

# **Követelmény, projekt, funkcionálitás**

**77 – gods\_of\_jar**

Konzulens:  
**Vörös András**

## **Csapattagok**

Réti Ádám	(AO7JX4)	ket.vill.adam@gmail.com
Tisza Miklós	(E1LX0J)	tisza.miklos.16@gmail.com
Czifra Barnabás	(NX9GA4)	barna.czifra@gmail.com
Molnár Márton	(KLM6OO)	molnar.marton.hun@edu.bme.hu
Kopach Artem	(JQBOLI)	kopach.artem@edu.bme.hu

2023.03.11

## 2. Követelmény, projekt, funkcionalitás

### 2.1 Bevezetés

#### 2.1.1 Cél

A dokumentum célja a "Sivatagi vízhálózat" című laborfeladathoz kapcsolódó követelmények megfogalmazása, leírása.

#### 2.1.2 Szakterület

A kialakítandó szoftver egy társasjáték, elsődleges célja a szórakozás biztosítása.

#### 2.1.3 Definíciók, rövidítések

- Bug: Az informatikában összesítve azt jelenti ez a szó, hogy egy program hibás működést produkál, hibásan működik.
- Auditálás: Az auditálás során ellenőrzik, hogy a szoftver a megfelelően került átadásra, telepítésre és üzembe helyezésre. Az auditálás során a szoftverfejlesztő és az ügyfél közös erőfeszítése révén ellenőrzik a szoftver átadásának és üzembe helyezésének folyamatát.
- DBMS: Adatbázis kezelő rendszer rövidítése.
- OS: Operációs rendszer rövidítése

#### 2.1.4 Hivatkozások

- *Feladat / Department of Control Engineering and Information Technology (bme.hu)*
- *Követelmény, projekt, funkcionalitás / Department of Control Engineering and Information Technology (bme.hu)*

#### 2.1.5 Összefoglalás

Áttekintés: A szoftver általános áttekintése, funkcióinak leírása.

Követelmények: A szoftverrel szemben megfogalmazott követelmények, a szoftvernek mely funkciókat kell megvalósítania.

Lényeges use-casek: A felhasználók által elérhető funkciók leírása.

Szótár: A követelmények köznyelvre fordított verziót tartalmazza.

Projekt terv: A projekt lépéseit, határidejüket, a projekthez felhasznált eszközöket tartalmazza.

Napló: Tartalmazza, hogy ki mennyi ideig dolgozott a különböző anyagrészeken.

## 2.2 Áttekintés

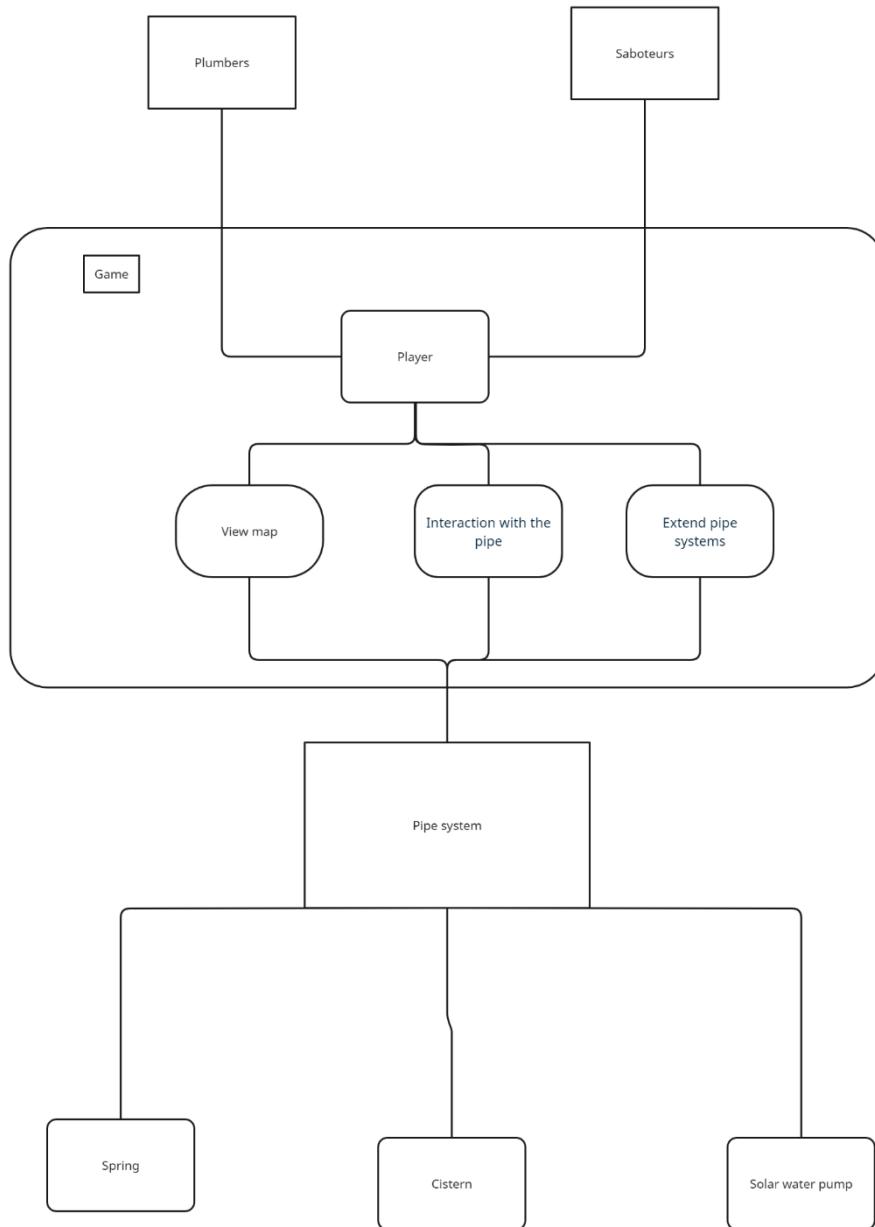
### 2.2.1 Általános áttekintés

#### A szoftverarchitektúra magas szintű áttekintése.

- Ebben a játékban több fő rendszert és alrendszert különböztethetünk meg. Az alrendszer a rendszer egy része, amely bizonyos feladatokat és funkciókat kap.
- Rendszerek - csőrendszer, mechanika, szabotőrök. Úgy tervezünk, hogy ezek a rendszerek MAIN ebben a játékban, amelynek lesznek alrendszeréi.
- A csőrendszerhez ( Pipe system ) viszont egyedi funkciókkal rendelkező alrendszerekre utalhatunk és különböztethetünk meg. Olyan alrendszer, mint a forrás (Spring), ciszterna (Cistern), napelemes vízszivattyú (Solar water pump).

- A mechanikus ( Plumbers ) és szabotőr ( Saboteurs ) rendszerhez tartozik a játékos alrendszer ( Player ), amely viszont szintén rendelkezik egy alrendszerrel ( az alrendszert ( funkcionálitást) nem kell szerepeltetni, de nélkül nem fogjuk elérni a játékos alrendszer és a csőrendszer megértését és összekapcsolását).
- A játékos alrendszerben ( Player ) a következő alrendszerket emeljük ki: a csőrendszer bővítése ( Extend pipe systems), a játékos térkép böngészése ( View map), a játékos és a cső közötti interakció ( Interaction with the pipe)

Az alábbiakban egy példa a rendszerek és alrendserek összekapcsolásának módjára:



## 2.2.2 Funkciók

A cél egy játékprogram készítése, melyet két csapatban játszanak a játékosok és amely grafikus felülettel rendelkezik. A grafikus felületen jelennek meg a játék elemei, melyek a következők:

- Vízforrás: A víz folyása innen indul.
- Ciszterna: Itt a játék során folyamatosan készülnek az új csövek, melyek segítségével tudják a szerelők bővíteni a vízhálózatot. Ide folyik be a víz, a szerelők az ide befolyt víz mennyisége alapján kapják a pontokat.
- Pumpa:
  - Több cső is csatlakozhat hozzá.
  - minden pumpához az arra jellemző, véges számú cső csatlakozhat maximum.
  - A pumpákhoz tartozik egy-egy víztartály, amelyet átmeneti tárolóként használ, ha a kifolyó víz értéke kisebb a befolyó víz értékénél.
  - A pumpák véletlenszerűen elromolhatnak, ekkor nem képesek vizet pumpálni.
  - Egy kimenete és egy bemenete lehet. A játékosok átállíthatják a bemenetet és a kimenetet is.
- Játékosok:
  - Szerelők
  - Szabotőrök
- Csövek:
  - minden csőnek van egy maximális kapacitása, amelynél több vizet nem képes szállítani.
  - A csövek egyik végét le lehet csatlakoztatni és egy másik elemhez fel lehet csatlakoztatni.
  - A csövek szabad végén a víz a sivatagba folyik, amennyiben a csőben folyik éppen víz.

A játékok két csapat játssza legalább 2-2 játékossal. A csapatok célja a játékmenet során eltér:

- Szerelők: Céljuk, hogy minél több víz folyjon a ciszternákba. Ehhez a következő eszközök állnak rendelkezésükre:
  - Pumpák megjavítása: Az elromlott pumpákat képesek megjavítani, hogy azok ismét megfelelően működjenek. Az elromlott pumpák nem képesek vizet szállítani.
  - Pumpák átállítása: Beállítják, hogy a pumpáknál melyik csőből melyikbe folyjon a víz. Céljuk, hogy a lehető legtöbb víz folyjon át az adott pumpán.
  - Csövek megfeszítése: A kilyukadt csöveket képesek megfeszíteni, hogy ismét rendesen folyjon bennük a víz.
  - Hálózat bővítése: A szerelők képesek a ciszternánál felvett új csövekkel és pumpákkal a vízhálózat bővítésére. Az új csöveket már meglévő pumpákra tudják rácsatlakoztatni. Az új pumpákat meglévő csövek közepére képesek elhelyezni oly módon, hogy a meglévő csövet kettévágják, majd az újonnan keletkezett csővégeket az új pumpához csatlakoztatják.

- Szabotőrök: Céljuk a játék során, hogy a lyukas csöveken minél több víz folyjon el a sivatagba.  
Ezt a következőképpen tudják elérni:
  - Csövek kilyukasztása: Képesek a csöveket kilyukasztani, így azon a csövön nem megy tovább a víz, hanem elfolyik a sivatagba.
  - Pumpák átállítása: A szabotőrök képesek beállítani, hogy a pumpáknál melyik csőből melyikbe folyjon a víz, így a törött csövek felé tudják irányítani a vizet.

A játék végén az a csapat nyer, amely több vizet szerez.

A játék során a játékosok csak a pumpák és a csövek mentén lépkedhetnek. A pumpákon egyszerre több játékos is tartózkodhat, így ott kikerülhetik egymást. A csöveken azonban egyszerre csak egy játékos tartózkodhat, így ott nem kerülhetik ki egymást.

### **2.2.3 Felhasználók**

- A felhasználók kortól függetlenül játszhatják ez a játékot, hiszen bárki számára szórakozást nyújthat. Ami ezen felhasználókat jellemzeti az a következők:
- Kreativitás: A játék során gyakran előfordulhat, hogy a csőhálózat bonyolultabbá válik, és az optimális vízszállítási utak megtalálása érdekében a játékosoknak kreatívan kell gondolkodniuk.
- Kommunikáció: A csapatoknak együtt kell működniük a játék során, és fontos, hogy hatékonyan kommunikáljanak egymással. Az együttműködés kulcsfontosságú, hogy a lehető legtöbb víz folyhasson a ciszternákba.
- Stratégiai gondolkodás: A játékosoknak a stratégiai gondolkodásra van szükségük ahhoz, hogy a csőhálózatot optimálisan építsék fel/alakítsák át, és hatékonyan használják a pumpákat, hogy minnél több vizet szállítsanak

### **2.2.4 Korlátozások**

- A szoftvernek a következőket kell tudnia:
- Teljesítmény: Hatékonyan és gyorsan kell működnie, még akkor is, ha nagy mennyiségű adatot kell feldolgozni.
- Biztonság: Megfelelő biztonsági intézkedésekkel kell rendelkeznie, például a felhasználói jogosultságok kezelésével, adatvédelemmel és a támadások elleni védelemmel.
- Kompatibilitás: Kompatibilisnek kell lennie a különböző operációs rendszerekkel, böngészőkkel és más szoftverekkel, amellyel együtt kellhet működnie.
- Megbízhatóság: Megbízhatónak kell lennie, és minimális hibaszázalékkal kell működnie.
- Karbantarthatóság: Karbantarthatónak kell lennie, és könnyen javíthatónak, ha esetleges hibák (bugok) merülnek fel.
- Skálázhatóság: Skálázhatónak kell lennie, és képesnek kell lennie az erőforrások rugalmas használatársa, ha a felhasználói igények megváltoznának.

### **2.2.5 Feltételezések, kapcsolatok**

A hivatkozásokban felsorolt anyagok, a konkrét feladatot (a játék leírását) tartalmazzák, amit majd a játékfejlesztőknek kell megvalósítani. Emellett még megtalálható, a dokumentumnak a kiindulási állapota.

