Beágyazott rendszerek szoftvertechnológiája

Házi feladat tervdokumentáció

RisTan

Csapattagok:

- Palkó András (H4JMOA)
- Dányi Péter (X3CUQW)
- Szappanos Miklós (QTPTZD)

Konzulens:

Mázló Zsolt

Áttekintés

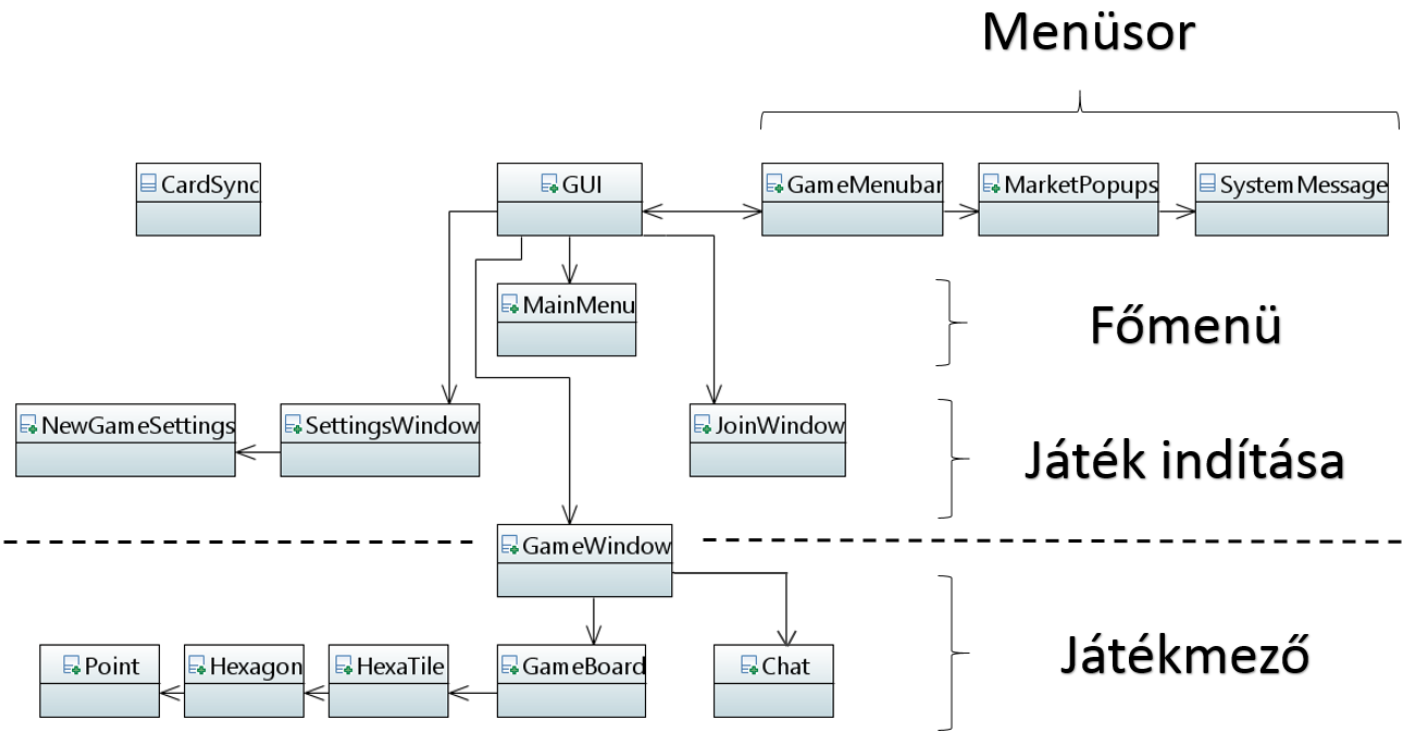
A játékot egy szerveren futó alkalmazásként képzeljük el. Ez azt jelenti, hogy az a játékos, aki létrehozza a játékot, elindít egy szervert és egy klienst. A többi játékos csatlakozáskor csak egy klienst indít el. A létrehozó játékos azért indít el egy helyi klienst a szerver mellett, hogy a játékosokat egységesen lehessen kezelni.

A szerver és a kliens feladata alapvetően a hálózati kommunikáció. Mind a szerver, mind a kliens tartalmaz egy kontroller referenciát, amely egy játéklogikát kezel. A létrehozó játékosnál két játéklogika lesz, egy a szerverhez és egy a klienshez kapcsolódva. A szerveren lévő játéklogika a „hivatalos”, a klienseké csak „másolat”. A GUI a kliensek játéklogikáját alapul véve rajzolja ki az aktuális állapotot.

A játék beállításait, a magic konstansokat, szrting literálokat és hasonlókat közös helyen, egy Config osztályban tároljuk.

GUI (felelős: Szappanos Miklós)

A grafikus felhasználói felület (GUI) segítségével a felhasználó eléri mindazon funkciókat, amik a játék vezérléséhez elengedhetetlenek ám a felhasználó beavatkozása szükséges hozzájuk. A funkciók származhatnak hálózat vagy játéklogika részről, ekkor a háttérben összetettebb folyamat játszódik le, de lehetnek kizárólag csak a grafikus felületre hatással lévők.



1. Ábra – A GUI osztályai és szerkezeti felépítése

A GUI alapja a GUI nevezetű JFrame-ből öröklődő osztály. Ehhez kapcsolódik a legtöbb JPanel ún. „Card Layout”-tal, azaz a kártyáknak nevezett JPanel-ből öröklődő ostályok (MainMenu, SettingsWindow, JoinWindow, GameWindow) közül mindig pontosan csak egy jelenik meg az ablakban (GUI osztályban) és ezek cserélődnek.

A főmenü (MainMenu)

Innen ágazik el a szerver indítás és a szerverhez való csatlakozás. A játékosok továbbá itt adhatják meg játék során használt nevüket.

Beállítások ablak (SettingsWindow)

Ebben az ablakban lehet korlátot szabni a csatlakozó játékosok számának. Ez az ablak a szerver oldali felhasználó számára érhető el, a beállítások elvégzése után jön létre a szerver.

