcsolódik a helyi szerverhez, majd „teszt üzenet”-et el is küldi. A kiszolgáló ezt fogadja, kiírja, és csupa nagybetűvel visszaküldi. A kliens ezt fogadja és feldob egy ablakot a „TESZT ÜZENET” felirattal. Tehát a kapcsolat működik, sőt annak mindkét oldalához logikát rendelhetünk és az üzenetet fel is dolgozhatjuk.

## A kapcsolatok kezelése

### Új kapcsolatok adminisztrálása

A kapcsolatok kialakítását megoldott, de ez még nem sok. A kapcsolatokat azonosítani kell. Minden egyes kapcsolat új felhasználót hoz létre. A felhasználóhoz egyéb adatok is tartoznak így például annak neve.

Az üzeneteket elküldés előtt megcímezzük. A címzettet pedig esetünkben nem a kapcsolat, hanem egy felhasználó a felhasználók mindegyike egy egyedi azonosítóval rendelkezik. A felhasználókat azonosító alapján találjuk meg. Az azonosító egy számlálóból származik. Ez gyakorlatban garantálja, hogy nem lesz két felhasználó ugyanazzal az azonosítóval.

### Kapcsolatzárás

A kapcsolatkezelés része még a kapcsolatok lezárása. A kapcsolat lezárása esetén, fontos hogy további adatcserére nincs lehetőség. Ez problémát okoz, abban az esetben, ha a kapcsolatot éppen üzenetküldés közben zárjuk, vagy a kapcsolat lezárás nélkül megszakad. Az üzenet vége már nem küldhető el. Ezt a kivételt kezelnünk kell. A hibát az alsóbb protokollok adják, jellemzően, azért mert zárt kapcsolaton próbálunk üzenetet küldeni, ami nem lehetséges. A kivétel kezelésére csak meg kell hívnunk a kapcsolat zárása metódust. Ezzel ugyanaz az eredmény mintha a felhasználó szabályosan távozott volna a kapcsolat bezárásával.

A kapcsolat zárása fontos mozzanat a kiszolgáló részéről, hiszen azt kezelni kell. Egy kapcsolat bezárásával egy felhasználó távozik. Ekkor körültekintően kell eljárni, biztosítani kell, hogy a felhasználó távozásáról értesül minden érintett részprogram. Ebben az esetben csak a csoportkezelő, mert a további intézkedések erre vannak bízva.

### Üzenetfogadás

A kapcsolatkezelés része még a beérkező üzenetek irányítása is. Az üzenetek értelmezését a parancsértelmező végzi. Így hát a beérkező üzenetek a parancsértelmező kapja meg. Az üzenet mellett szerepel a küldő is. Ez igen fontos mivel tudnunk kell, hogy melyik parancsot melyik felhasználó küldte. Ezután az üzenettel a kapcsolatkezelő nem foglalkozik.

### Üzenetküldés

A kapcsolatkezelő még egy dologról gondoskodik, ez pedig a kimenő üzenetek. Minden kimenő üzenet egy felhasználónak van címezve. Ez az üzenet a felhasználóhoz rendelt kapcsolaton fog célba érni. A program bármely ponton küldhet üzenetet a kliens felé. Ehhez csupán egy felhasználó objektumra lesz szükség.

## A parancsértelmező

A parancsok mit sem érnek, ha nincsenek végrehajtva. A parancsértelmező gondoskodik arról, hogy a felszanáló által kiadott parancs a felelős modulhoz jusson, a megfelelő paraméterekkel. Lényegében csak parancsok szintaxisát rendeli programrészekhez.

Az kapott üzenetekről először eldönti, hogy parancs vagy felhasználói üzenet-e. Ha parancs, akkor először az üzenetet darabokra vágja a megadott elválasztó karaktereknél. Az így kapott tömböt elemenként már ki lehet értékelni.

Az érintett modul kiválasztása az első. Ez lehet a csoportkezelő, vagy a játékvezérlő. Majd következik a parancs kiválasztása. A csoportkezelőnek legfontosabb utasításai a csoport létrehozása, a csoportba lépés, a csoport elhagyás, a létező csoportok listázása és egy adott csoport adatainak lekérése. Ezek közül néhányat csak paraméterrel lehet meghívni.

## A csoportkezelő

A csoportkezelő, mint már említettem központi része a kiszolgálónak. Fontos feladatot lát el a kapcsolatkezelőtől és a parancsértelmezőtől is utasításokat fogad el. Kezdeti állapotban is létezik egy speciális szoba, a nulladik szoba. A nulladik szoba a modul betöltésekor jön létre. A kapcsolatkezelő minden új kapcsolat esetén ebbe a csoportba helyezi az újonnan létrejött felhasználót.