## 2.3 Követelmények

### 2.3.1 Funkcionális követelmények

#### 2.1 Elsődleges Követelmények

Azonosító	Leírás	Use-case	Ellenőrzés	Prioritás	Forrás
R01	A drukmákorú sivatagon át bonyolult csőrendszer szállítja a vizet a hegyi forrásokból a sivatagon túl elterülő városok ciszternáiba	Water Flowage	A hegyi forráshoz megfelelő csövet csatlakoztatunk, amellyel elkezdődik a víz folyása	Alapvető	Feladatleírás 1.bekezdés 1.sor
R02	A csőrendszer egyszerű, elágazás nélküli csövekből és napelemmel működő pumpákból áll.	View Map	A játék elindulása után megfigyelhetjük, hogy az általunk választott játékos ténylegesen belátja-e az egész pályát	Alapvető	Feladatleírás 1.bekezdés 2.sor
R03	Egy pumpa több csövet is összeköthet, hogy melyik belekötött csőből melyik másikba pumpáljon	Adjust Pump	Egy játékos az Adjust Pump funkciót használva a pumpára megfigyelhető, hogy egy másik csőből (amelyben folyik a víz) átfolyik-e a víz a másik választott csőbe a pumpa módosítása után	Opcionális	Feladatleírás 1.bekezdés 3.sor
R04	A többi rákötött cső eközben el van zárva	Water Flowage	Ha egy pumpával két csövet összekötünk csak a két összekötött csőben kell, hogy folyjon a víz másik a pumpához kötött csőben nem	Alapvető	Feladatleírás 1.bekezdés 5.sor
R05	A pumpák véletlen időközönként el tudnak romlani	Pump Breakdown	Egy játék elindítása után kellő időt várva elromlik-e egy pumpa vagy sem (teszteléshez ennek a várható idejét tudjuk)	Fontos	Feladatleírás 1.bekezdés 6.sor

<b>R06</b>	ilyenkor megszűnik az adott pumpánál a vízáramlás	<b>Water Flowage</b>	Egy pumpa elromlásánál megnézzük, hogy két cső közötti kötés megszűnik-e, azaz a forrás csőből nem megy-e több víz a cél csőbe	<u>Alapvető</u>	Feladatleírás 1.bekezdés 6.sor
<b>R07</b>	A csőhálózatot a szerelők tartják karban	<b>Repair Pump</b> <b>Repair Pipe</b>	A szerelők pumpa és cső javító tevékenységei ellenőrizhetők egy cső vagy pumpa elromlása után	<u>Fontos</u>	Feladatleírás 2.bekezdés 1.sor
<b>R08</b>	Ők javítják meg az elromlott pumpákat	<b>Repair Pump</b>	Az elromlott pumpára állva a szerelő elvégzi a pumpa javításának tevékenységét, és megnézzük, hogy ez elromlott állapotból működőképes állapotba viszi-e át	<u>Fontos</u>	Feladatleírás 2.bekezdés 1.sor
<b>R09</b>	ők állítják át a pumpákat, hogy mindenkor a lehető legtöbb víz tudjon áthaladni a hálózaton	<b>Adjust Pump</b>	A pumpa javításához hasonlóan ezt is a pumpára állva tesztelhetjük le, hogy a cselekvést elvégezve megtörtént-e a cél cső megváltoztatása	<u>Opcionális</u>	Feladatleírás 2.bekezdés 1.sor
<b>R10</b>	és ha egy cső kilyukad, az Ő dolguk a cső megfeszítése is	<b>Repair Pipe</b>	A szerelő játékossal a csőre rálépve a cső javítását elvégezve megtörténik-e a csőnek a javítása, azaz lyukas állapotból működőképes állapotba kerül-e	<u>Fontos</u>	Feladatleírás 2.bekezdés 2.sor
<b>R11</b>	A kilyukadt csövekből a víz kifolyik, a csövek végén lévő pumpához már nem jut belőle	<b>Water Flowage</b>	Amennyiben egy cső ki lett lyukasztva egy szabotőr által megnézzük, hogy a csőből további csőbe/pumpába már nem folyik tovább a víz	<u>Alapvető</u>	Feladatleírás 2.bekezdés 3.sor

R12	Mivel a sivatag veszélyes hely, a szerelők csak a csöveken és a pumpákon haladhatnak. A pumpáknál kikerülhetik egymást, de a csöveken már nem tudnak elmenni egymás mellett	<b>Move</b> <b>View Map</b>	2 játékos segítségével egyszerűen tesztelhetjük, hogy: 1. képesek-e egymás mellett állni a csövön? ha igen akkor javítani  2.képesek-e egy pumpán együtt álni?  3.képesek-e egymás mellett elmenni a csöveken?	<u>Alapvető</u>	Feladatleírás 4.bekezdés 1.sor
R13	A hálózaton élnek a nomád szabotörök is, akik a pumpákat tudják átállítani és a csöveket szokták kilyukasztani	<b>Adjust Pipe</b> <b>Leak Pipe</b>	Egy szabotőr játékost választva megnézzük, hogy képesek vagyunk-e egy csövet kilyukasztani, ha igen változott-e a cső állapota működőképesből lyukasztottra?	<u>Fontos</u>	Feladatleírás 3.bekezdés 1.sor
R14	A csövek kellően rugalmasak ahhoz, hogy az egyik végüket lecsatlakoztatva	<b>Detach Pipe</b>	Bármely fele játékossal játszva letudunk-e választani szabadvégű csöveket	<u>Opcionális</u>	Feladatleírás 1.bekezdés 9.sor
R15	egy másik aktív elemhez elvihetők és ott felcsatlakoztathatók legyenek	<b>Attach Pipe</b>	Bármely játékost típust kiválasztva képesek vagyunk-e egy szabadvégű csőhöz egy nálunk lévő csövet illeszteni	<u>Opcionális</u>	Feladatleírás 1.bekezdés 9.sor
R16	A mozgásuk ugyanúgy a csőhálózathoz kötött, mint a szerelőké	<b>Move</b>	Hasonlóan ellenőrizzük, mint a szerelők mozgását	<u>Alapvető</u>	Feladatleírás 3.bekezdés 2.sor
R17	A szabotörök dolga, hogy minél több víz folyjon el a lyukakon, a szerelők pedig azon dolgoznak, hogy minél több víz jusson a ciszternába	<b>Water Flowage</b>	Itt nincsen ellenőrizhető cselekvés	<u>Alapvető</u>	Feladatleírás 5.bekezdés 1.sor

## 2.2 További követelmények

Azonosító	Leírás	Use-case	Ellenőrzés	Prioritás	Forrás
R18	Ha a szabatőr olyan csövet lyukaszt ki, amelyben nem volt víz akkor a szabatőr ne szerezzen pontokat	<b>Leak Pipe</b> <b>Water Flowage</b>	Egy szabatőr játékost választva kilyukasztunk egy olyan csövet, amelyben nincsen víz, ekkor a szabatőr csapat pontjának nem kéne emelkednie	<u>Alapvető</u>	Mivel a feladatleírás ezt explicit nem tartalmazza ezért itt nincsen megjelölhető forrás
R19	A játék véget ér, ha a hegyi forrásból kifolyt az összes víz	<b>Water Flowage</b>	Egy játékot végigjátszva megfigyeljük, hogy amint a hegyi forrásból kifogyott a víz véget ért-e a játék, azaz: 1. semelyik csapat sem kap további pontokat 2. semelyik csapat tagja se képes cselekvésre	<u>Alapvető</u>	Mivel a feladatleírás ezt explicit nem tartalmazza ezért itt nincsen megjelölhető forrás
R20	A szabatőr csak azon csövet lyukaszthatja ki, amely csövön áll mást nem	<b>Leak Pipe</b> <b>Move</b>	Ezt egyszerűen tesztelhetjük azzal, hogy egy szabatőr játékost leteszünk egy csőre és egy másik csövet megpróbálunk kilyukasztani	<u>Alapvető</u>	Mivel a feladatleírás ezt explicit nem tartalmazza ezért itt nincsen megjelölhető forrás
R21	A szerelő csak azon csövet és pumpát javíthatja meg amelyen áll mást nem	<b>Repair Pipe</b> <b>Repair Pump</b> <b>Move</b>	Hasonlóan teszteljük, mint a szabatőr speciális cselekvését	<u>Alapvető</u>	Mivel a feladatleírás ezt explicit nem tartalmazza ezért itt nincsen megjelölhető forrás

R22	A szerelő és szabotőr csak azon pumpát állíthatja, amelyen áll mást nem	Adjust Pump	Hasonlóan az előző kettő cselekvéshez ezt is ugyanúgy tesztelhetjük	Alapvető	<i>Mivel a feladatleírás ezt explicit nem tartalmazza ezért itt nincsen megjelölhető forrás</i>
-----	---	-------------	---	----------	---

### 2.3.2 Erőforrásokkal kapcsolatos követelmények

Kis komment előre: Azt tervezzük, hogy ez a játék egy konzolos játék lesz. Tehát a következő, az összes követelmény egy konzolos játékra vonatkozik. Ehhez a szakaszhoz a honlapról származó ismereteket is felhasználtuk. [https://en.wikipedia.org/wiki/Requirements\\_engineering](https://en.wikipedia.org/wiki/Requirements_engineering)

Azonosító	Leírás	Ellenőrzés	Prioritás	Forrás	Komment
RE01	Hardveres követelmények	Processzor sebesség, memória (RAM) , op.rendszer verziója , szabad lemezterület	Fontos	<a href="https://www.antsz.hu/adata/cms76137/20130820_KMR_Kovetelményjegyzek.pdf">https://www.antsz.hu/adata/cms76137/20130820_KMR_Kovetelményjegyzek.pdf</a>	<i>“A representation of a system, including a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.”</i>
RE02	Alapszoftveres követelmények	DBMS verzió, webszerver verzió, kliensoldali böngésző verzió	Alapvető	<a href="https://en.wikipedia.org/wiki/System_requirements">https://en.wikipedia.org/wiki/System_requirements</a>	

RE03	Architekturális követelmények	Szoftver komponenseinek és a rendszereknek az egymással való integrálása, adatok közti kommunikáció	Alapvető	<a href="https://en.wikipedia.org/wiki/Systems_architecture">https://en.wikipedia.org/wiki/Systems_architecture</a>	"An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline"
RE04	Logisztikai követelmények	Szoftver telepítése, karbantartása, biztonsági mentése, frissítése, (upgrade RAM, OS version and etc)	Alapvető	<a href="https://en.wikipedia.org/wiki/Upgrade">https://en.wikipedia.org/wiki/Upgrade</a>	Az új verziók megjelenése miatt változhatnak az operációs rendszer verziók követelményei, a szabad lemezterület mennyisége, a RAM-memória mennyisége stb. "Examples of common hardware upgrades include installing additional memory (RAM), adding larger hard disks, replacing microprocessor cards or graphics cards, and installing new versions of software. Many other upgrades are possible as well."
RE05	Felhasználói követelmények	Interfész kialakítása, adatok megjelenítési módja	Opcionális		

### 2.3.3 Átadással kapcsolatos követelmények

Azonosító	Leírás	Ellenőrzés	Prioritás	Forrás	Komment
D01	A szoftver telepítésének, frissítésének lépései	Telepítési teszt	Alapvető	<a href="https://hu.wikipedia.org/wiki/Telep%C3%ADt%C3%A1s_(informatika)">https://hu.wikipedia.org/wiki/Telep%C3%ADt%C3%A1s_(informatika)</a> <a href="https://en.wikipedia.org/wiki/Installation_(computer_programs)">https://en.wikipedia.org/wiki/Installation_(computer_programs)</a>	

D02	A szoftver működéséhez szükséges OS., hardver verziói	Felhasználói teszt	Alapvető	<a href="https://en.wikipedia.org/wiki/Software_requirements">https://en.wikipedia.org/wiki/Software_requirements</a>	
D03	A szoftverhez szükséges konfigurációs beállítások és azok módosítása	Felhasználói teszt	Alapvető	<a href="https://en.wikipedia.org/wiki/Configuration_management">https://en.wikipedia.org/wiki/Configuration_management</a>	“Configuration management (CM) is a process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life”
D04	A szoftver karbantartása	Auditálás	Fontos	<a href="https://en.wikipedia.org/wiki/Software_maintenance">https://en.wikipedia.org/wiki/Software_maintenance</a> <a href="https://de.wikipedia.org/wiki/Softwarewartung">https://de.wikipedia.org/wiki/Softwarewartung</a>	

### 2.3.4 Egyéb nem funkcionális követelmények

Azonosító	Leírás	Ellenőrzés	Prioritás	Forrás	Komment
NF01	A szoftvernek biztonságosnak kell lennie, és meg kell védenie az adatokat az illetéktelen hozzáféréstől vagy manipulációtól.	Rendszeres ellenőrzés, felhasználói teszt	Alapvető	<a href="https://www.jacobacci.com/en/patents/software-protection#:~:text=What%20is%20software%20protection%3F,program%20codes%20and%20graphical%20interfaces.">https://www.jacobacci.com/en/patents/software-protection#:~:text=What%20is%20software%20protection%3F,program%20codes%20and%20graphical%20interfaces.</a>	“Software protection is the protection of information that is stored on hardware and used by a computer system to perform operations”
NF02	Hordozhatóság: A szoftvernek különböző operációs rendszerekre és platformokra történő telepítést kell támogatnia.	Szoftver különböző környezetekben való tesztelése	Alapvető	<a href="https://www.techtargent.com/whatis/definition/compatibility#:~:text=For%20a%20system%20to%20be,one%20OS%20but%20not%20another.">https://www.techtargent.com/whatis/definition/compatibility#:~:text=For%20a%20system%20to%20be,one%20OS%20but%20not%20another.</a>	

NF03	Megbízhatóság: A szoftvernek stabilan kell működnie, és minimalizálni kell a hibák és meghibásodások számát.	A teszteléssel meg lehet vizsgálni, hogy a szoftver a különböző bemenetekre megfelelően reagál-e, nem vezet-e hibás működéshez, és az elvárt kimenetet adja-e.	Alapvető	<a href="https://www.linkedin.com/pulse/importance-stabilising-your-applications-samaneh-khaleghi/">https://www.linkedin.com/pulse/importance-stabilising-your-applications-samaneh-khaleghi/</a>  <a href="https://www.vm.pl/news/stable-software-means-a-stable-future-how-to-avoid-system-instability">https://www.vm.pl/news/stable-software-means-a-stable-future-how-to-avoid-system-instability</a>	Ezekből a weboldalokból azt a fontos megállapítást tettük, hogy a tesztelés lefedettsége, a hibakeresés fontosabb, mint valaha, mielőtt egy projektet kiadnánk a világba.
NF04	Tesztelhetőség: A szoftvernek tesztelhetőnek kell lennie, és biztosítani kell a megfelelő tesztelési környezetet.	Új változtatások és frissítések előtt teszteket kell végezni, hogy meggyőződjünk a változtatások helyes működéséről	Alapvető	<a href="https://testfort.com/blog/what-is-testability-in-software-testing">https://testfort.com/blog/what-is-testability-in-software-testing</a>  <a href="https://www.techtargent.com/searchapparchitecture/tip/5-key-software-testability-characteristics">https://www.techtargent.com/searchapparchitecture/tip/5-key-software-testability-characteristics</a>	
NF05	Felhasználói élmény: A szoftvernek egyszerű és felhasználóbarát felületet kell biztosítania, és a felhasználók számára könnyen elérhető kell lennie.	Felhasználói teszt	Alapvető	<a href="https://start.docuware.com/blog/document-management/what-makes-software-user-friendly">https://start.docuware.com/blog/document-management/what-makes-software-user-friendly</a>	Biztos, hogy User-friendly kell megcsinálni

## 2.4 Lényeges use-case-ek

### 2.4.1 Use-case leírások

<b>Cím</b>	<b>Move</b>
<b>Leírás</b>	A szerelők és a szabatőrök csak a csöveken és a pumpákon haladhatnak.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékosok a csöveken és pumpákon haladva mozognak
<b>Alternatív Forgatókönyv</b>	A játékos egy olyan csőre próbálna mozogni, amelyen már valaki halad, ekkor a játékos nem tud azon a csövön közlekedni
<b>Alternatív Forgatókönyv</b>	A játékosok a pumpánál már megkerülhetik egymást

<b>Cím</b>	<b>View Map</b>
<b>Leírás</b>	A játékosok látják a pályán lévő csöveket és pumpákat
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	<ol style="list-style-type: none"> <li>1. A rendszer kirajzolja a csövek és pumpák aktuális állapotát</li> <li>2. A játékos megtekinti a csövek és pumpák aktuális állapotát</li> </ol>