Játékhoz csatlakozás (JoinWindow)

Megadott IP című szerverhez lehet csatlakozni.

Játékablak (GameWindow)

A „Card Layout” mindaddig érvényben van (értsd a képernyőn lévő változások mindig egy-egy JPanel váltást jelentenek), amíg a tényleges játék el nem indul (a pillanat, amikor kirajzolódik a mező). A

GameWindow még Card Layout segítségével hívható meg, azonban a benne lévő GameBoard már más logika szerint működik.

GameBoard

A GameBoard nevezetű (szintén JPanel-ből öröklődő) osztály tartalmaz minden grafikus megjelenítést, így ez a panel marad érvényben a játék indításától (amikor minden játékos csatlakozott) egészen a játék befejezéséig. A GameBoard frissíti a kirajzolt grafikus objektumokat minden esemény hatására, továbbá (a Config-ban beállított érték alapján) periodikusan magától is frissül.

Chat

A GameWindow része egy chat ablak is, melyen keresztül a játékosok üzeneteket válthatnak egymással, továbbá kaphatnak rendszerüzeneteket a játéktól. (Részletesebben a hálózat résznél.)

Menüsor

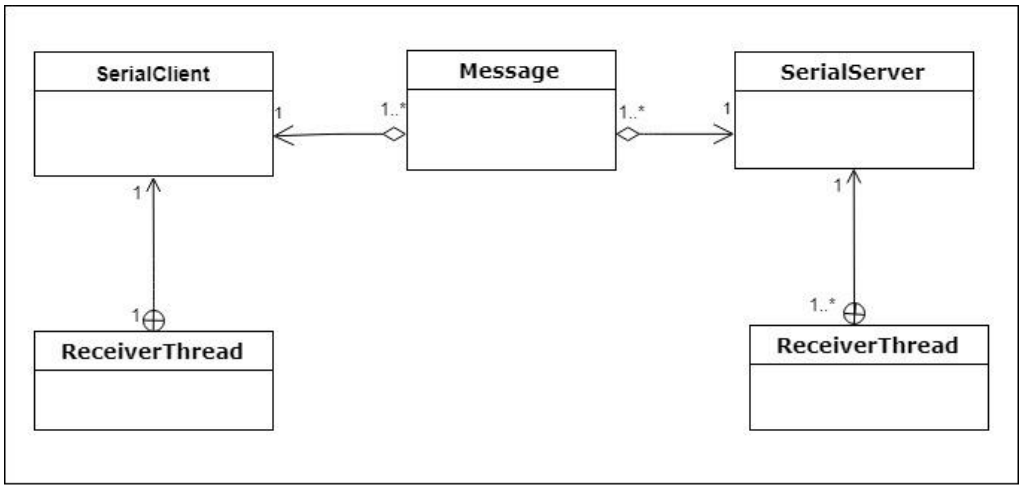
A GUI JFrame egy fontos kiegészítője a menüsor, ami egy JMenuBar-ból öröklődő osztály. Ezen keresztül van lehetőség a piachoz szükséges menüpontokat elérni (MarketPopups osztály) és a játékosok további fontos, személyre szóló üzeneteket kaphatnak (SystemMessage osztály).

Kártyák közötti szinkronizáció (CardSync)

A Card Layout egyik jellemzője az, hogy előre beállított, statikus panelek között vált. Előfordulhat azonban olyan eset is, amikor valamit meg kell változtatni az egyik (már előre elkészített, elrendezett) panelen. Erre példa egy olyan egyszerű eset, amikor a játékos megadja a nevét a főmenüben és a következő menü már azzal a szöveggel fogadja, hogy „Hello <játékos neve>!”. Arra is szükség lehet, hogy a panelek egymás között információkat cseréljenek (lásd előbbi név beállítós példát: MainMenu név string → JoinWindow). Ezen okokból jött létre a CardSync nevű kártyák közötti információáramlást és szinkronizációt biztosító osztály. Ez az osztály egyben arra is felhasználható, hogy a játéklogikában szereplő állapotokat lekérdezzük és nyomon kövessük.

Hálózat (felelős: Dányi Péter)

A játékot, ahogy a specifikációban is szerepelt hálózaton is lehet játszani. A következő UML osztálydiagram szemlélteti mely osztályokat implementáltam a hálózati kommunikáció eléréséhez. (a jobb áttekinthetőség miatt a metódusokat és tagváltozókat nem ábrázoltam.



A szerver és kliens

A szerver és kliens TCP/IP protokollon keresztül kommunikál egymással, socket-ek segítségével. A szerveren egy külön socket van arra, hogy fogadja a kliensek kapcsolódási kérését. Amikor a kapcsolat létrejön, a szerver egy másik socket-et kap, amellyel már a konkrét kliensnek tud adatot küldeni és tőle fogadni.

A kliensek kapcsolódási kérésére, majd az üzenetek fogadására egy belső szálat hoztam létre. Értelemszerűen, a szerveren több ilyen szál van, amelyeket egy tömbben tároltam.

Az üzenetek

A *Message* osztály feladata az átküldendő adat becsomagolása. Két tagváltozóból áll: típus és adat. Az adat egy általános *Object* osztályú, amely cast-olható a típus ismeretében. Az üzenetek típusai:

- azonosító (a kommunikáció elején szükséges)
- név – játékoslista frissítéséhez
- szöveg – chat funkcióhoz
- akció – minden esemény a játékban egy akciót generál

Az osztálynak továbbá szerializálhatónak kell lennie, hogy bájtfolyammá tudjuk alakítani küldéskor.

A chat

A chat lényege, hogy a felhasználó által küldött szöveges üzenet eljusson mindenkihez. Ez úgy valósult meg, hogy egy adott chat ablakban kiküldött üzenetet a kliens elküldi a szervernek, majd az továbbítja minden kliensnek. Az érkező üzenetek figyeléséhez és megjelenítéséhez egy szálat hoztam létre.

Kapcsolat a játékkal

A játék része a chat ablak (játéktábla mellett helyezkedik el), ahol a játékosok egymással tudnak beszélni, egyeztetni a kereskedési szándékukat. Minden esemény a szerverben lévő *ServerController* osztályhoz jut, amely megpróbálja végrehajtani az akciót, kivétel esetén hibaüzenetet kap a játékos. Sikeres végrehajtás esetén minden kliens végrehajtja azt a saját játékállapotán.