### Csoportok létrehozása

A kapcsolódó felhasználók a megfelelő paranccsal hozhatnak létre szobát. A parancsot a parancsértelmező kapja meg, és értelmezi. Megfelelő parancs esetén kérést küld a coportkezelőnek, ami teljesíti az utasítást. A szoba létrehozás parancsnak megadható a létrehozni kívánt szoba neve, ezt nem kötelező megadni. Ha nincs megadva név, akkor a szoba neve a szerveren beállított módon a csoportot létrehozó játékos nevéből generálódik.

### Csoportba lépés menete

A felhasználó a megfelelő parancs kiadásával csoportba léphet. A csoportba lépés több mozzanatra osztható.

Csoportba lépés feltételei

Akárki akármikor megpróbálhat tetszőleges csoportba csatlakozni. Ennek két kimenetele lehet. Az egyik, hogy sikerül a szobaváltás. Így a felhasználó kérése teljesült. A másik, hogy a kérés végrehajtása megszakad. Ennek több oka is lehet. A szobába lépés nem történhet meg, ha a felhasználó egy nem létező szobába próbál csatlakozni. Vagy akkor, ha a csoport jelszóval védett és a megadott jelszó helytelen. A meghiúsult végrehajtás pedig hibaüzenettel jár, amit a felhasználó meg is kap. A parancs kiadásának így semmi hatása nincs. A felhasználó marad az eredeti csoportjában.

Sikeres csoportváltás lépései

Ideális esetben a csoportváltás sikeres. Először a felhasználó lekerül a csoport tagjait tartalmazó listáról. Ilyenkor a felhasználó eredeti csoportjának tagjai kapnak egy üzenetet arról, hogy a megadott felhasználó elhagyta a szobát, tehát innentől ő nem kapja meg az szoba üzeneteit.

Az új csoportba lépésnél a célcsoportban lévő felhasználók listájára kerül az új tag. Emellett a célszoba tagjai kapnak egy üzenetet arról, hogy az adott felhasználó csatlakozott, és már ő is „hallja” a beszélgetést.

### Csoport elhagyásának menete

A csoport elhagyására is van lehetőség. A szoba elhagyása a csatlakozáshoz hasonló. Az eredeti szobába listájából törlődik a felhasználó, az új szoba listájába pedig belekerül. Emellett mindkét csoport tagjai megkapják a megfelelő értesítést. Egyébként ez egyenértékű a nulláadik szobába való csatlakozással, ezért gyakorlatban ez így van megoldva.

### Üres szobák kezelése

Egy szoba megüresedik, hogyha minden felhasználó elhagyja azt. Itt elhagyás alatt értjük a másik szobába csatlakozást is. Az üres szobák felhalmozódását tehát meg kell előzni. Erre az a megoldás született, hogy ha a szobát az utolsó felhasználó is elhagyja, a szoba megszűnik.

### Segédfüggvények

A szobakezelő több segédfüggvényt is tartalmaz, ezek nem feltétlenül szükségesek a szobakezelés működéséhez, de logikailag ide tartoznak. Ilyen függvény például a csoportok listázása, vagy a csoportos üzenetküldés. A modul tartalmaz segédmetódusokat, de a legtöbb kívülről is elérhető.

### Hívható függvények

A kiszolgáló minden részprogramból elérhető hívások ezek. Egy zárt egységet, ebben az esetben a szobakezelőt rejtik maguk mögött. Csak korlátozott hozzáférést engednek. Pontosabban csak az előre megszabott utasításokat hajtják végre. A helyi változókkal és a felhasználók kezelésével a hívó programrésznek nem kell foglalkoznia. Például csak annyit kell mondanunk, hogy az adott felhasználót az adott szobába szeretnénk rakni. Ezzel mi végeztünk is a feladattal. A csoportkezelő majd gondoskodik annak elvégéséről

Ez a megoldás követi az objektumorientált programozás adatrejtés elvét, ami szerint az adattagok alapesetben védettek, és csak a szükséges adattagok érhetőek el. Az elérés pedig közvetlen elérés helyett jellemzőkkel van megoldva.

A JavaScript nyelvben lehetőség van jellemzők létrehozására. Ez viszont más nyelvekhez képes jelentősen kényelmetlenebb.

4. ábra – Jellemző létrehozása

**function** join()

{

//...