<b>Cím</b>	<b>Adjust Pump</b>
<b>Leírás</b>	A játékosok átállíthatják a pumpák állásait
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos átállíthatja, hogy melyik csőből melyik csőbe pumpáljon vizet a pumpa
<b>Alternatív forgatókönyv</b>	Minden pumpánál külön-külön állítható, hogy melyik csőből melyik másikba szállítson, pumpáljon vizet
<b>Alternatív forgatókönyv</b>	Minden pumpánál több csövet össze lehet kötni egymással
<b>Alternatív forgatókönyv</b>	A szerelők feladata, hogy a cisternákba kerüljön a megfelelő mennyiségű víz
<b>Alternatív forgatókönyv</b>	A szabotőrök dolga, hogy a pumpák állításával minél jobban hátráltassák a szerelők munkáját
<b>Alternatív forgatókönyv</b>	A többi rákötött cső eközben zárva van

<b>Cím</b>	<b>Attach Pipe</b>
<b>Leírás</b>	A játékos egy meglévő csőhöz (amelyen éppen tartózkodik) hozzácsatlakoztat egy másik csövet
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos hozzá csatlakoztat egy csövet azon csőhöz, amelyen éppen áll
<b>Alternatív forgatókönyv</b>	A csőhöz nem lehet további csöveget csatlakoztatni (mivel már van hozzá csatlakoztatva egy másik cső). Ekkor nem történik meg a cső csatlakoztatása
<b>Alternatív forgatókönyv</b>	Amennyiben azon csőben víz folyt, amelyhez egy újabb csövet csatlakoztattunk, az új csatlakoztatott csőben is elkezd folyni a víz

<b>Cím</b>	<b>Detach Pipe</b>
<b>Leírás</b>	A játékos egy szabad végű csövet lecsatlakoztat azon csőről, amelyen éppen áll
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos sikeresen lecsatlakoztatja a szabadvégű csövet
<b>Alternatív forgatókönyv</b>	A lecsatlakoztatott csőben víz folyt, így az új szabadvégű csőből elkezd folyni a víz
<b>Alternatív forgatókönyv</b>	A cső lecsatlakoztatása sikertelen volt mivel ahhoz a csőhöz, amelyen a játékos áll nem csatlakozik szabadvégű cső

<b>Cím</b>	<b>Repair Pump</b>
<b>Leírás</b>	A szerelő játékosok javítják meg az elromlott pumpákat
<b>Aktorok</b>	Mechanic
<b>Forgatókönyv</b>	A szerelő megjavítja az elromlott pumpát

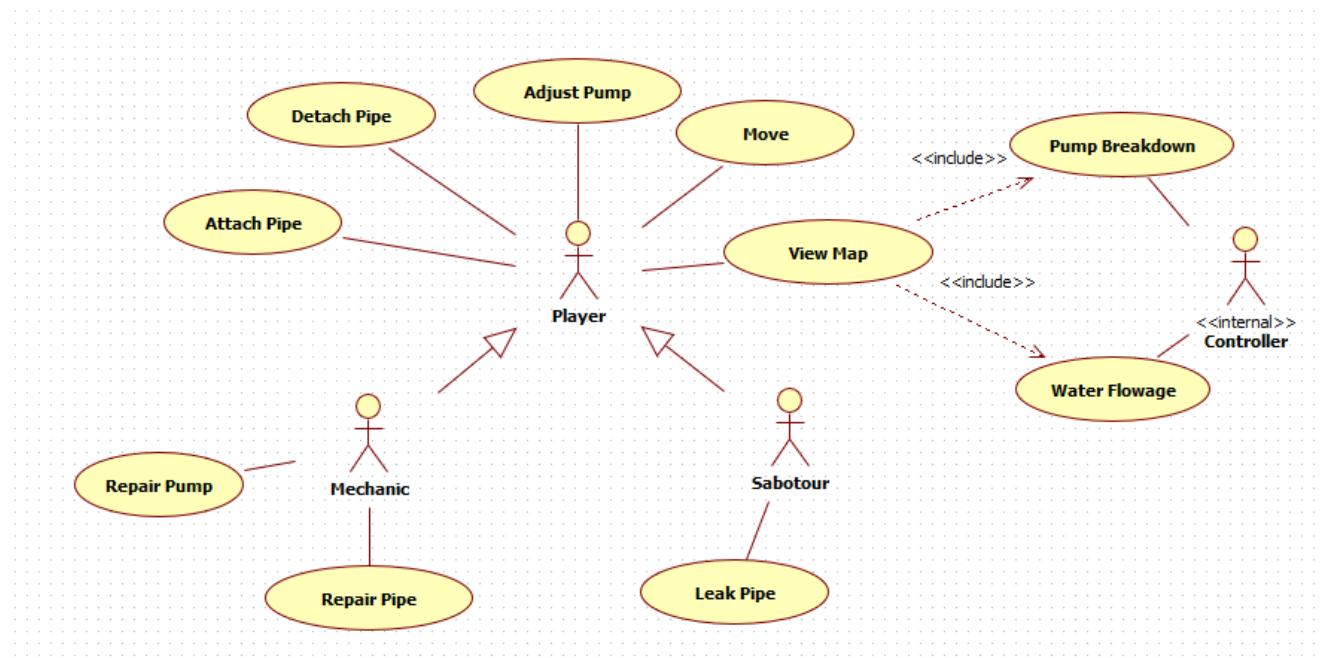
<b>Cím</b>	<b>Repair Pipe</b>
<b>Leírás</b>	A szerelő játékosok javítják meg a kilyukasztott csöveget
<b>Aktorok</b>	Mechanic
<b>Forgatókönyv</b>	A szerelő megjavítja a kilyukasztott csövet

<b>Cím</b>	<b>Leak Pipe</b>
<b>Leírás</b>	A szabotőr játékosok kilyukasztják a csöveget
<b>Aktorok</b>	Saboteur
<b>Forgatókönyv</b>	A szabotőr kilyukasztja a csövet, ezzel közelebb jutva a győzelemhez

<b>Cím</b>	<b>Pump Breakdown</b>
<b>Leírás</b>	A pumpák véletlen időközönként el tudnak romlani
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	Pumpa elromlik
<b>Alternatív Forgatókönyv</b>	A pumpán folyt át víz, ekkor amíg nincsen megjavítva a pumpa addig nem megy át az egyik csőből a másikba

<b>Cím</b>	<b>Water Flowage</b>
<b>Leírás</b>	A víz folyását jelzi, ezt a gép irányítja
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A sivatagon keresztül csöveken keresztül szállítódik a víz
<b>Alternatív forgatókönyv</b>	A víz a hegyi forrásból származik, ebben véges mennyiségű víz van
<b>Alternatív forgatókönyv</b>	A szabotörök feladata, hogy a minél több víz kerüljön ki a csövekből
<b>Alternatív forgatókönyv</b>	A szerelők feladata, hogy minél több víz kerüljön a ciszternába
<b>Alternatív forgatókönyv</b>	Ha egy csövön folyik át víz és a csövet kilyukasztják a csőből a víz szivárogni kezd
<b>Alternatív forgatókönyv</b>	Ha egy pumpa elromlik akkor a kimeneti csőben nem lesz víz
<b>Alternatív forgatókönyv</b>	Ha a hegyi forrásból kifogy a víz akkor a játéknak vége
<b>Alternatív forgatókönyv</b>	Az a játékos nyer amely több vizet szerzett a játék végére

## 2.4.2 Use-case diagram



## 2.5 Szótár

Adjust pump	Pumpa átállítása
Auditálás	Annak az ellenőrzése, hogy a program megfelel-e a követelményeknek.
Controller	Gép által irányított folyamatok
DBMS	Adatbázis-kezelő rendszer
Detach Pipe	Cső egyik végének leválasztása
Interfész	Felhasználói felület
Leak pipe	Cső kilyukasztása
Move	Mozgás a pályán

Op. rendszer	Operációs rendszer
OS	Operációs rendszer
Player	Játékos
Pump breakdown	Pumpa elrontása
RAM	A számítógép memóriája
Repair pipe	Cső megjavítása
Repair pump	Pumpa megjavítása
Szoftver	Az elkészítendő program
Use-case	Használati eset
View map	Pálya megtekintése
Water flowage	Víz folyása

## 2.6 Projekt terv

A projekt elkészítése egy 13 héten át tartó folyamat, amelynek heti rendszerességgel készítjük el az egyes lépéseit. Az ütemezés a következő:

- Követelmény, projekt, funkcionalitás márc. 13. 14:00
- Analízis modell (I. változat) március. 20. 14:15
- Analízis modell (II. változat) márc. 27. 14:15
- Szkeleton tervezése ápr. 3. 14:15
- Szkeleton elkészítése(2 hét) ápr. 17. 14:15
- Prototípus koncepciója ápr. 24. 14:15
- Részletes tervek máj. 3. labor
- Prototípus elkészítése máj. 8. 14:15
- Grafikus változat tervei máj. 15. 14:15
- Grafikus változat elkészítése(2 hét) máj 31. labor
- Egyesített dokumentáció jún. 2. 14.00

**Munkamegosztás, felelősségek:**

Az egyes feladatokért mindenki egyenlő mértékben felel, minden lépést együtt teljesítünk, ez persze nem zárja ki azt, hogy nem bontjuk részfeladatokra az egyes feladatokat. Ha ez így történik, akkor is a részfeladatok elkeszülte után összerakjuk a munkánkat és ellenőrizzük, hogy helyes-e a végeredmény. Ha helyes, akkor a csapatvezető leadja a munkánkat a Hercules feladatbeadó rendszeren keresztül. A papír alapú beadásról, nyomtatásról mindig az adott héten egyeztetünk, hogy ki nyomtatja ki és adjá le.

**Használt eszközök, platformok, erőforrások:**

Kommunikációra a Discord szolgál. Itt osztjuk meg a közösen szerkesztendő szöveges(!) dokumentumokat (ezek jellemzően Word illetve pdf dokumentumok lesznek). Magát az elkészítendő dokumentumokat pedig egy közösen szerkeszthető (online) dokumentumban készítjük el: Google Dokumentumok. A kódokat Git verziókezelő platformon osztjuk meg egymással. Fejlesztőkörnyezetnek az Eclipse-et használjuk (Eclipse IDE for Java Developers).

## 2.7 Napló

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résztvevők</b>	<b>Leírás</b>
2023.03.11. 12:00	2,5 óra	Réti Ádám Tisza Miklós Czifra Barnabás Kopach Artem Molnár Márton	Értekezlet: Feladatak kiosztása.
2023.03.11. 12:30	2,5 óra	Molnár Márton	Tevékenység: 2.1 Bevezetés leírása 2.2.2 Funkciók leírása 2.5 Szótár
2023.03.11. 12:30	2.5 óra	Réti Ádám	Tevékenység: 2.1.3 Definíciók, rövidítések 2.2.3 Felhasználók 2.2.4 Korlátozások 2.2.5 Feltételezések, kapcsolatok
2023.03.11. 15:00	1.5 óra	Réti Ádám Kopach Artem	Értekezlet: 2.3.2 Erőforrásokkal kapcsolatos követelmények 2.3.3 Átadással kapcsolatos követelmények 2.3.4 Egyéb nem funkcionális követelmények
2023.03.11.12:00	1.5 óra	Tisza Miklós Czifra Barnabás	Értekezlet: 2.3 Követelmények funkciionalitásának meghatározása
2023.03.11. 13:00	1 óra	Czifra Barnabás	Tevékenység: 2.6 Projekt terv
2023.03.11 12:30	2 óra	Kopach Artem	Értekezlet: 0. Címoldal sablont megcsináltam 2.2.1 Általános áttekintés (képpel)
2023.03.11 13:30	1.5 óra	Tisza Miklós	Tevékenység: 2.4 Use-Case leírások
2023.03.12 2:00	1 óra	Réti Ádám	Tevékenység: Dokumentum formáltságának megszerkesztése (Beszúrások, sortörések stb.)

# **Analízis modell I.**

**77 – gods\_of\_jar**

Konzulens:  
**Vörös András**

## **Csapattagok**

**Réti Ádám -  
csapatvezető**

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)      ket.vill.adam@gmail.com  
(E1LX0J)      tisza.miklos.16@gmail.com  
(NX9GA4)      barna.czifra@gmail.com  
(KLM600)      molnar.marton.hun@edu.bme.hu  
(JQBOLI)      kopach.artem@edu.bme.hu

2023.03.18

### **3. Analízis modell kidolgozása**

#### **3.1 Objektum katalógus**

##### **3.1.1 Objektum 1 - Pipe**

Cső - a cső a csőrendszerünk egyszerű elemei. A cső lehet "transfer" a vizet A pontból B pontba. Feladatai közé tartozik a víz átadása és a víz áramlásának útvonalának biztosítása. Olyan tulajdonságokkal rendelkezik, mint a másik csővel való összeköttetés (ami a rendszer kiterjesztéséhez és megváltoztatásához vezet), a szivattyú elérhetősége, a rugalmas.

##### **3.1.2 Objektum 2 - Container**

Az container - ez a csőrendszernek egy funkcionális elemei (pl., egy forrás(spring), ciszterna (cisterna), napelemes vízszivattyú(solar water pump) és stb.) Feladatai, hogy forrást vagy célt biztosítson a vízáramlás számára. Olyan attribútumokkal rendelkezik, mint a helye (hovatartozás, csőcsatlakozás n) és a kapacitása.

##### **3.1.3 Objektum 3 - Pumps**

Először is, úgy döntünk, hogy a "Pump" külön objektumként írjuk, mert ez a Pump egy fizikai elemei, amely aktívan részt vesz a vízátadási folyamatban. Feladatai, mint például: a víz áramlásának szabályozása a csöveken keresztül, a víz tárolása a víztartályában ("Each pump contains a water tank"), mint ideiglenes víztározó, a víz átadása a bejövő csőből a kimenő csőbe, a kimenő cső kapacitásának ellenőrzése a víz átadása előtt.

##### **3.1.4 Objektum 4 - Plumbers**

Plumbers - a plumbers felelősek a csőrendszer karbantartásáért és javításáért. Feladataik közé tartozik az elromlott szivattyúk javítása, a szivattyúk irányának beállítása, a szívárgó csövek javítása és a ciszternáknál gyártott csövekkel bővíthetik a rendszert. Olyan tulajdonságokkal rendelkeznek, mint a helyük és a csapathovatartozásuk. A vízvezeték-szerelők "main" feladata - cél, hogy minél több vizet juttassanak el a forrásokból a ciszternákba.

### 3.1.5 Objektum 5 - Saboteurs

Saboteurs - a saboteurs felelősek a csőrendszer szabotálásáért. Feladataik közé tartozik a szivattyúk irányának megváltoztatása és a csövek kilyukasztása. Olyan tulajdonságokkal rendelkeznek, mint a tartózkodási helyük, a csapathoz való tartozásuk. A szabotórcsapat feladata, hogy minél több vizet szívárogasson ki a csőrendszerből.