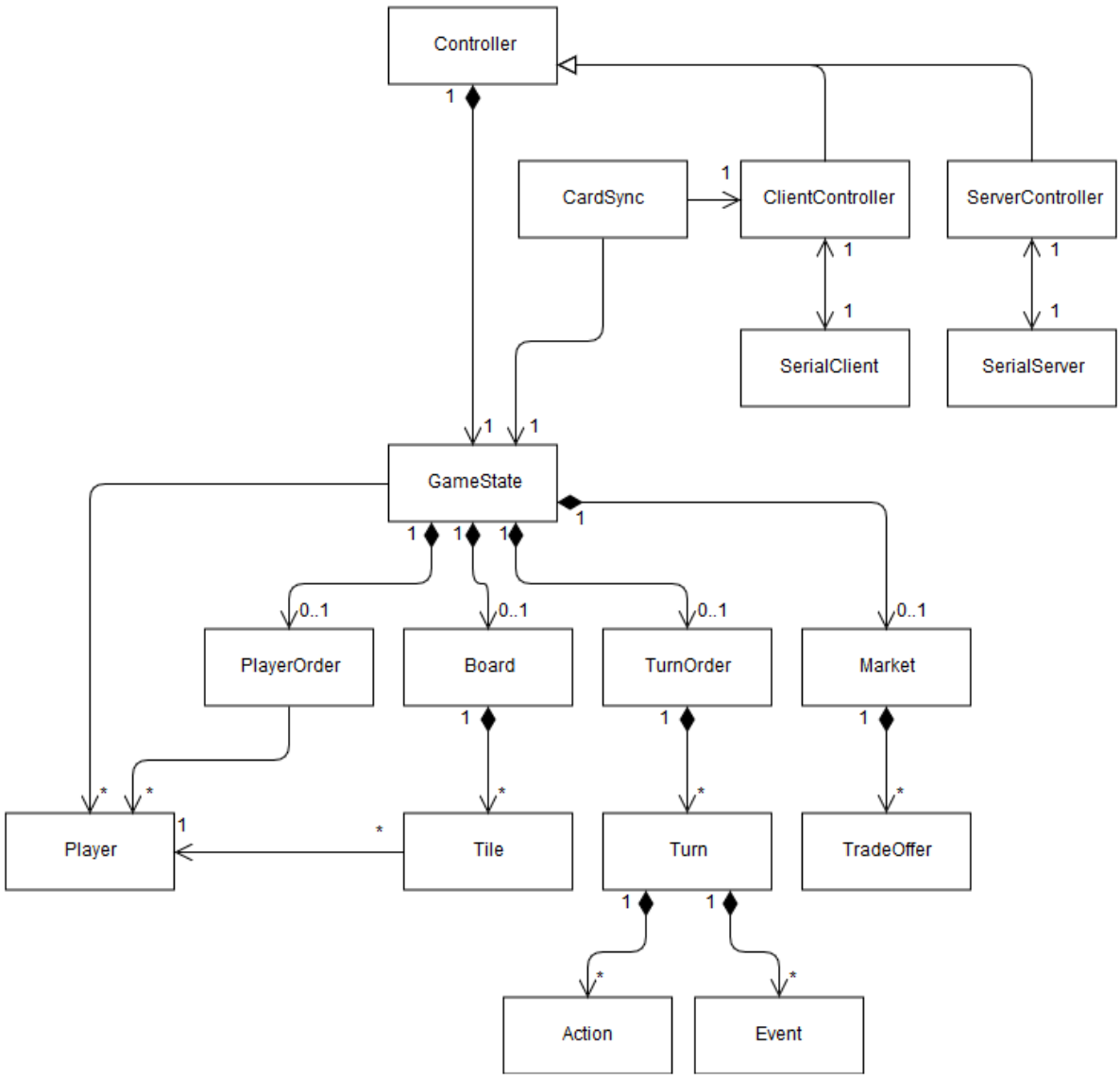
Játéklogika (felelős: Palkó András)

Áttekintés

A játéklogika feladata a játék belső állapotának, állapotváltozásainak kezelése. Az 2. ábra mutatja a játéklogika áttekintését. A játéklogika központi eleme a GameState. Ez az osztály tartalmazza a játék aktuális állapotát.

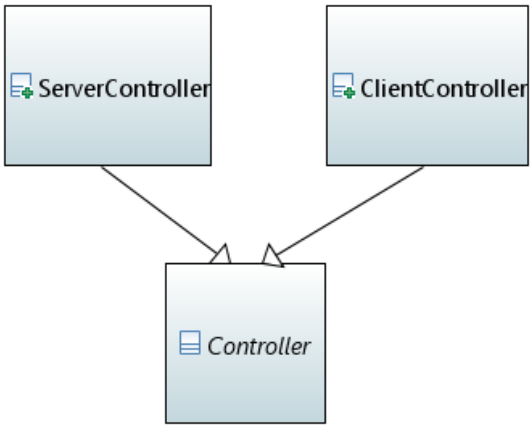
A GameState-re referenciát tárolunk a Controller osztályban. A Controller osztály biztosítja a kapcsolatot a szerver/kliens és a GameLogic között. Ennek megfelelően létezik ServerController és ClientController, melyek tartalmaznak egy-egy referenciát rendre a szervert és klienst megvalósító osztályokra.

A GUI-val való kapcsolódást a CardSync biztosítja. Ez egyrészt tartalmaz egy referenciát a kliens GameState-jére, amely alapján rajzol, másrészt a ClientController-re, amely az állapotváltozásokért felelős. A ClientController tudja továbbá megmondani, hogy az épp aktív játékos a helyi-e.



2. ábra: játéklogika áttekintés

A kontroller leszármazási hierarchiát mutatja a 3. ábra: kontroller hierarchia.