}

Object.defineProperty(**this**,"join", {writable: false, value: join});

Az ábra mutatja a csoportkezelő egyik jellemzőjének létrehozását. Ennek első lépése az elérni kívánt függnény létrehozása. Eztuán az „Object.definePorperty” metódussal készíthatő el a kívánt jellemző. Az első paraméter a módosítani kívánt objektum, a végrehajtás után ennek az objektumnak lesz tagja a jellemző. Az új attribútum neve a második paraméterként megadott változó értéke lesz. A harmadik paraméterben pedig egy asszocív tömbben adhatjuk meg jellemző leírását. A lerás határozza meg, hogy mi történjen a jellemző írásakor, olvasásakor vagy meghívásakor. Ezenfellül megadhatjuk, hogy a jellemző felülérható-e.

## A játékkezelő

Minden csoporthoz alapértelmezetten egy játékkezelő is tartozik. A játékkezelő kezdetben nem aktív. Már ebben a passzí állapotban is fontos szerepe van. A játékkezelőnek mindig készen kell állnia a játék elindítására. Ezért már a kezdettől figyelemmel kíséri a szobába lépő és azt elhagyó játékosokat.

### Játékba lépés

Ahogy a felhasználó játékba lép, azt játékosként is kell adminisztrálni. A játékosokhoz nem csak a névvel, azonosítóval és kapcsolattal ellátott felhasználói adatstruktúra tartozik. A játékos lehet csapat tagja, és rendelkezhet pontszámmal és még sok más a játéktípustól függő változóval. A játékosok jellemzőit csatlakozáskot alaphelyzetbe kell állítani. Ezeket a tulajdonságokat részben a játék határozza meg. A játékmenet pedig jórészt ezen értéktől függ.

A játékosok jellemzői

A játékos jellemzői egy játékos objektum attribútumaiként vannak tárolva. A játékos elsőső attribútuma annak típusa. Alapesetben játékos, de ennek szerepe, hogy a részprogramok is tudhassák, hogy a kérdéses objektum játékos. A második jellemzője a játékosoknak a felhasználó. Ez a felhasználó típusú objektum lehetővé teszi az egyedi azonosítást és a közvetlen üzenetküldést. A játékkezelő a játékost nevének a felhasználó nevét használja.

Ezután a játékra jellemző attribútumok következnek. Így a játékos csapata, a játékos pontszáma, pozíciója, sebessége, mérete. Követve vannak még a játékos lenyomott gombjai is, és az utolsó kiküdött frissítés.

### Játék elhagyása

A játék elhagyásának kezelése egy újabb feladat. Ennek során a játékost ismét csak el kell távolítani a csapatból, de itt már egészen más dolgokkal is kell foglalkozni. Itt már a játék logikájától függően kell eljárni. Esetünkben a felhasználó pontszámát nullázni kell, és ha esetleg a játékosnál zászló van, akkor annak is vissza kell kerülnie a helyére.

### A játék elindítása

A játék a start metódus hívásával indítható el. A játék ekkor végzi el az utolsó előklszületeket. Így betölti a pályaelemeket. Többek között a zászlókat és a pályán lévő akadályokat. A játék viszont még mindig nem indulhat. A játékelemeket még inicializálni kell. Ezt egyetlen függvény végzi. A függvény a játékelem tipusától függően végzi el az összes szükséges beállítást. A játék elindításával a játék státusza „running”-ra vált. Ezzel jelzi, hogy a játék elkezdődött. A játék mostámr valóban készen áll a működése.

### A játék futása

A játék menete a játék folyamatos frissítéséből származik. Ez a firssítés ideális esetben fix időközönként következik be. Minden frissítés alkalmával lefut a játék frissítésére szolgáló logika. A logikában van meghatározva, hogy a játék hogyan reagál egyes helyzetekre, például a megfelelő gomb lenyomása esetén a játékos mozgásba lendül. Ha a mozgás hatására beleütközik valamibe, akkor visszapattan. A játéklogika efféle esetek feldolgozásából áll.