### 3.1.6 Objektum 6 - Game

Játék - a játékot két csapat játssza: a plumbers és a saboteurs. Feladatai, hogy nyomon kövesse az egyes csapatok által összegyűjtött víz mennyiséget, és az összegyűjtött víz mennyisége alapján meghatározza a győztest.

### 3.1.7 Objektum 7 - Cistern

Először is úgy döntöttünk, hogy a Cistern is felvesszük Objektum katalógusba, mert az ugyanolyan fontos szerepet játszik ebben a feladatban, mint a Pump. A Cistern tehát a csőrendszer aktív elemei, amely víztároló tartályként működik. Felelős a csövek készítéséért, a víz tárolásáért és a ciszternában lévő vízszint növeléséért.

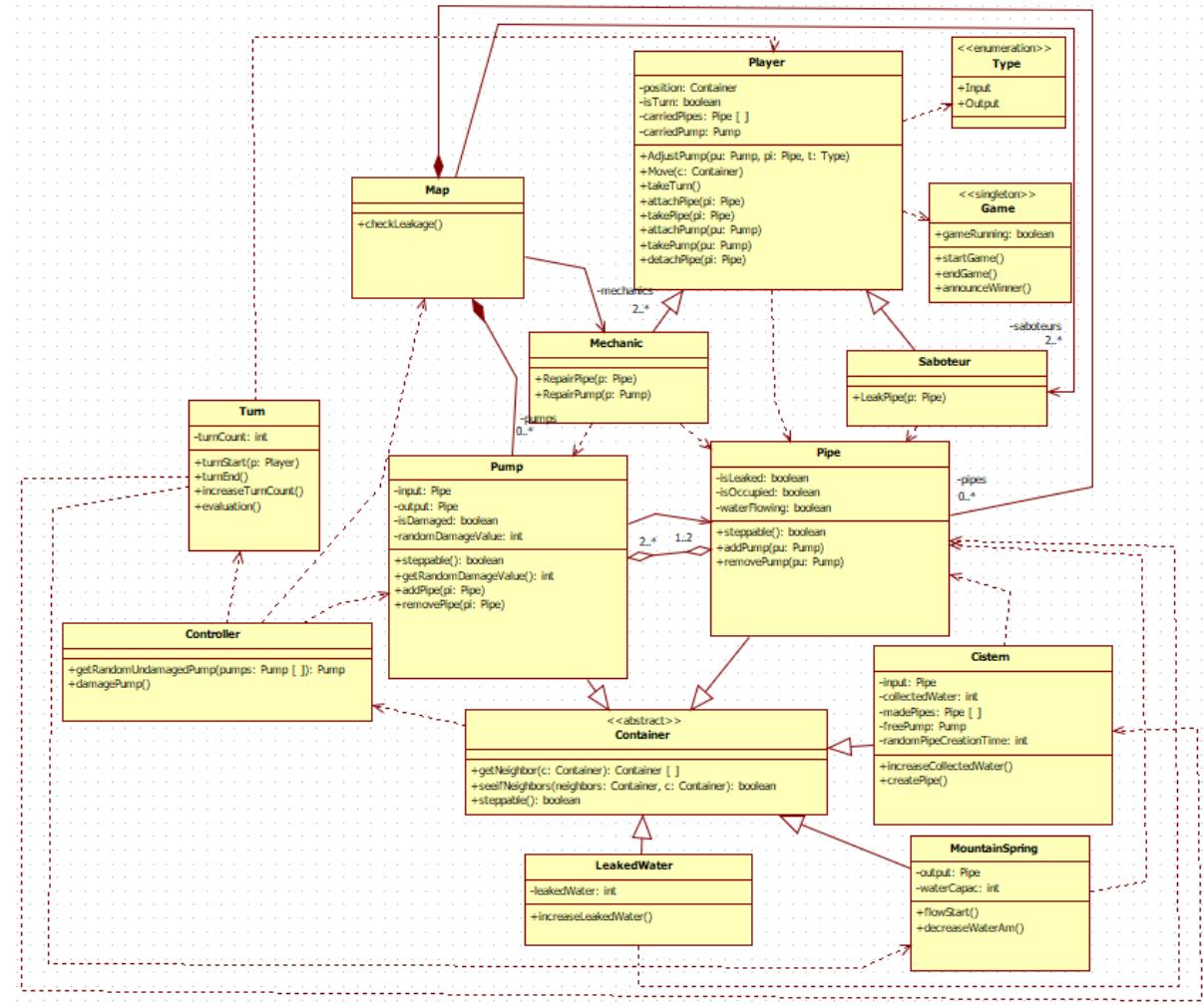
**Komment:** “A class is a template for objects. A class defines object properties including a valid range of values, and a default value. A class also describes object behavior. An object is a member or an "instance" of a class.” és “Analogy: an object is something made from a drawing, a class is a drawing.”

Source: [https://en.wikipedia.org/wiki/Object\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Object_(computer_science)) ,  
[https://www.ncl.ucar.edu/Document/HLUs/User\\_Guide/classes/classoview.shtml#:~:text=Class%20versus%20object&text=A%20class%20is%20a%20template%20for%20objects.,%22instance%22%20of%20a%20class.](https://www.ncl.ucar.edu/Document/HLUs/User_Guide/classes/classoview.shtml#:~:text=Class%20versus%20object&text=A%20class%20is%20a%20template%20for%20objects.,%22instance%22%20of%20a%20class.) ).

Néhány pont arról, hogy melyik objektum melyik osztályhoz/osztályokhoz kapcsolódik:

Objektum	Class
Pipe	Pipe
Container	MountainSpring, Cistern, Pump, LeakedWater, Container
Pumps	Pump
Plumbers	Mechanic, Player
Saboteurs	Saboteur, Player
Game	Game, Turn, Controller, Container, Map

## 3.2 Statikus struktúra diagramok



### 3.3 Osztályok leírása

#### 3.3.1 Cistern

- **Felelősség**

Ez a ciszterna, a szerelőknek ide kell eljuttatni a forrásból érkező vizet

- **Ősosztályok**

Container → Cistern

- **Interfészek**

-

- **Attribútumok**

-input: Pipe	Ez a bemeneti cső, innen érkezik a víz a ciszternába, ha ezen a csövön keresztül megy víz
-collectedWater: double	Itt tároljuk a ciszternába került víz mennyiségét
-madePipes: Pipe [ ]	Itt tároljuk a ciszterna által készített csöveket.
-freePump: Pump	Itt tároljuk a mozgatható pumpát.
-randomPipeCreationTime: int	Ez felel a csövek véletlenszerű időközönkénti készítéséért/létrehozásáért.

- **Metódusok**

+increaseCollectedWater()	Ezzel a metódussal tudjuk növelni a ciszternában lévő víz mennyiségét
+createPipe()	Ezzel a metódussal adódnak csövek a ciszternába elvitelre

### 3.3.2 Container

- **Felelősség**

Egy általános tárolót valósít meg (Pipe, Pump, Leaked, Mountain Spring, Cistern)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+getNeighbor(c: Container): Container [ ]	Lekérdezi az adott konténer szomszédjait
+steppable(): boolean	Lekérdezi, hogy az adott Container léphető-e azaz a játékos rá tud-e lépni, ez a függvény csak Pipe és Pump esetében tér vissza true-val a többi Container esetén false
-seeifNeighbors(neighbor: Container, c: Container): boolean	Megnézi, hogy a megadott Container objektum közvetlen szomszédságában van-e a másik Container objektum

### 3.3.3 Controller

- **Felelősség**

Ez felelős a játékon belül történt eseményekért (véletlenszerű pumpa elrontása)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+damagePump(p: Pump)	Ez a metódus felelős azért, hogy véletlenszerű időközönként elrontson egy véletlenszerű pumpát
----------------------	--

### 3.3.4 Game

- **Felelősség**

A játék elindításáért és leállításáért felelős

- **Ősosztályok**

-

- **Interfész**

-

- **Attribútumok**

-gameRunning: boolean	Azt tárolja, hogy a játék még megy-e
-----------------------	--------------------------------------

## Metódusok

+startGame()	Ez a metódus felelős a játék elindításáért
+endGame()	Ez a metódus felelős a játék leállításáért
+announceWinner	Ez a metódus felelős a játék végén a nyertes csapat kihirdetéséért

## LeakedWater

- **Felelősség**

Ez felelős a csövekből kiszivárgott víz nyomon követésére

- **Ősosztályok**

Container -> LeakedWater

- **Interfészek**

-

- **Attribútumok**

-leakedWater: int	Ebben tároljuk a kiszivárgott víz mennyiségét (egységekben)
-------------------	---

- **Metódusok**

+increaseLeakedWater()	Ezzel tudjuk növelni a kiszivárgott víz mennyiségét
------------------------	---

### 3.3.5 Map

- **Felelősség**

Ennek a felelőssége a pályán elhelyezett pumpák és csövek tárolása

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-pipes: Pipe[0..* ]	Itt tároljuk a pályán lévő csöveket
-pumps: Pumps[ 0..*]	Itt tároljuk a pályán lévő pumpákat
-mechanics: Mechanic[2..*]	Itt tároljuk a pályán lévő szerelőket
-saboteurs: Saboteur[2..*]	Itt tároljuk a pályán lévő szabotőröket

- **Metódusok**

+checkLeakage()	Ennek a metódusnak a segítségével tudjuk végig vizsgálni, hogy a pályán a csöveknél van-e szivárgás
-----------------	---

### 3.3.6 Mechanic

- **Felelősség**

Ez a szerelő játékos, a szerelő javíthatja meg a csöveket, pumpákat és állíthatja is azokat

- **Ősosztályok**

Player → Mechanic

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+RepairPipe(p: Pipe)	Ezen metódussal javítja meg a szerelő az adott csövet
+RepairPump(p: Pump)	Ezen metódussal javítja meg a szerelő az adott csövet

### 3.3.7 MountainSpring

- **Felelősség**

Ez a hegyi forrás, ebből származik a víz, amelyért a játékosok versengenek

- **Ősosztályok**

Container → MountainSpring

- **Interfészek**

-

- **Attribútumok**

-waterCapac: int	A hegyi forrásvíz kapacitása, azaz mennyi vizet képes tárolni a hegyi forrás
-output: Pipe	Tárolja azt a csövet, amely kivezeti a vizet a forrásból

- **Metódusok**

+flowstart(): void	Ennek a metódusnak a segítségével kezd a forrásból kiszivárogni a víz
+decreaseWaterAm():	A hegyi forrásban lévő víz mennyiséget csökkenti

### 3.3.8 Pipe

- **Felelősség**

Ez a cső, ez felelős a víz szállításához, ennek segítségével szállítódhat a víz a ciszternába (amely a szerelők előnyét jelenti) vagy ezeknek kilyukasztásával kerülhetnek előnybe a szabotőrök.

- **Ósosztályok**

Container → Pipe

- **Interfészek**

-

- **Attribútumok**

-isLeaking: boolean	Ez az attribútum jelzi, hogy az adott cső ki van-e lyukasztva vagy sem
-isOccupied: boolean	Ez az attribútum jelzi, hogy valaki éppen az adott csövön közlekedik (tehát foglalt)
-waterFlowing: boolean	Ez az attribútum jelzi, hogy az adott csövön víz folyik át
-pumps: Pump[1..2]	A csőhöz tartozó pumpák legkevesebb 1 maximum 2 pumpa tartozhat 1 csőhöz

- **Metódusok**

+steppable(): boolean	Azt adja meg, hogy a játékos ráléphet-e ezen Container-re, ez igaz értéket ad
+getNeighbors(): Container [ ]	A cső szomszédjait adja vissza

### 3.3.9 Player

- **Felelősség**

Egy általános játékos valósít meg (Mechanic, Saboteur)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-position: Container	Ezzel tároljuk, hogy az adott játékos hol helyezkedik el (Itt megadnánk, hogy a Container az nem lehet Mountain Spring Cistern és LeakedWater típusú, mivel azon a játékos nem állhat)
-isTurn: boolean	Ez jelzi, hogy a játékos köre van
-carriedPipes: Pipe [ ]	Ez tartalmazza a játékosok által, hordozott csöveket
-carriedPump: Pump	Ez tartalmazza a játékosok által, hordozott pumpát.

- **Metódusok**

+AdjustPump(pu: Pump, pi: Pipe, t: Type)	Ennek a metódusnak a segítségével állíthatják a játékosok a csöveket
+Move(c: Container)	A játékos ezzel a metódussal végzi el a mozgást (hasonlóan a positionnél a Container itt is csak cső és pumpa lehet)
+takeTurn()	Ez a metódus felelős a játékos kör kezdéséért
+attachPipe(pi: Pipe)	Ezen metódussal képes a játékos egy szabadvégű csőhöz újabb, nála lévő csövet csatlakoztatni
+attachPump(pu: Pump)	Ezen metódussal képes a játékos egy ketté vágott cső két része közé egy újabb, nála lévő pumpát csatlakoztatni
+detachPipe(pi: Pipe)	Ezzel lehet egy szabadvégű csövet lecsatlakoztatni a csőrendszerről
+takePipe(pi: Pipe)	Ezzel képes felvenni a játékos a ciszternában elkészített csövet
+takePump(pu: Pump)	Ezzel képes felvenni a játékos a ciszternában lévő pumpát

### 3.3.10 Pump

- **Felelősség**

Ez a pumpa, ez teszi lehetővé a csövek közötti összeköttetést

- **Ósosztályok**

Container → Pump

- **Interfész**

-

- **Attribútumok**

-connectedPipes: Pipe[ ]	Itt tároljuk a pumpához csatlakozó összes csövet
--------------------------	--

-input: Pipe	Ebben tároljuk, hogy a pumpában melyik cső a bemeneti cső (azaz melyikból akarjuk átpumpálni a vizet)
-output: Pipe	Ebben tároljuk a kimeneti csövet (azaz, hogy melyik csőbe akarjuk pumpálni a vizet)
-isDamaged: boolean	Ez az attribútum tárolja, hogy az adott pumpa éppen sérült-e vagy sem
-randomDamageValue: int	Azt az értéket tartalmazza amely meghatározza, hogy az adott pumpa mely körben fog megsérülni

- **Metódusok**

+steppable(): boolean	Igaz értéket ad vissza
+getRandomDamageValue(): int	Lekérdezi a pumpához tartozó elromlás körének értékét

### 3.3.11 Saboteur

- **Felelősség**

Ez a szabotőr játékos, a szabotőr lyukaszthatja ki a csöveket illetve a szabotőr a szerelőhöz hasonlóan átállíthatja a pumpákat

- **Ősosztályok**

Player → Saboteur

- **Interfész**

-

- **Attribútumok**

-

- **Metódusok**

+LeakPipe(p: Pipe)	Ezen metódussal lyukasztja ki a szabotőr az adott csövet
--------------------	--

### 3.3.12 Turn

- **Felelősség**

A kör megvalósításáért felelős

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-turnCount: int	Játék kezdete óta eltelt körök száma
-----------------	--------------------------------------

- **Metódusok**

+turnStart(p: Player)	Adott játékos körét indítja el
+turnEnd()	Adott játékos körét befejezi
+increaseTurnCount()	Növeli a turnCount értékét a kör végén.
+evaluation()	Körönként frissíti a pályán lévő objektumokat.