3. ábra: kontroller hierarchia

A játék állapota (GameState) több, jól szeparálható részből áll. Egyrészt, a játéktábla (Board) és a mezők (Tile) állapota. Másodszor, a játékosok sorrendje és az aktuális játékos kiléte (PlayerOrder). Továbbá, a piac (Market) állapota, vagyis a benne lévő kereskedelmi ajánlatok (TradeOffer). Végül, a körökre (TurnOrder), az aktuális körre (Turn) vonatkozó információk, az automatikus műveletekkel (Action) és kötelező eseményekkel (Event) együtt.

Játéktábla

A játék hatszöges mezőkön zajlik, ehhez illeszthető egy 60°-os koordinátarendszer úgy, hogy a mezők középpontja az egész koordinátákon helyezkedik el. A mezőknél el kell tárolni, hogy milyen nyersanyag (Resource), építettség (BuildingLevel) jellemző rájuk, illetve, hogy melyik játékos (Player) a tulajdonosa.

A táblát kell tudni generálni egyrészt erőforrás valószínűségek alapján véletlenszerűen, másrészt egy adott erőforráselrendezést újra le kell tudni generálni. Előbbit a játék indításánál a szerveren, utóbbit a klienseken alkalmazzuk. Utóbbihoz egyéni szerializálót és deszerializálót készítek.

A táblánál végezhető el könnyen az is, hogy egy adott játékosnak adjunk nyersanyagokat az általa birtokolt mezők alapján. A mezők elfoglalásakor a játéksok pontszáma is frissítendő, és mivel a pontszám csak a birtokolt mezőkből származik, ezzel a játékosok pontszámának kezelését megoldottuk.

Játékosok

Minden játékosra el kell tárolni az azonosítóját, a nevét, az általa birtokolt nyersanyagokat és a pontszámát. A játékállapotnak tartalmaznia kell egy listát a játékosokról, továbbá egy listát a játékosok sorrendjéről.

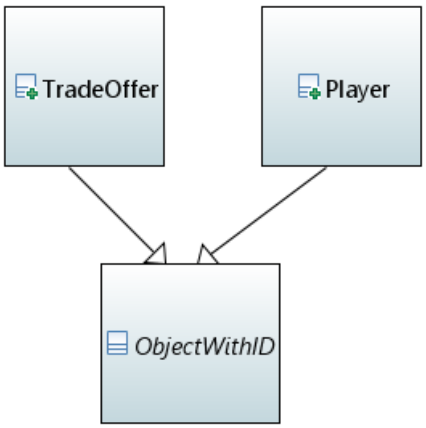
A sorrend igazából a játékos lista összekeverve. A PlayerOrder osztály tartalmazza a sorrend listát, meg tudja mondani az aktuális játékost, és tud váltani a következő játékosra. A következő játékosra váltás esetén meg kell különböztetni, hogy körbeértünk-e vagy sem, erre azért van szükség, mert ha körbeértünk, akkor a következő körbe kell átlépnünk. Ezt a megkülönböztetést egy visszatérési értékkel megoldhatjuk.

A sorrendet kell tudni generálni egyrészt véletlenszerűen (a szerveren), másrészt adott sorrendet újra le kell tudni generálni (a klienseken). Utóbbihoz egyéni szerializálót és deszerializálót készítek.

Piac

A piac (Market) tárolja a kereskedelmi ajánlatokat (TradeOffer). Minden kereskedelmi ajánlatnál el kell tárolni, hogy ki akar cserélni milyen nyersanyagból mennyit milyen nyersanyagból mennyire.

A kereskedelmi ajánlatok azonosítására szükséges egy azonosító. Mivel a játékosoknak is van azonosítójuk, ezért ezeknek közös ősosztálya (ObjectWithID) lesz, amely az automatikusan növekedő azonosítót kezeli. A leszármazási hierarchiát mutatja a 4. ábra.



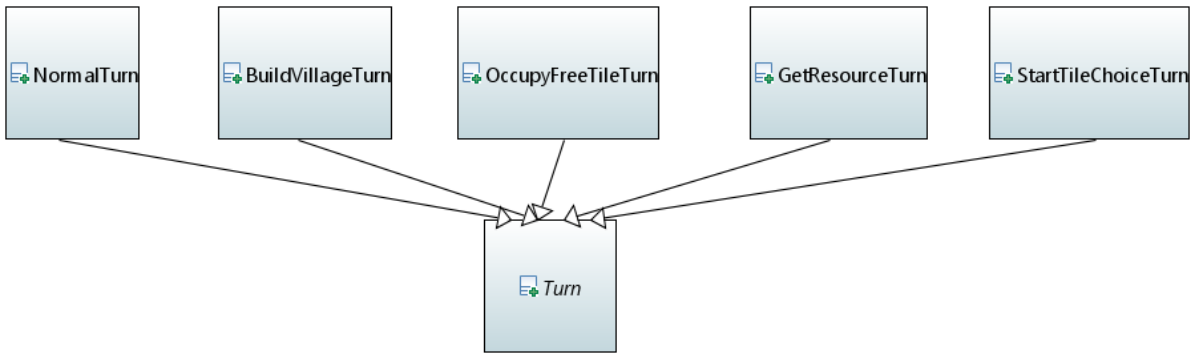
4. ábra: ObjectWithID hierarchia

Körök

A játékállapot egyik legfontosabb része a körök tárolása (TurnOrder). Egy kör (Turn) meghatározza, hogy engedélyezett-e a kereskedelem, milyen automatikus akciók és kötelező események vannak, milyen akciók engedélyezettek, illetve mennyi ideje van még az aktuális játékosnak. A kört minden játékos kezdésekor alaphelyzetbe kell állítani. Ha egy körben minden játékos volt már, akkor a következőre kell váltani. A játék vége észrevehető, ha a következő körre váltáskor a körök elfogytak.

A körsorrend is létrehozható kör lista alapján (a szerveren), illetve készül hozzá egyedi szerializáló és deszerializáló a kliensekre való átvitelhez.

A játék előkészítő és fő szakasza is feloszthatóra körökre, ezért többféle körre lesz szükség. A leszármazási hierarchiát mutatja a 5. ábra.



5. ábra: kör hierarchia

A StartTileChoiceTurn a kezdő mező választásáról szól: a játékosok kiválasztják az egyetlen kezdő mezőjüket, mindezt ingyen.