A játék futása tehát a frissítéseknek köszönhető. A frissítési programrészt pedig pontos időközönként kell meghívni. Erről a játék elindításakor létrehozott időzítő gondoskodik.

timer = setInterval(**function** (){update()}, 10);

<https://www.google.hu/webhp?hl=hu>5. ábra – Időzítő készítése

### A játék frissítése

A JavaScript alapértelmezetten támogatja időzítőket. A mellékelt ábrán látható ennek működése. A „timer” a játék egy belső objektuma. A „setInterval” függvénnyel időzítőt készíthetünk, amely tetszőleges függvényt hív meg tetszőleges időközönként. Esetünkben a játék frissítésére szolgáló „update” metódust 10 milliszekundumonként, tehát másodpercenként 100-szor. Minél kisebb frissítések között eltelt idő annál folyamatosabb lesz a játék. Ennek persze határt szab a számítás és megjelenítés teljesítménye is. A számítási sebesség nem gond. Gyakorlatilag csak a megjelenítés sebessége szab határt. A forgalomban lévő monitorok frissítési sebessége korlátozott, jellemzően 60 hertz. Ez annyit jelent, hogy a kép másodpercenként 60-szor frissül. Ez jelentősen kevesebb, mint a játékfrissítés sebessége. Ennek eredménye, hogy a megjelenítés nem fogja a játék minden állapotát ábrázolni. Előfordul majd, hogy képkockák között két frissítési ciklus is lefut. Játékoknál ez gyakran előfordul, ennek a frissítési eltérésnek nincs szemmel látható következménye.

### A játék lezárása

A játék játékos nélkül nem létezhet. Ezért ha az utolsó felhasználó is elhagyja a játékot, a játék véget ér. A játék frissítését le kell állítani. Ezt már korábban mutatott „timer” objektum segítségével tehetjük. Pontosabban ennek elnevezésére és tárolására csak azét volt szükség, hogy azt később törölhessük. Ha nem mentettük volna el, akkor most nem tudnánk törölni azt. Az időzítő funkciók egyébként globális objektumként érhetőek el, tehát a „require” használata nélkül.

### A játéküzenetek

A játékkezelőnek fontos hogy a felsőbb utasításokat fogadni tudja. Erre az „message” metódus ad megoldást. A játéknak a kiszolgálóprogram akármelyik részprogramja küldhet üzenetet. Ezek az üzenetek felhasználóhoz kötöttek. Az üzenetek itt ismét értelmezése kerülhetnek. Az már a játéklogikától is függhet. Valójában a megvalósításban ezek az üzenetek csak a felhasználó által megnyomott billentyűk kezelésére van használva, így ezezk az üzenetek nincsenek valójában lekezelve.

## A játékmotor

### A játékmotor célja

A játékmotor nem más, mint a játékötlet megvalósításának a keretrendszere. A játékmotor megoldásokat biztosítja az elemi műveletek végrehajtásához. A játékmotor jól illeszkedik az eddigi rendszerbe. A játékmotor valójában a játékkezelő része. Egységes módon enged interakciót a felhasználói felülettel. Ez a már elkészült programrészeken nyugszik. Az üzenetek elküldése a már jól bevált módon zajlik. Kérdéses viszont, hogy az elküldött adatot hogyan ábrázoljuk a felhasználói oldalon. Ennek megoldása a kliens-oldali parancsfeldolgozás. A kliens megkapja a rá vonatkozó utasításokat, egy objektumstruktúrában. Ezt feldolgozva megkapja, hogy milyen objektumokat kell megjeleníteni.

### A játék lehetséges elemei

A játékhoz csak a néhány elemre van szükség. Ezek mindegyike meghatározott tulajdonságokkal bír. Legfontosabbak a zászlók. Ezeknek egyik jellemzője a tulajdonos csapat azonosítója. Játékelemek még a játékban lévő falak, körök és egyéb akadályok. Ezek mindegyike eltérő módon van kezelve mind a szerveré és kliens oldalán.

### Megjelenítés a kliens-oldalon

Az egyes elemekhez más-más megjelenítési procedúra van rendelve. Általánosan igaz, hogy az elemeknek van pozíciója, és valamiféle kiterjedése. Vonalak kezdő és végpontjukkal vannak megadva. A köröknek pozíciója és sugara van. Szövegnek pozíciója és értéke és mérete is van. A most felsoroltak mindegyikének megadhatunk színt is. Vannak még speciális jellemzők is, ezek egyike alkalmas a szöveg középre igazítására. A megjelenítendő adathalmaz tartalmaz még egy pozíció tipusú bejegyzést is. Ez az adott játékos pozícióját jelöli a pályán. Ennek ismeretében megoldható, hogy a „kamera” kövesse a játékost. A követés gyakorlatilag csak annyit jelent, hogy minden el van tolva ezzel a pozícióval.

Lehetőség van még közvetlenül a felületre rajzoni. Ekkor a követett játékos pozíciója nem befolyásolja a megjelenítést.

### Ütközés fallal

### Rugalmas ütközés

### Ütközés játékossal

### Pontszerzés és annak logikája