### 3.3.13 Type

- **Felelősség**

A pumpa módosításának típusát határozza meg (azaz, hogy bemeneti csövet vagy kimeneti csövet akarunk változtatni)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

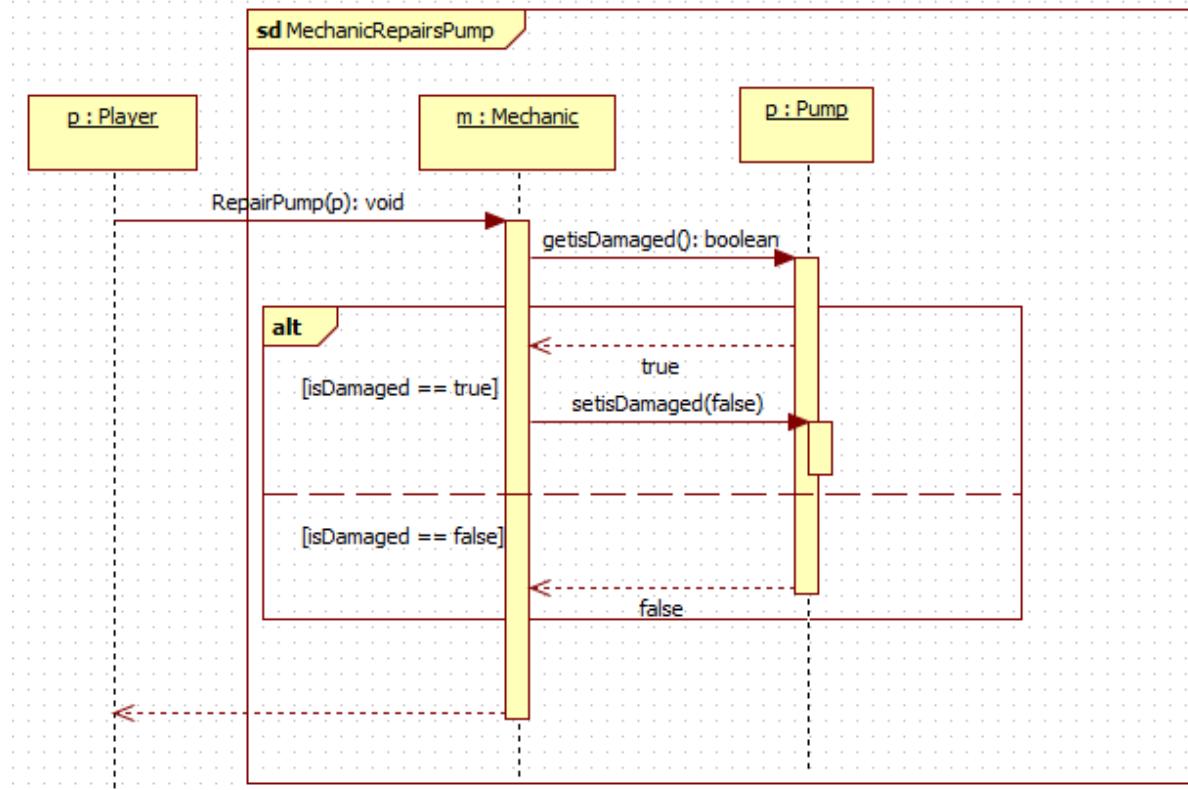
+Input	Ez a bemeneti csövet jelenti
+Output	Ez a kimeneti csövet jelenti