Az OccupyFreeTileTurn az előkészítés során szükséges terjeszkedésről szól, itt minden játékos ingyen elfoglal néhány szomszédos, szabad mezőt.

A BuildVillageTurn a kezdő falu felépítését tartalmazza, mindezt ingyen.

A GetResourceTurn körben a játékosok előre meghatározott nyersanyagcsomagot kapnak, ezzel befejeztük az előkészítő szakasz köreit. Ezek egyikében sem lehet kereskedni. Az előkészítési körök tartalmaznak kötelező eseményeket, például el kell foglalni egy szabad mezőt.

A NormalTurn a játék fő szakaszának egy körét írja le, a 7 nappal, költséges műveletekkel, kereskedéssel együtt. A kör elején minden játékos a mezőiről származó nyersanyagokat automatikusan megkapja.

Akciók

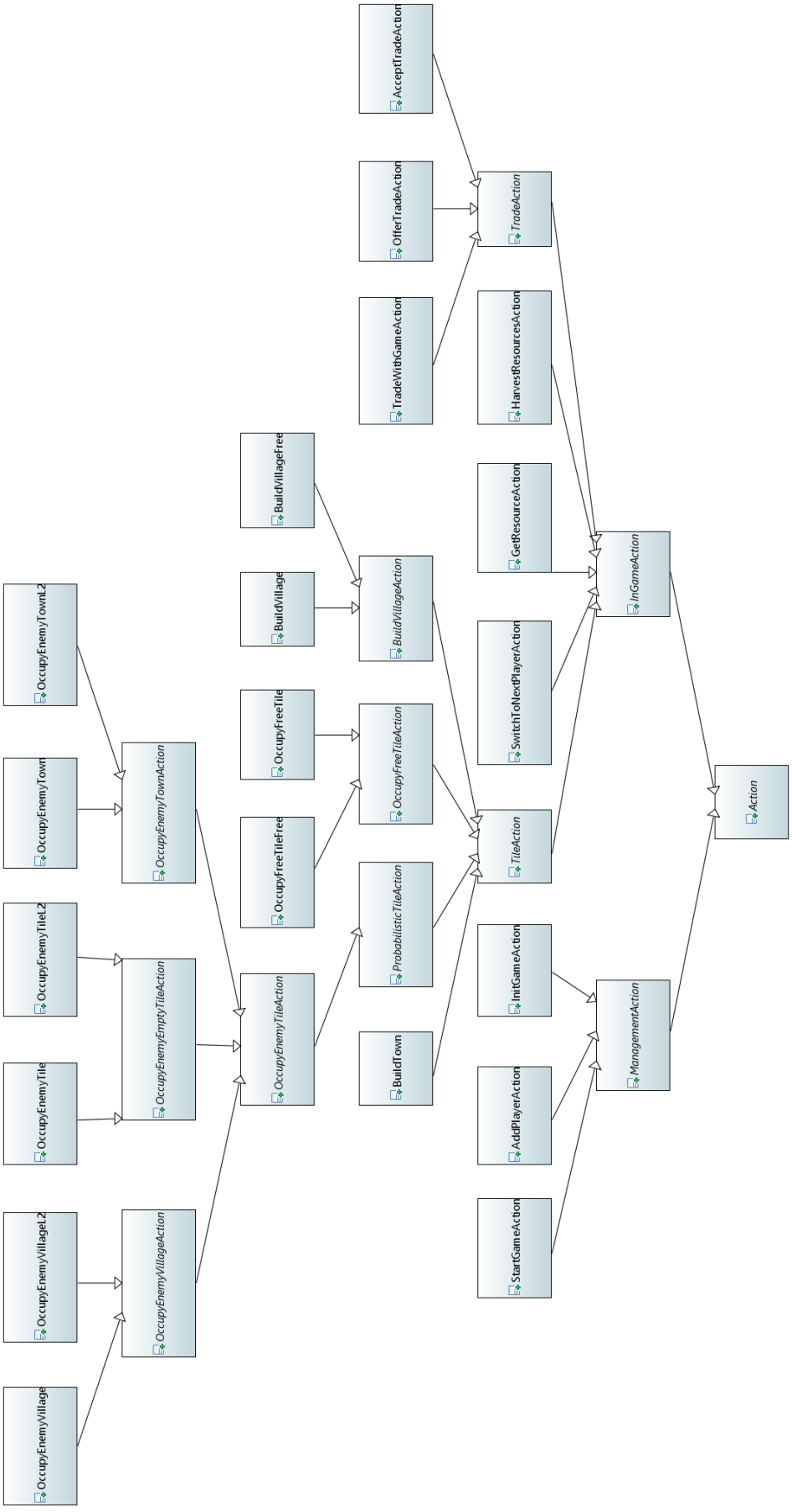
Minden, ami a játékállapotot változtathatja, egy akció (Action). Az akció végrehajtható egy játékállapoton. Az akció végrehajtását mindig ellenőrzés előzi meg, hogy az akció érvényes-e. Például érvénytelen egy akció, ha az aktuális játékos a nyersanyag költségét nem képes megfizetni egy másik játékos mezőjének elfoglalásának. Érvénytelen akció esetén saját kivétel (GameLogicException egy megfelelő leszármazottja) dobódik.

Az akciók felosztható játék előtti (ManagementAction) és közbeni (InGameAction) akciókra. A játék előtti akciók a játékos hozzáadása (AddPlayerAction), a játék inicializálása a szerveren (InitGameAction) és a játék elindítása a klienseken (StartGameAction).

A játék közbeni akciók lehetnek váltás a következő játékosra (SwitchToNextPlayerAction), adott nyersanyagból adott mennyiség megkapása (GetResourceAction), az elfoglalt mezőkből származó nyersanyagok begyűjtése (HarvestResourcesAction), kereskedelmi akció (TradeAction) vagy adott mezőhöz kapcsolódó akció (TileAction).

A kereskedelmi akciók lehetnek kereskedelmi ajánlat feladása, elfogadása vagy kereskedelem a játékkal. A mezőkhöz kapcsolódó akciók lehetnek (üres vagy ellenséges) mező elfoglalása és építkezés. A játék előkészítési szakaszában is használt akcióknak létezik ingyenes változata. Minden, a specifikációban meghatározott művelethez létezik akció.

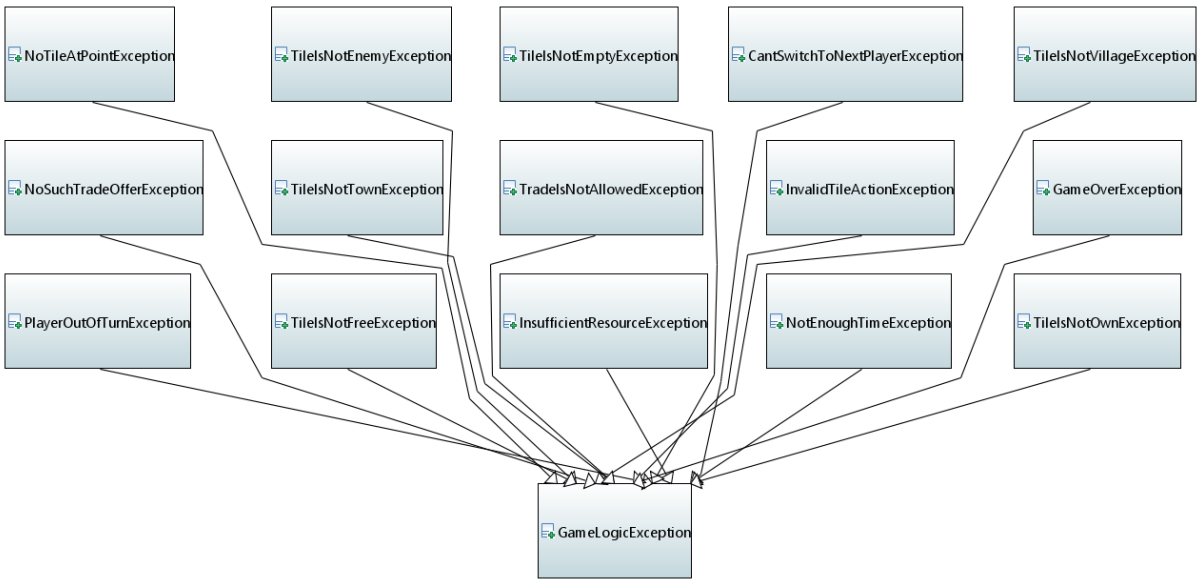
Mivel a játékállapot állandó, ha a játékosok nem csinálnak semmit, ezért elegendő a hálózaton az akciókat küldözgetni, és ha szinkronban voltak a játékállapotok, akkor az akció végrehajtása után továbbra is szinkronban maradnak. Ezért az akció szerializálható.



6. ábra: akció hierarchia

Kivételek

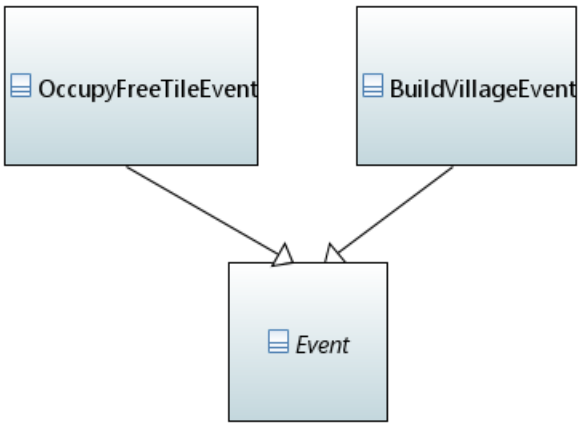
Ha egy akció érvénytelen, akkor a végrehajtásakor a GameLogicException egy megfelelő leszármazottja dobódik, hogy miért érvénytelen. A kivételeket mutatja a 7. ábra.



7. ábra: kivétel hierarchia

Események

A játék előkészítésénél előfordul, hogy a játékosnak meg kell csinálnia valamit, például elfoglalni egy szomszédos mezőt, de lényegtelen, hogy melyiket. Az ilyen „akció mintákat” tartalmazzák az események. Az eseményeket mutatja a 8. ábra. Lehetséges események a szabad mező elfoglalása, illetve falu építése.



8. ábra: esemény hierearchia

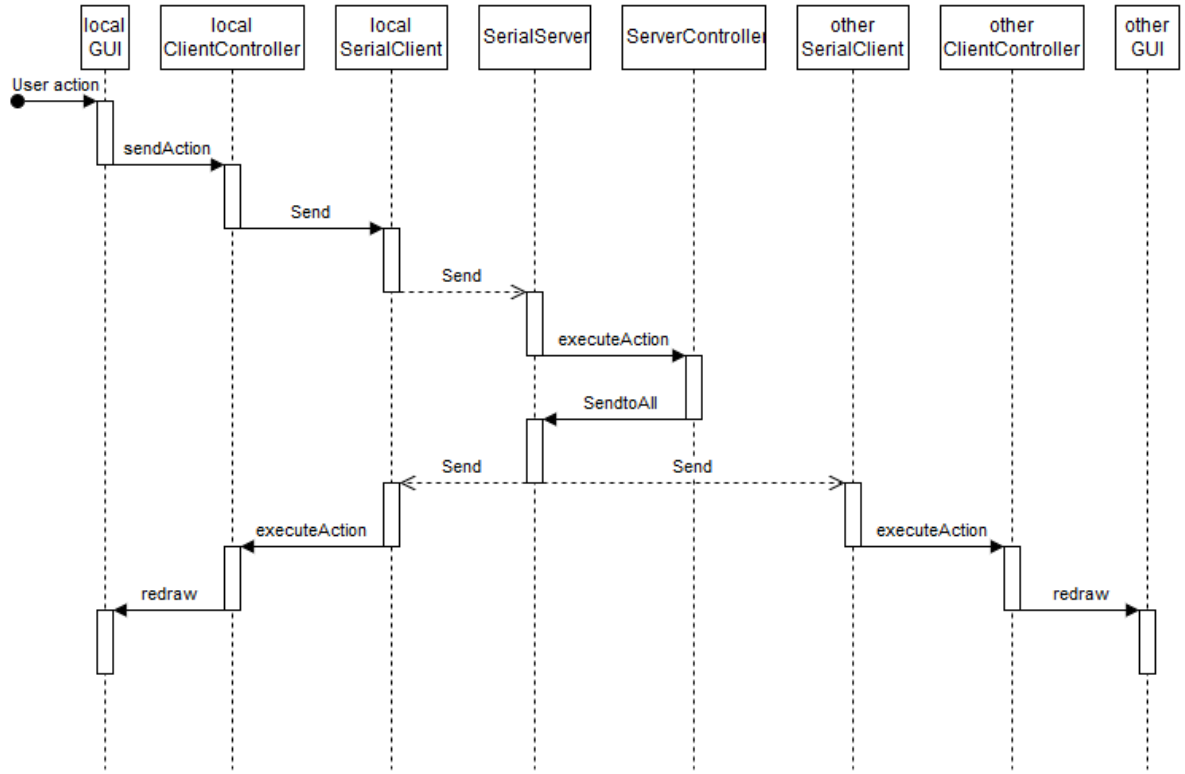
Az események tudják ellenőrizni, hogy egy akció teljesíti-e őket vagy sem. A körök tartalmazhatnak kötelező eseményeket, és addig nem lehet a következő játékosra váltani, amíg ezek nem teljesülnek.