- **Metódusok**

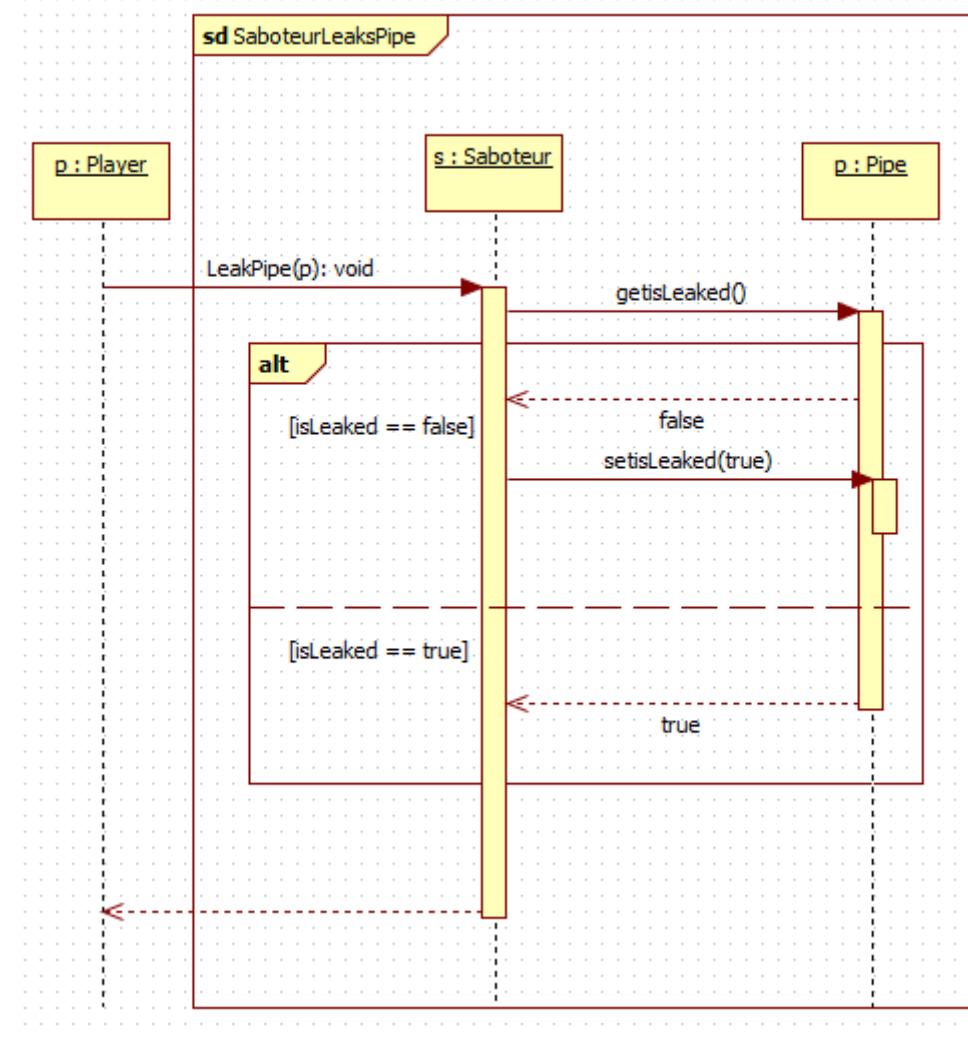
-

## 3.4 Szekvencia diagramok

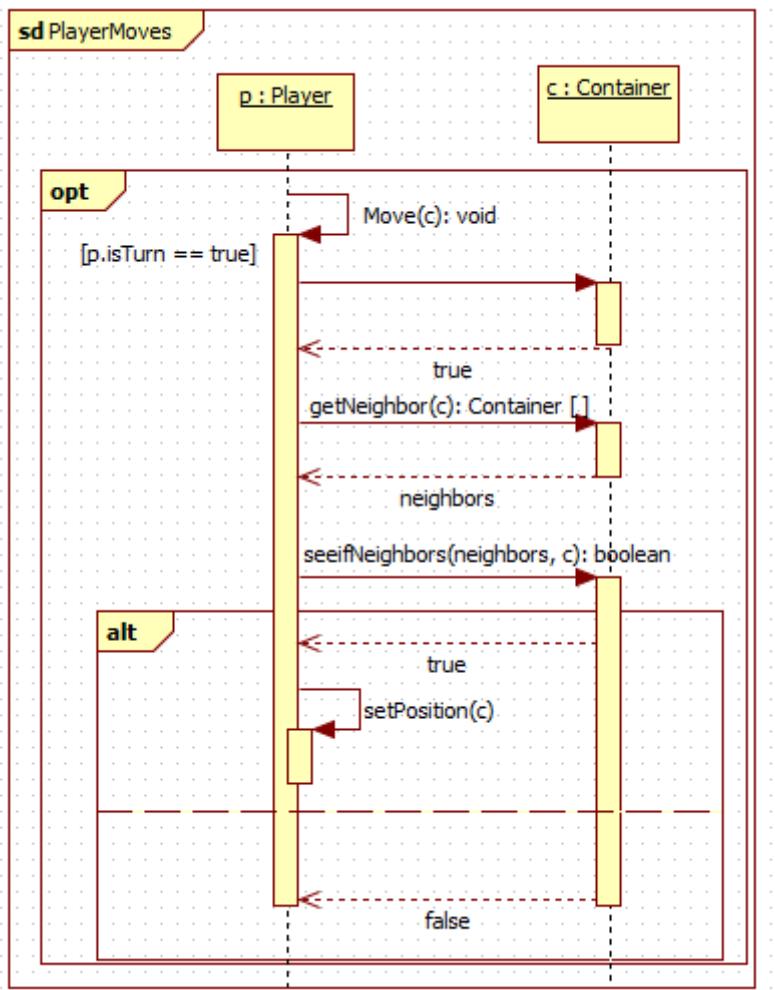
### 3.4.1 Mechanic repairs pump



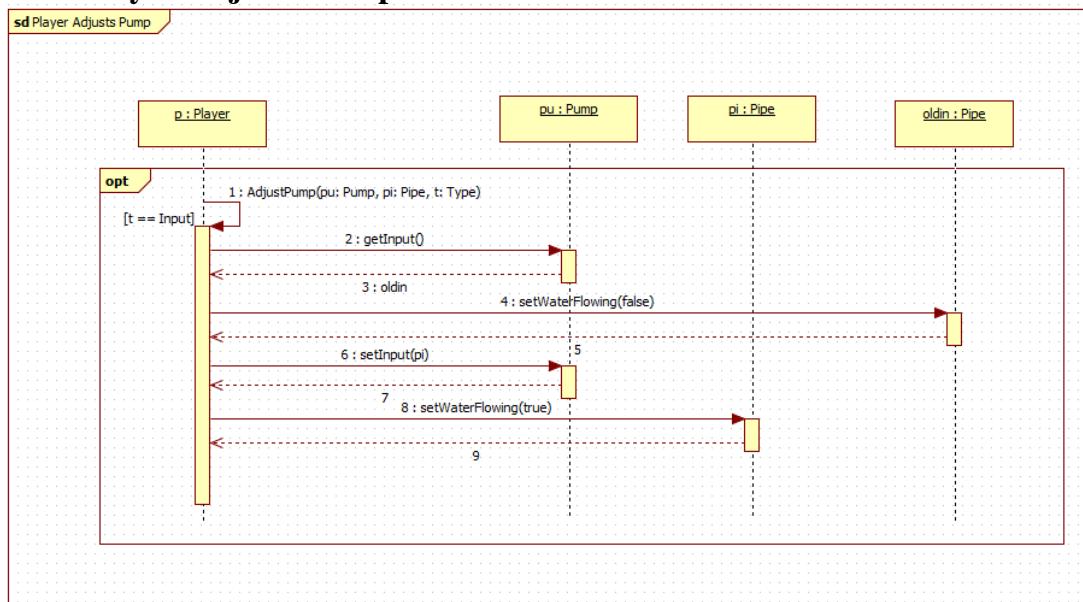
### 3.4.2 Saboteur leaks pipe



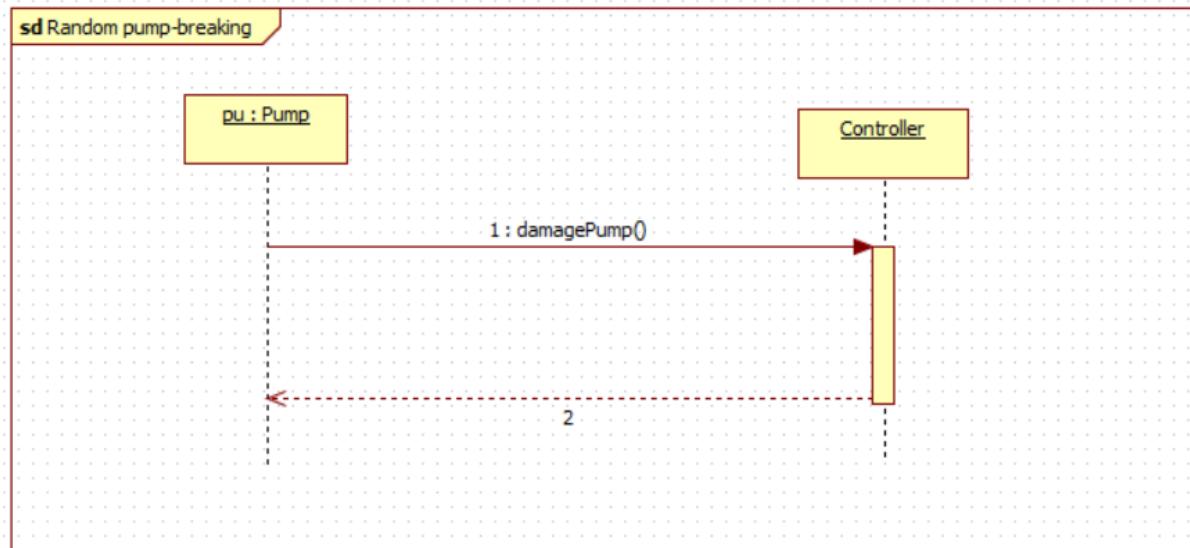
### 3.4.3 Player Moves



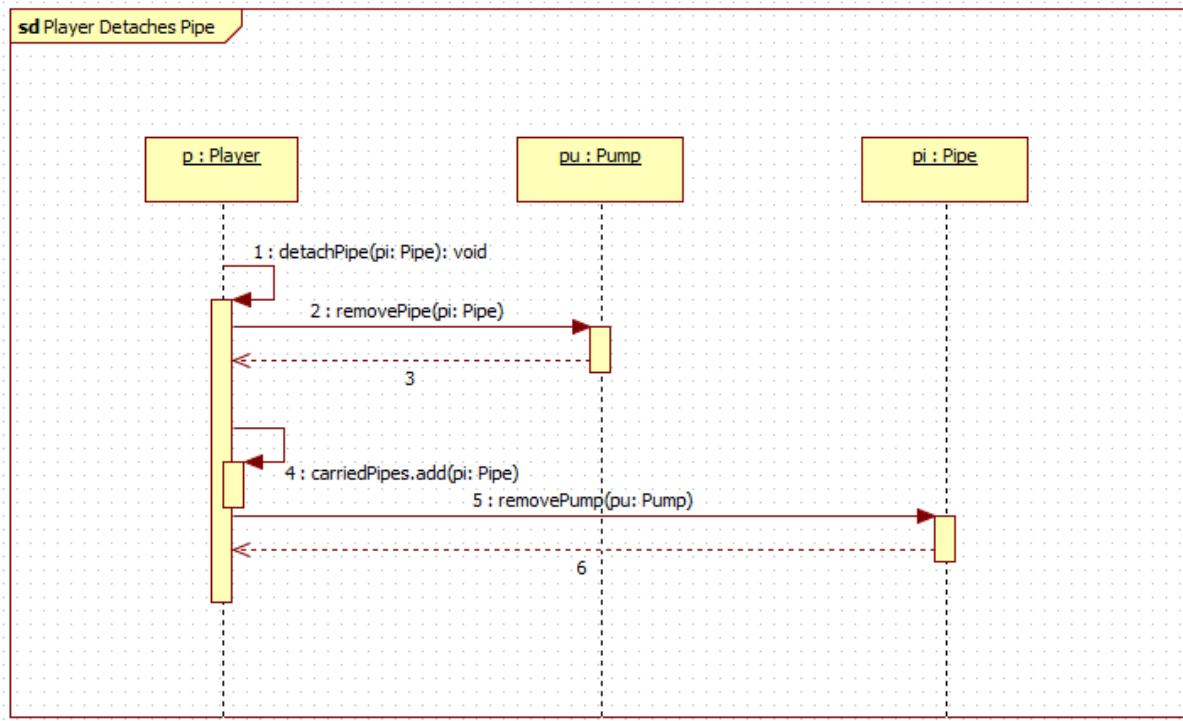
### 3.4.4 Player Adjusts Pump



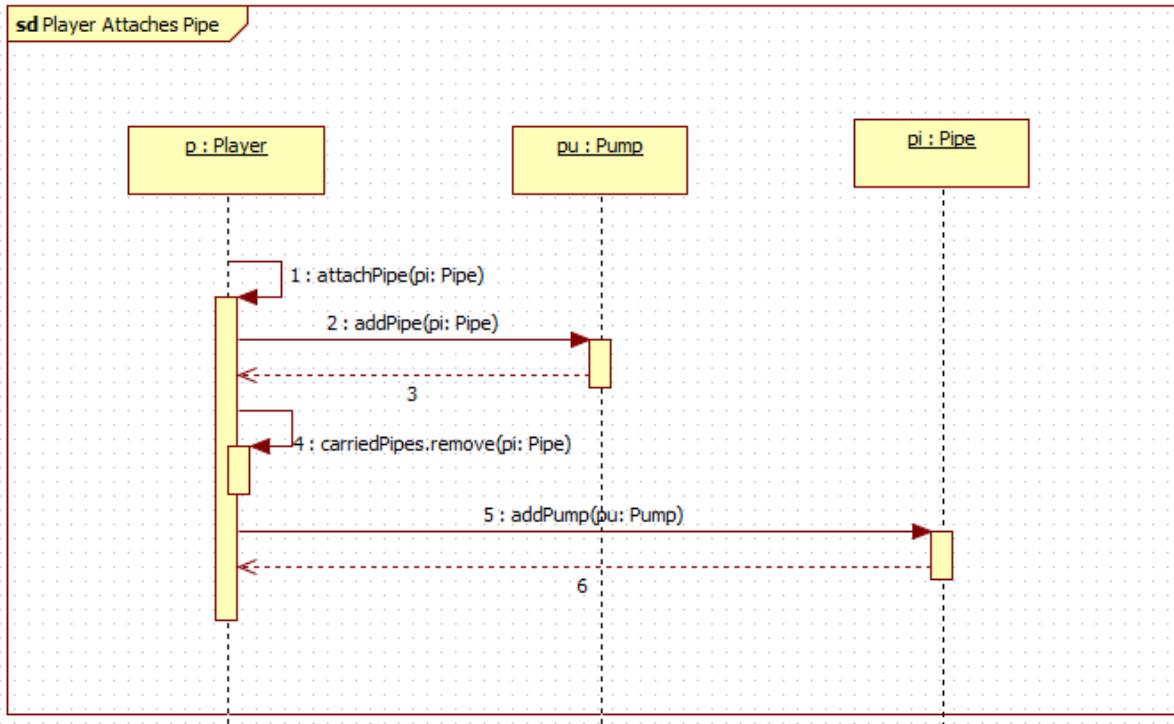
### 3.4.5 Random pump-breaking



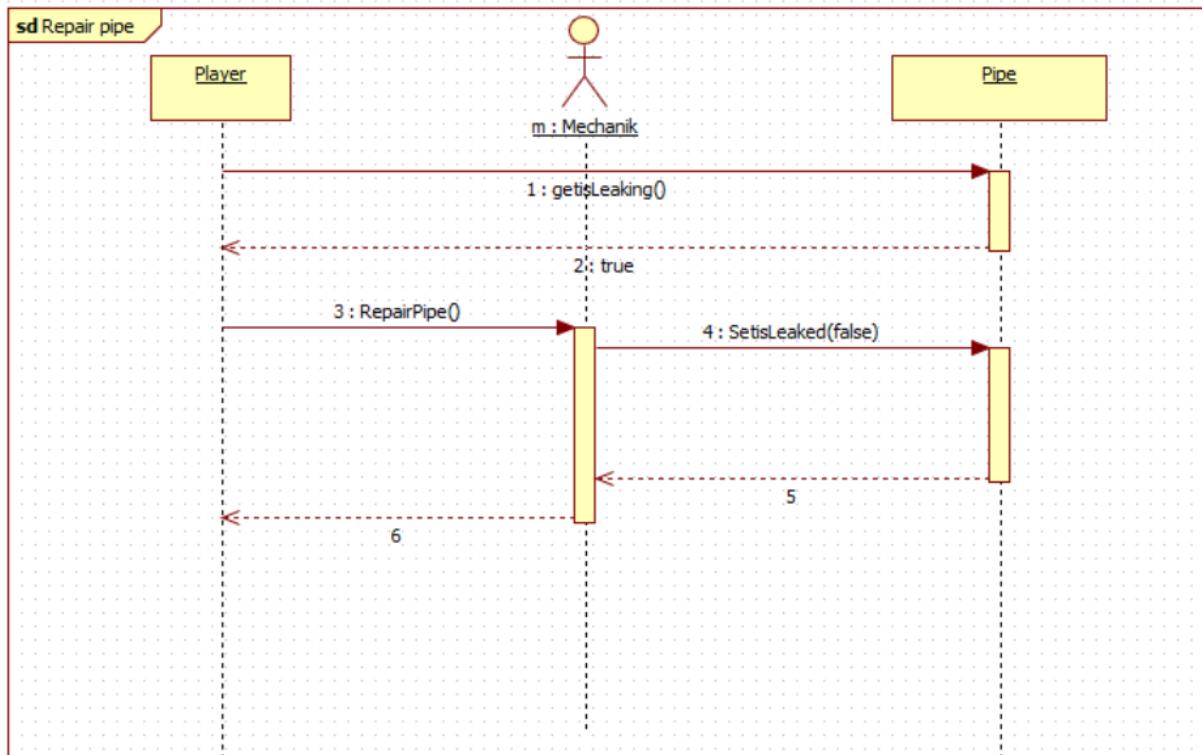
### 3.4.6 Player Detaches Pipe



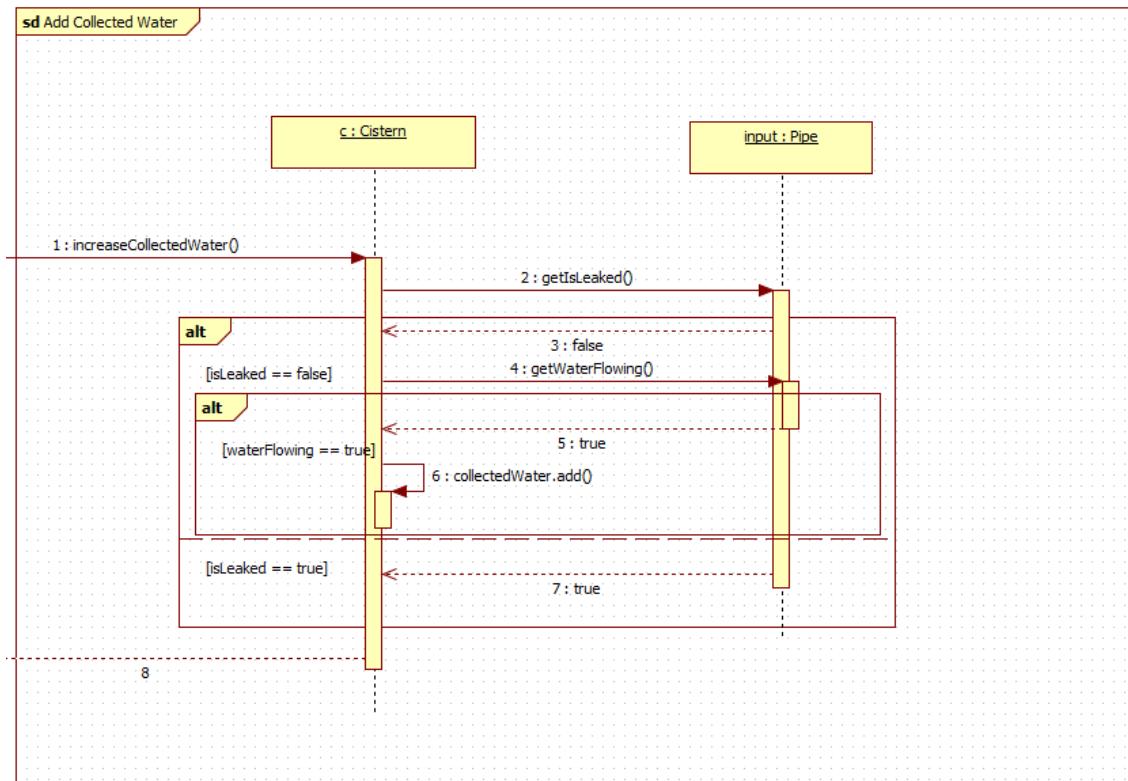
### 3.4.7 Player Attaches Pipe



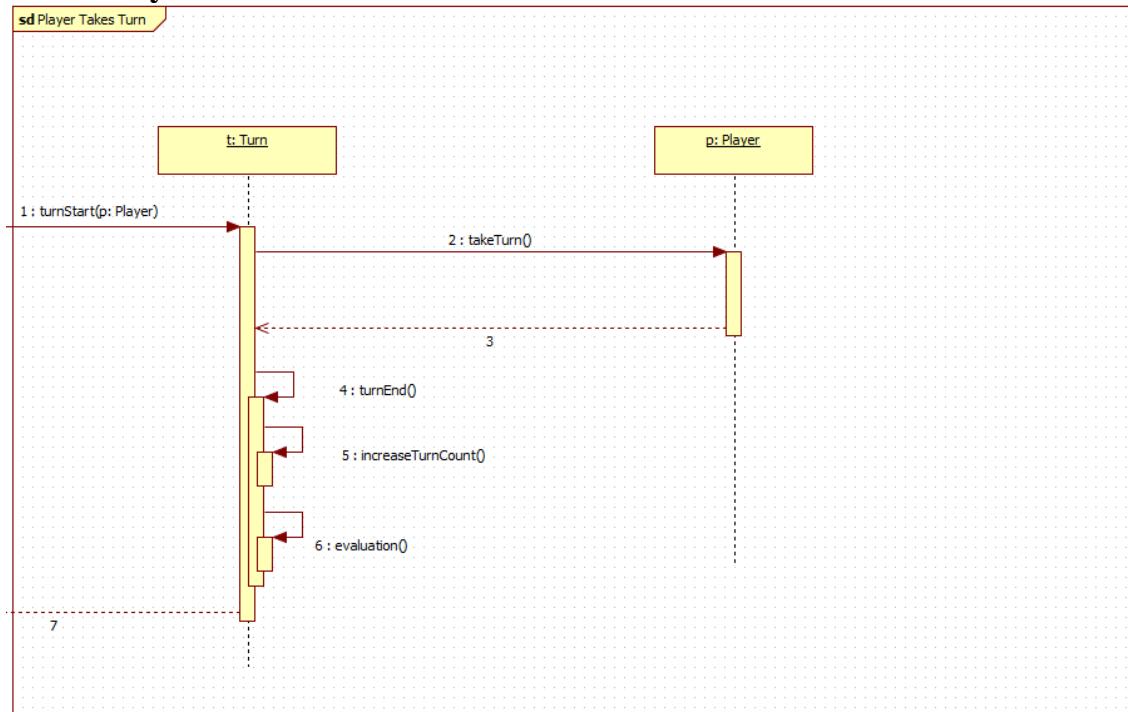
### 3.4.8 Repair pipe



### 3.4.9 Add Collected Water

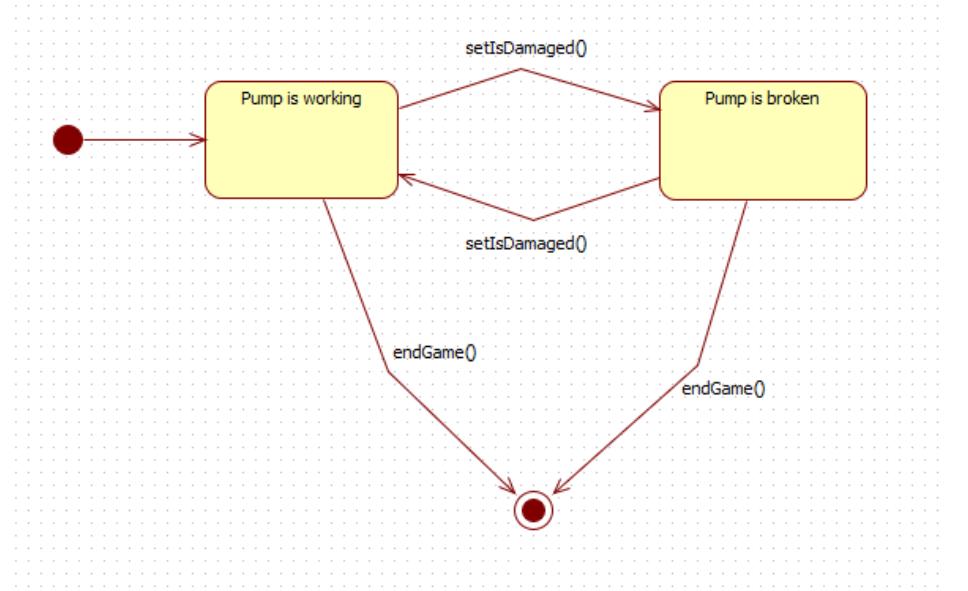


### 3.4.10 Player Takes Turn

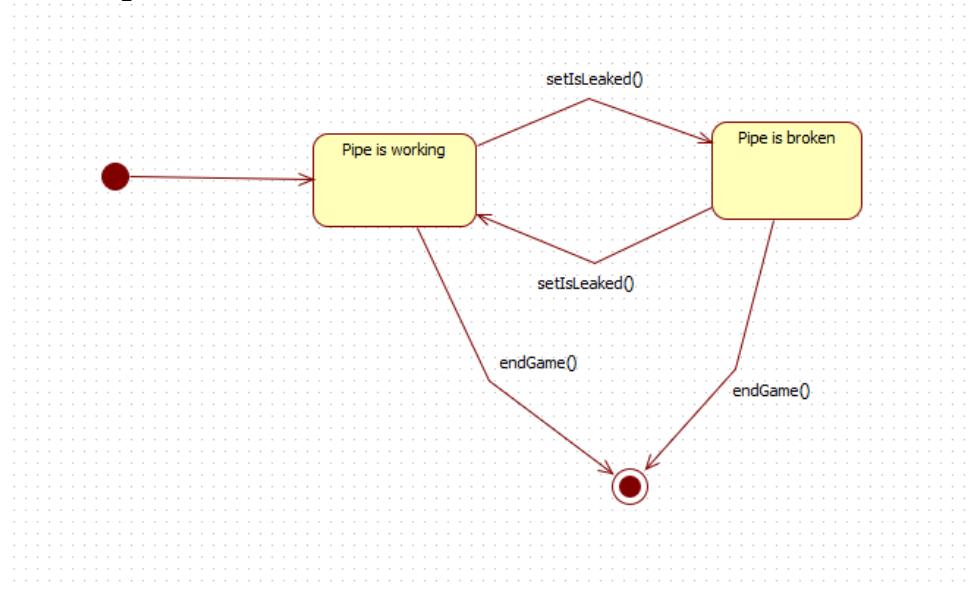


## 3.5 State-chartok

### 3.5.1 Pump State



### 3.5.2 Pipe State



### 3.6 Napló

Kezdet	Időtartam	Résztervezők	Leírás
2023.03.18. 12:30	0.5 óra	Réti Ádám Tisza Miklós Kopach Artem Czifra Barnabás Molnár Márton	Értekezlet. Feladatok kiosztása. Feladatok megtervezése, különböző elemek megvitátása
2023.03.18. 13:00	5 óra	Réti Ádám Tisza Miklós	Értekezlet: 3.2 Statikus struktúra diagramok 3.3 Osztályok leírása
2023.03.18. 19:00	1 óra	Czifra Barnabás Molnár Márton	Értekezlet: 3.4 Szekvencia diagramok
2023.03.18. 12:30	3 óra	Kopach Artem	Tevékenység: 0. Új cím sablont írtam 3.1 Objektum katalógus (commenttel, szerkesztésével és átdolgozásokkal, az osztálydiagramot tekintve)
2023.03.20	1 óra	Kopach Artem	Tevékenység: 0*. Nyomtatás és beadás
2023.03.18 20:00	1.5 óra	Molnár Márton Czifra Barnabás	3.4 Szekvencia diagramok elkészítése: 3.4.4, 3.4.6, 3.4.7, 3.7.8
2023.03.18 21:30	0.5 óra	Molnár Márton	3.5 Statechart elkészítése
2023.03.18 21:30	0.5 óra	Czifra Barnabás	Statechart-ban segítés
2023.03.18 22:00	0.5 óra	Tisza Miklós	Tevékenység: 3.3 Osztályok leírása
2023.03.19 5:00	0.5 óra	Réti Ádám	Dokumentum formáltságának megszerkesztése
2023.03.19 9:30	1 óra	Molnár Márton	3.4.9, 3.4.10 Szekvencia diagramok elkészítése

# **Analízis modell II.**

**77 – gods\_of\_jar**

Konzulens:  
**Vörös András**

## **Csapattagok**

**Réti Ádám -  
csapatvezető**

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)	ket.vill.adam@gmail.com
(E1LX0J)	tisza.miklos.16@gmail.com
(NX9GA4)	barna.czifra@gmail.com
(KLM60O)	molnar.marton.hun@edu.bme.hu
(JQBOLI)	kopach.artem@edu.bme.hu

2023.03.25

### 3. Analízis modell kidolgozása

#### 3.1 Objektum katalógus

##### 3.1.1 Objektum 1 - Pipe

Cső - a cső a csőrendszerünk egyszerű elemei. A cső lehet "transfer" a vizet A pontból B pontba. Feladatai közé tartozik a víz átadása és a víz áramlásának útvonalának biztosítása. Olyan tulajdonságokkal rendelkezik, mint a másik csővel való összeköttetés (ami a rendszer kiterjesztéséhez és megváltoztatásához vezet), a szivattyú elérhetősége, a rugalmas.

##### 3.1.2 Objektum 2 - Container

Az container - ez a csőrendszernek egy funkcionális elemei (pl., egy forrás(spring), ciszterna (cisterna), napelemes vízszivattyú(solar water pump) és stb.) Feladatai, hogy forrást vagy célt biztosítson a vízáramlás számára. Olyan attribútumokkal rendelkezik, mint a helye (hovatartozás, csőcsatlakozás n) és a kapacitása.

##### 3.1.3 Objektum 3 - Pumps

Először is, úgy döntünk, hogy a "Pump" külön objektumként írjuk, mert ez a Pump egy fizikai elemei, amely aktívan részt vesz a vízátadási folyamatban. Feladatai, mint például: a víz áramlásának szabályozása a csöveken keresztül, a víz tárolása a víztartályában ("Each pump contains a water tank"), mint ideiglenes víztározó, a víz átadása a bejövő csőből a kimenő csőbe, a kimenő cső kapacitásának ellenőrzése a víz átadása előtt.

##### 3.1.4 Objektum 4 - Plumbers

Plumbers - a plumbers felelősek a csőrendszer karbantartásáért és javításáért. Feladataik közé tartozik az elromlott szivattyúk javítása, a szivattyúk irányának beállítása, a szívárgó csövek javítása és a ciszternáknál gyártott csövekkel bővíthetik a rendszert. Olyan tulajdonságokkal rendelkeznek, mint a helyük és a csapathovatartozásuk. A vízvezeték-szerelők "main" feladata - cél, hogy minél több vizet juttassanak el a forrásokból a ciszternákba.

### 3.1.5 Objektum 5 - Saboteurs

Saboteurs - a saboteurs felelősek a csőrendszer szabotálásáért. Feladataik közé tartozik a szivattyúk irányának megváltoztatása és a csövek kilyukasztása. Olyan tulajdonságokkal rendelkeznek, mint a tartózkodási helyük, a csapathoz való tartozásuk. A szabotőrcsapat feladata, hogy minél több vizet szívárogasson ki a csőrendszerből.

### 3.1.6 Objektum 6 - Game

Játék - a játékot két csapat játssza: a plumbers és a saboteurs. Feladatai, hogy nyomon kövesse az egyes csapatok által összegyűjtött víz mennyiségét, és az összegyűjtött víz mennyisége alapján meghatározza a győztest.

### 3.1.7 Objektum 7 - Cistern

Először is úgy döntöttünk, hogy a Cistern is felvesszük Objektum katalógusba, mert az ugyanolyan fontos szerepet játszik ebben a feladatban, mint a Pump. A Cistern tehát a csőrendszer aktív elemei, amely víztároló tartályként működik. Felelős a csövek készítéséért, a víz tárolásáért és a ciszternában lévő vízszint növeléséért.