Akció végrehajtási modell

Az akció végrehajtásának modelljét mutatja a 9. ábra. Ez úgy kezdődik, hogy a helyi játékos a helyi GUI-n valamit csinál, például kiválasztja, hogy egy mező el szeretne foglalni. Ennek megfelelően a GUI meghívja a helyi ClientController sendAction függvényét, ami elküldi az akciót a szerverre (Send). Fontos, hogy a helyi kliens itt még nem hajtja végre az akciót.

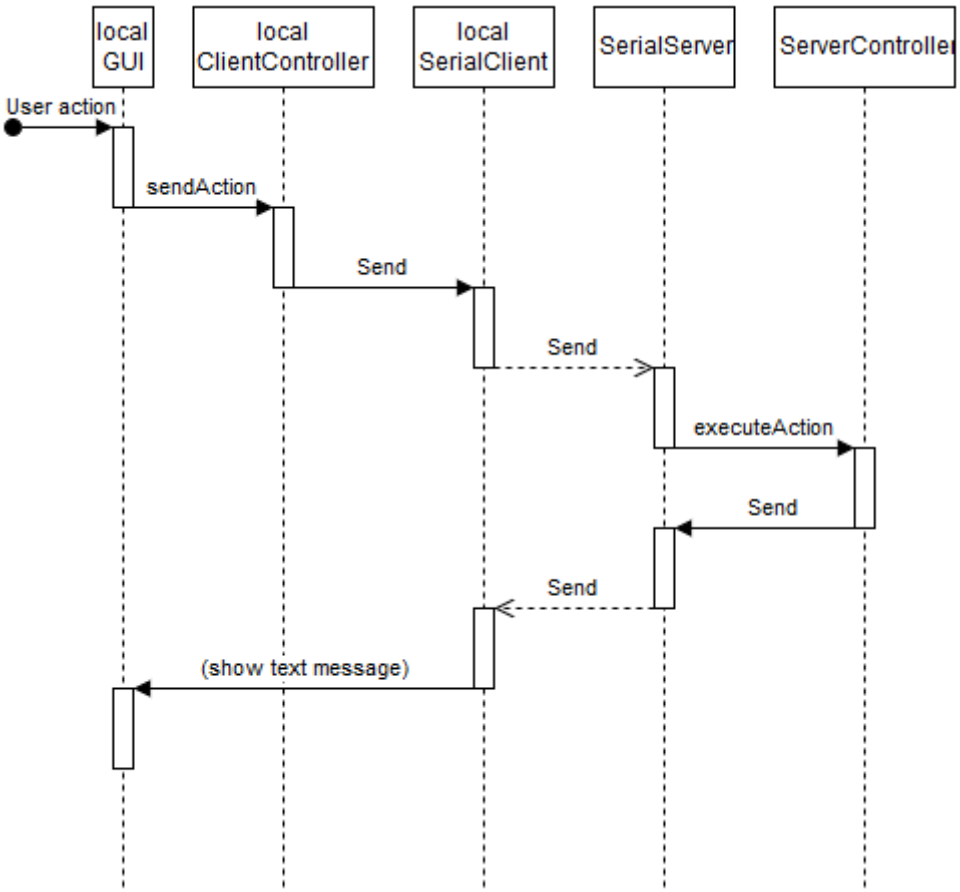
A szerver megkapja az akciót, amit a saját játékállapotán végrehajt. Ez abból áll, hogy ellenőrzi, hogy az akció érvényes-e, ha igen, akkor végrehajtja, majd ellenőrzi, hogy ez teljesítette-e valamelyik kötelező eseményt. Ha igen, akkor a kötelező eseményt. Továbbá ellenőrzi, hogy az adott játékos tud-e még bármit csinálni a körben, és ha nem, akkor továbblépteti a kört a következő játékosra.

Ezután a szerver elküldi az akciót az összes kliensnek végrehajtásnak, a helyi kliensnek is. A kliensek végrehajtják az akciót (aminek érvényesnek kell lennie, mivel a játékállapotok szinkronban voltak az akció végrehajtása előtt és a szerveren érvényes volt az akció), így a játékállapotok újra szinkronba kerülnek. Végül a módosított játékállapotot a GUI-k kirajzolják.



9. ábra: normál akció végrehajtás

Ha a szerver az akciót érvénytelennek találja, akkor a végrehajtás megszakad, a játékállapot nem módosul. A kapott kivétel alapján valamilyen hibaüzenetet küld vissza az akciót küldő játékos részére. Ezt mutatja a 10. ábra.



10. ábra: érvénytelen akció végrehajtás

Aktív játékos váltás

Az aktív játékos az, aki épp soron van. Az aktív játékos körének kezdetén az aktuális kört alaphelyzetbe kell állítani, majd végrehajtani az automatikus akciókat, ha vannak (például normál körben a nyersanyagok begyűjtése a mezők után). Ezután ellenőrizni kell, hogy a játékos tud-e mit csinálni, és ha nem, akkor tovább kell léptetni a következő játékosra.

A játék továbbléptethető a következő játékosra, ha az automatikus akciókat végrehajtottunk, és a kötelező események teljesültek. Ekkor akár a játékos kérésére is tovább lehet lépni a következő játékosra.

Amennyiben a játék továbbléptethető, és az aktív játékos nem tud semmit sem csinálni, akkor a játékot automatikusan továbbléptetjük a következő játékosra.

Az aktív játékos körének végén (a következő játékosra léptetéskor) amennyiben a játékosok körbeértek, a következő kört kell venni.

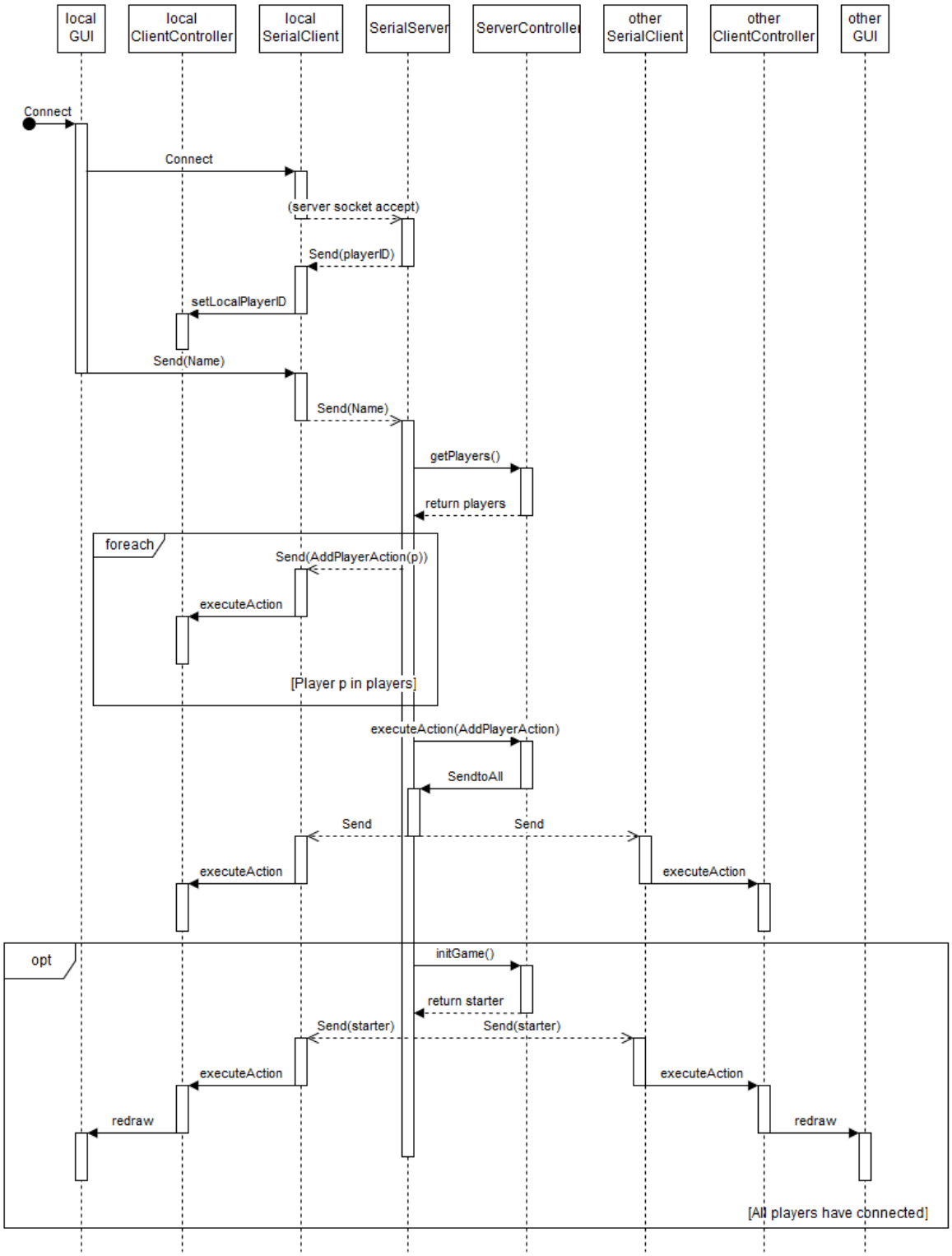
Játékosok csatlakozása, a játék indítása

Eddig mindig feltételeztük, hogy van egy futó, szinkronizált játékállapotunk. Ezt a játékosok csatlakozásánál kell felépíteni. A folyamatot a 11. ábra tartalmazza.

Amikor egy játékos csatlakozik, akkor erre válaszul a szerver visszaküld egy PlayerID-t, ez lesz a helyi játékos azonosítója. Ezzel a helyi ClientController beállítja a helyi játékos azonosítóját.

A csatlakozás után a helyi kliens elküldi a helyi játékos nevét a szerverre. Ez alapján a szerver létrehoz egy Player-t a játékos nevével és azonosítójával. Továbbá, lekérdezi az eddig csatlakozott játékosok listáját. Ebből a listából mindegyik játékoshoz készít egy AddPlayerAction-t, amit az újonnan csatlakozó játékos részére elküld végrehajtásra. Ezeket az újonnan csatlakozó játékos végrehajtja, és így szinkronban lesznek a játékállapotok a szerver és a kliensek között (de még nem tartalmazzák az újonnan csatlakozott játékost). Most már csak az újonnan csatlakozott játékoshoz kell létrehozni egy AddPlayerAction-t, amit a szerveren és az összes kliensen végre kell hajtani.

Amennyiben az összes játékos csatlakozott, elindítható a játék. Ez a ServerController initGame() függvényével történik, ami a szerver játékállapotot inicializálja, és visszaad egy StartGameAction-t, amit a klienseken végre kell hajtani. Ezt a visszaadott StartGameAction-t a kliensek végrehajtják, és ezzel elkezdődik a játék.



11. ábra: játékosok csatlakozása