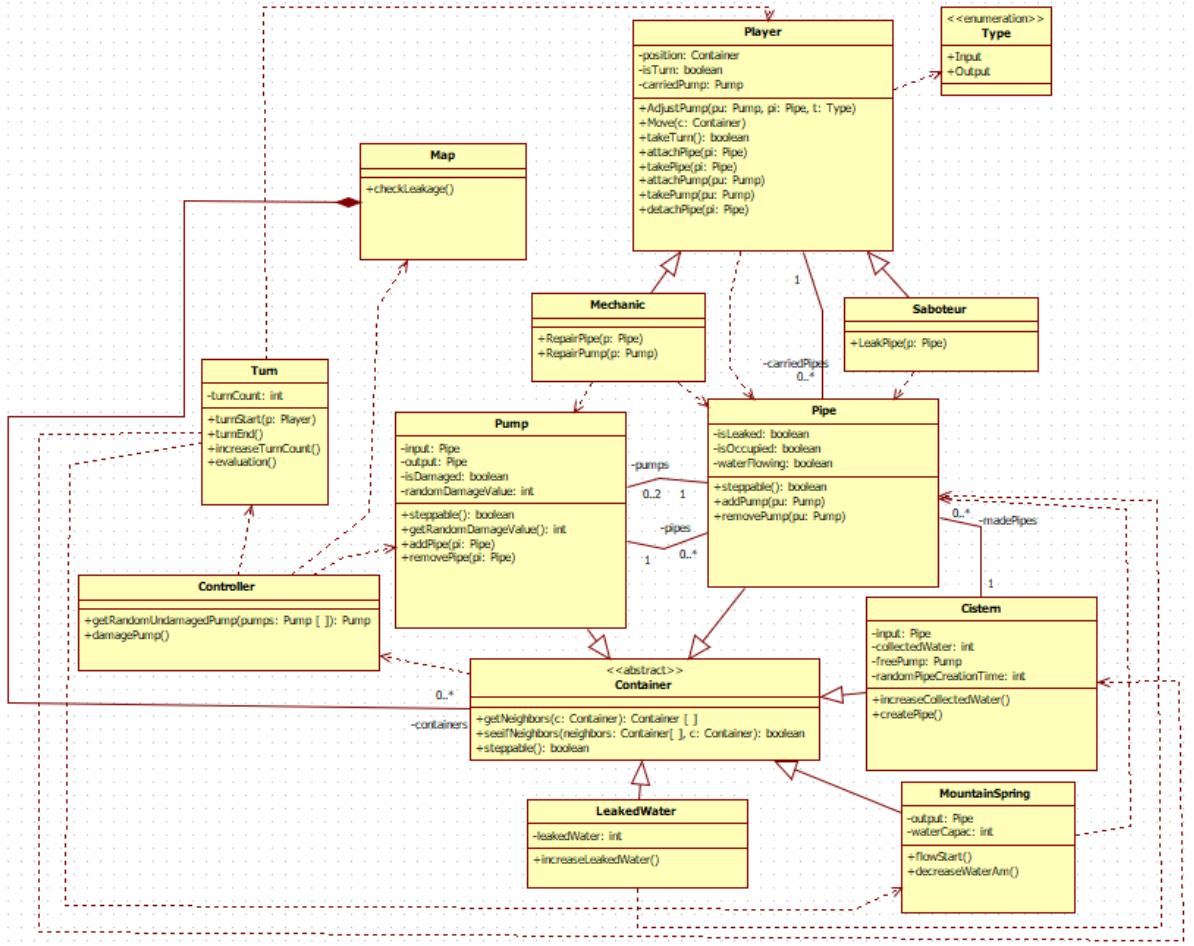
**Komment:** “A class is a template for objects. A class defines object properties including a valid range of values, and a default value. A class also describes object behavior. An object is a member or an "instance" of a class.” és “Analogy: an object is something made from a drawing, a class is a drawing.”

Source: [https://en.wikipedia.org/wiki/Object\\_\(computer\\_science\)](https://en.wikipedia.org/wiki/Object_(computer_science)) ,  
[https://www.ncl.ucar.edu/Document/HLUs/User\\_Guide/classes/classoview.shtml#:~:text=Class%20versus%20object&text=A%20class%20is%20a%20template%20for%20objects.,%22instance%22%20of%20a%20class.](https://www.ncl.ucar.edu/Document/HLUs/User_Guide/classes/classoview.shtml#:~:text=Class%20versus%20object&text=A%20class%20is%20a%20template%20for%20objects.,%22instance%22%20of%20a%20class.) ).

Néhány pont arról, hogy melyik objektum melyik osztályhoz/osztályokhoz kapcsolódik:

Objektum	Class
Pipe	Pipe
Container	MountainSpring, Cistern, Pump, LeakedWater, Container
Pumps	Pump
Plumbers	Mechanic, Player
Saboteurs	Saboteur, Player
Game	Game, Turn, Controller, Container, Map

## 3.2 Statikus struktúra diagramok



### 3.3 Osztályok leírása

#### 3.3.1 Cistern

- **Felelősség**

Ez a ciszterna, a szerelőknek ide kell eljuttatni a forrásból érkező vizet

- **Ősosztályok**

Container → Cistern

- **Interfészek**

-

- **Attribútumok**

-input: Pipe	Ez a bemeneti cső, innen érkezik a víz a ciszternába, ha ezen a csövön keresztül megy víz
-collectedWater: double	Itt tároljuk a ciszternába került víz mennyiségét
-madePipes: Pipe [ ]	Itt tároljuk a ciszterna által készített csöveket.
-freePump: Pump	Itt tároljuk a mozgatható pumpát.
-randomPipeCreationTime: int	Ez felel a csövek véletlenszerű időközönkénti készítéséért/létrehozásáért.

- **Metódusok**

+increaseCollectedWater()	Ezzel a metódussal tudjuk növelni a ciszternában lévő víz mennyiségét
+createPipe()	Ezzel a metódussal adódnak csövek a ciszternába elvitelre

### 3.3.2 Container

- **Felelősség**

Egy általános tárolót valósít meg (Pipe, Pump, Leaked, Mountain Spring, Cistern)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+getNeighbor(c: Container): Container [ ]	Lekérdezi az adott konténer szomszédjait
+steppable(): boolean	Lekérdezi, hogy az adott Container léphető-e azaz a játékos rá tud-e lépni, ez a függvény csak Pipe és Pump esetében tér vissza true-val a többi Container esetén false
-seeifNeighbors(neighbor: Container, c: Container): boolean	Megnézi, hogy a megadott Container objektum közvetlen szomszédságában van-e a másik Container objektum

### 3.3.3 Controller

- **Felelősség**

Ez felelős a játékon belül történt eseményekért (véletlenszerű pumpa elrontása)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+damagePump()	Ez a metódus felelős azért, hogy véletlenszerű időközönként elrontson egy véletlenszerű pumpát
getRandomUndamagedPump(pumps: Pump []): Pump	Visszaad egy véletlenszerű nem sérült pumpát

### LeakedWater

- **Felelősség**

Ez felelős a csövekből kiszivárgott víz nyomon követésére

- **Ősosztályok**

Container -> LeakedWater

- **Interfészek**

-

- **Attribútumok**

-leakedWater: int	Ebben tároljuk a kiszivárgott víz mennyiségét (egységekben)
-------------------	---

- **Metódusok**

+increaseLeakedWater()	Ezzel tudjuk növelni a kiszivárgott víz mennyiségét
------------------------	---

### 3.3.4 Map

- **Felelősség**

Ennek a felelőssége a pályán elhelyezett pumpák és csövek tárolása

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-containers: Container[0..*]	Itt tároljuk listában a pályán lévő konténereket (pl.: Mountain Spring, Cistern, Pipe stb.)
------------------------------	---

- **Metódusok**

+checkLeakage()	Ennek a metódusnak a segítségével tudjuk végig vizsgálni, hogy a pályán a csöveknél van-e szivárgás
-----------------	---

### 3.3.5 Mechanic

- **Felelősség**

Ez a szerelő játékos, a szerelő javíthatja meg a csöveket, pumpákat és állíthatja is azokat

- **Ősosztályok**

Player → Mechanic

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+RepairPipe(p: Pipe)	Ezen metódussal javítja meg a szerelő az adott csövet
+RepairPump(p: Pump)	Ezen metódussal javítja meg a szerelő az adott csövet

### 3.3.6 MountainSpring

- **Felelősség**

Ez a hegyi forrás, ebből származik a víz, amelyért a játékosok versengenek

- **Ősosztályok**

Container → MountainSpring

- **Interfészek**

-

- **Attribútumok**

-waterCapac: int	A hegyi forrásvíz kapacitása, azaz mennyi vizet képes tárolni a hegyi forrás
-output: Pipe	Tárolja azt a csövet, amely kivezeti a vizet a forrásból

- **Metódusok**

+flowstart(): void	Ennek a metódusnak a segítségével kezd a forrásból kiszivárogni a víz
+decreaseWaterAm():	A hegyi forrásban lévő víz mennyiségett csökkenti

### 3.3.7 Pipe

- **Felelősség**

Ez a cső, ez felelős a víz szállításához, ennek segítségével szállítódhat a víz a ciszternába (amely a szerelők előnyét jelenti) vagy ezeknek kilyukasztásával kerülhetnek előnybe a szabotőrök.

- **Ősosztályok**

Container → Pipe

- **Interfészek**

-

- **Attribútumok**

-isLeaking: boolean	Ez az attribútum jelzi, hogy az adott cső ki van-e lyukasztva vagy sem
-isOccupied: boolean	Ez az attribútum jelzi, hogy valaki éppen az adott csövön közlekedik (tehát foglalt)
-waterFlowing: boolean	Ez az attribútum jelzi, hogy az adott csövön víz folyik át
-pumps: Pump[1..2]	A csőhöz tartozó pumpák listája (referenciaként tároljuk a csőhöz tartozó pumpákat)

- **Metódusok**

+steppable(): boolean	Azt adja meg, hogy a játékos ráléphet-e ezen Container-re, ez igaz értéket ad
+addPump(pu: Pump)	A csőhöz csatlakoztatott pumpák listájához újabb pumpa hozzáadását megvalósító függvény
+removePump(pu: Pump)	A csőhöz csatlakoztatott pumpák listájából kitöröl egy megadott pumpát

### 3.3.8 Player

- **Felelősség**

Egy általános játékost valósít meg (Mechanic, Saboteur)

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-position: Container	Ezzel tároljuk, hogy az adott játékos hol helyezkedik el (Itt megadnánk, hogy a Container az nem lehet Mountain Spring Cistern és LeakedWater típusú, mivel azon a játékos nem állhat)
-isTurn: boolean	Ez jelzi, hogy a játékos köre van
-carriedPipes: Pipe [ ]	A játékos által hordozott csövek listája (referenciaként tároljuk a játékos által cipelt csöveket)
-carriedPump: Pump	Ez tartalmazza a játékosok által, hordozott pumpát.

- **Metódusok**

+AdjustPump(pu: Pump, pi: Pipe, t: Type)	Ennek a metódusnak a segítségével állíthatják a játékosok a csöveket
+Move(c: Container)	A játékos ezzel a metódussal végzi el a mozgást (hasonlóan a positionnál a Container itt is csak cső és pumpa lehet)
+takeTurn()	Ez a metódus felelős a játékos kör kezdéséért
+attachPipe(pi: Pipe)	Ezen metódussal képes a játékos egy szabadvégű csőhöz újabb, nála lévő csövet csatlakoztatni

+attachPump(pu: Pump)	Ezen metódussal képes a játékos egy ketté vágott cső két része közé egy újabb, nála lévő pumpát csatlakoztatni
+detachPipe(pi: Pipe)	Ezzel lehet egy szabadvégű csövet lecsatlakoztatni a csőrendszeről
+takePipe(pi: Pipe)	Ezzel képes felvenni a játékos a ciszternában elkészített csövet
+takePump(pu: Pump)	Ezzel képes felvenni a játékos a ciszternában lévő pumpát

### 3.3.9 Pump

- **Felelősség**

Ez a pumpa, ez teszi lehetővé a csövek közötti összeköttetést

- **Ősosztályok**

Container → Pump

- **Interfészek**

-

- **Attribútumok**

-connectedPipes: Pipe[ ]	A pumpához csatlakoztatott csövek listája (referenciaként tároljuk a pumpához csatlakoztatott csöveget)
-input: Pipe	Ebben tároljuk, hogy a pumpában melyik cső a bemeneti cső (azaz melyikból akarjuk átpumpálni a vizet)
-output: Pipe	Ebben tároljuk a kimeneti csövet (azaz, hogy melyik csőbe akarjuk pumpálni a vizet)
-isDamaged: boolean	Ez az attribútum tárolja, hogy az adott pumpa éppen sérült-e vagy sem
-randomDamageValue: int	Azt az értéket tartalmazza amely meghatározza, hogy az adott pumpa mely körben fog megsérülni

- **Metódusok**

+steppable(): boolean	Igaz értéket ad vissza
+addPipe(pi: Pipe)	A pumpához csatlakoztatott csövek listájához újabb csövek adását megvalósító függvény
+removePipe(pi: Pipe)	A pumpához csatlakoztatott csövek listájából kitöröl egy megadott csövet

### 3.3.10 Saboteur

- **Felelősség**

Ez a szabotőr játékos, a szabotőr lyukaszthatja ki a csöveket illetve a szabotőr a szerelőhöz hasonlóan átállíthatja a pumpákat

- **Ősosztályok**

Player → Saboteur

- **Interfészek**

-

- **Attribútumok**

-

- **Metódusok**

+LeakPipe(p: Pipe)	Ezen metódussal lyukasztja ki a szabotőr az adott csövet
--------------------	--

### 3.3.11 Turn

- **Felelősség**

A kör megvalósításáért felelős

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

-turnCount: int	Játék kezdete óta eltelt körök száma
-----------------	--------------------------------------

- **Metódusok**

+turnStart(p: Player)	Adott játékos körét indítja el
+turnEnd()	Adott játékos körét befejezi
+increaseTurnCount()	Növeli a turnCount értékét a kör végén.
+evaluation()	Körönként frissíti a pályán lévő objektumokat.

### 3.3.12 Type

- **Felelősség**

A pumpa módosításának típusát határozza meg (azaz, hogy az AdjustPump(pi: Pipe, type: Type) függvénynél bemeneti csövet vagy kimeneti csövet változtatunk, amennyiben bemeneti csövet akkor a type értéke Input, amennyiben pedig kimeneti csövet a type értéke Output). Maga ez az osztály azért egy hasznos enumerációs osztály mivel így egy pumpa módosításánál egyszerűen megtudjuk változtatni azt a fajta csövet amelyet akarunk és

- **Ősosztályok**

-

- **Interfészek**

-

- **Attribútumok**

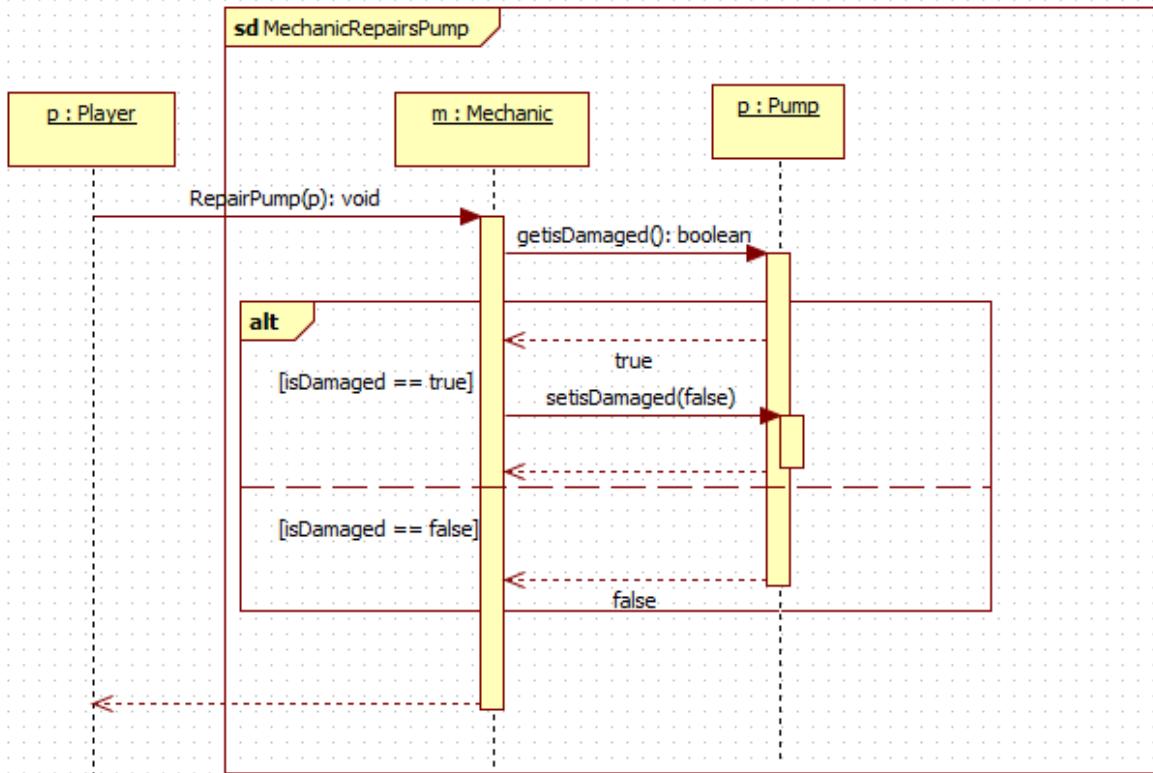
+Input	Ez a bemeneti csövet jelenti
+Output	Ez a kimeneti csövet jelenti

- **Metódusok**

-

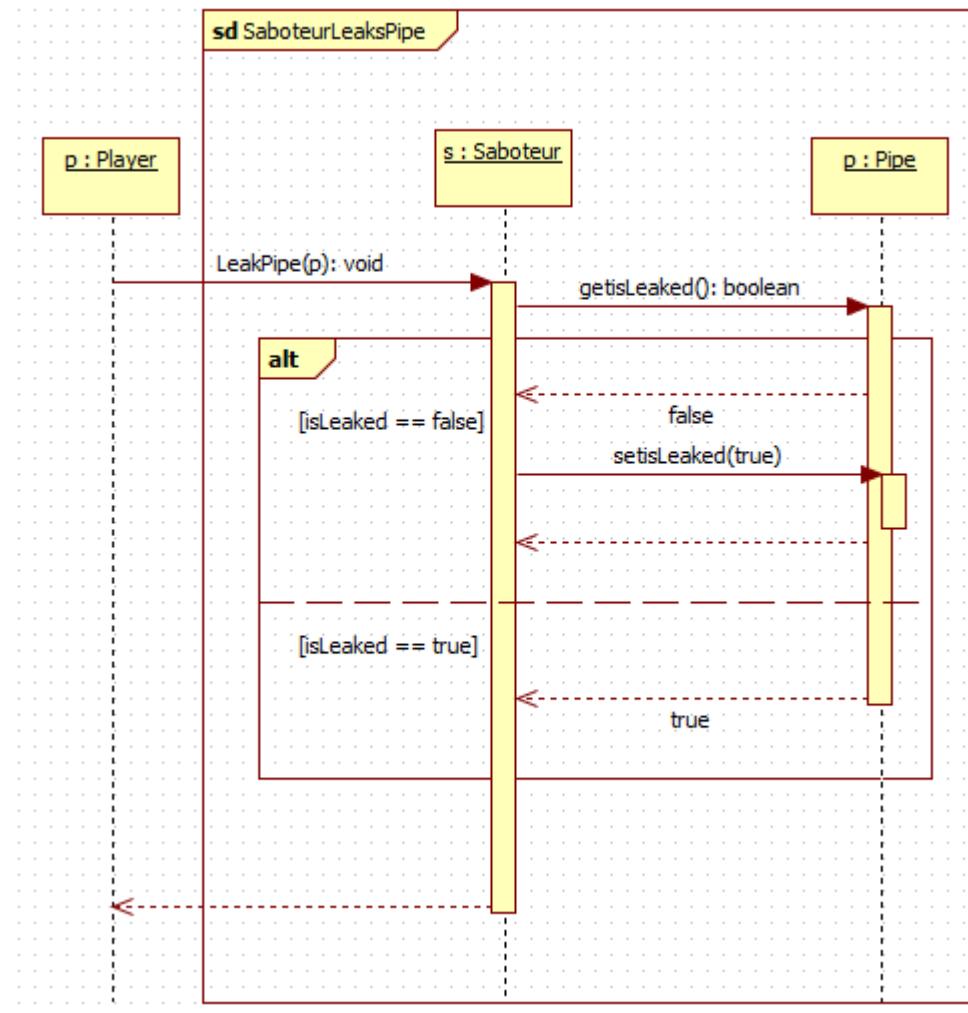
## 3.4 Szekvencia diagramok

### 3.4.1 Mechanic repairs pump



Komment: Mechanic repairs pump szekvencia diagramban Player hív ("call") `RepairPump(p)` (`p: Pump`), utána a Mechanic a Pump objektumon a "getisDamage()" függvény meghívásával ellenőrzi, hogy a szivattyú megsérült-e (függvény boolean visszaad a választ). Után alternative scenarios kezdeni, ahol `isDamaged` lehet igaz (`[isDamaged == true]` ez jelent, hogy Pump megsérült) vagy hamis (`[isDamaged == false]` Pump nem megsérült). Elsőben, Mechanic csőjavítás (A Mechanic új Pump szerel be a sérült szivattyú helyére, és csatlakoztatja a be- és kimeneti csöveket az új szivattyúhoz.). Ha nincs hiba, Mechanic vissza Playerhez.

### 3.4.2 Saboteur leaks pipe



Komment: **LeakPipe(p)** kezdeni a processzt. Saboteur megnézni ellenőrzi a szivárgó cső meglétét a következő függvény meghívásával **getIsLeaked()**. Utána alternative scenarios kezdeni, ahol **isLeaked** lehet igaz (**[isLeaked == true]** ez jelent, hogy Pipe szivárgó) vagy hamis (**[isDamaged == false]** Pipe nem szivárgó). Másodikban, ha a cső nem szivárgó (azaz a függvény(**getIsLeakde()**) hamis értéket ad vissza), a Saboteur folytatja a szabotázs folyamatot. (A szabotőr úgy lyukasztja ki a csövet, hogy a lehető legnagyobb mennyiségi víz távozzon.). Ha a cső már szivárgott, Mechanic vissza Playerhez.

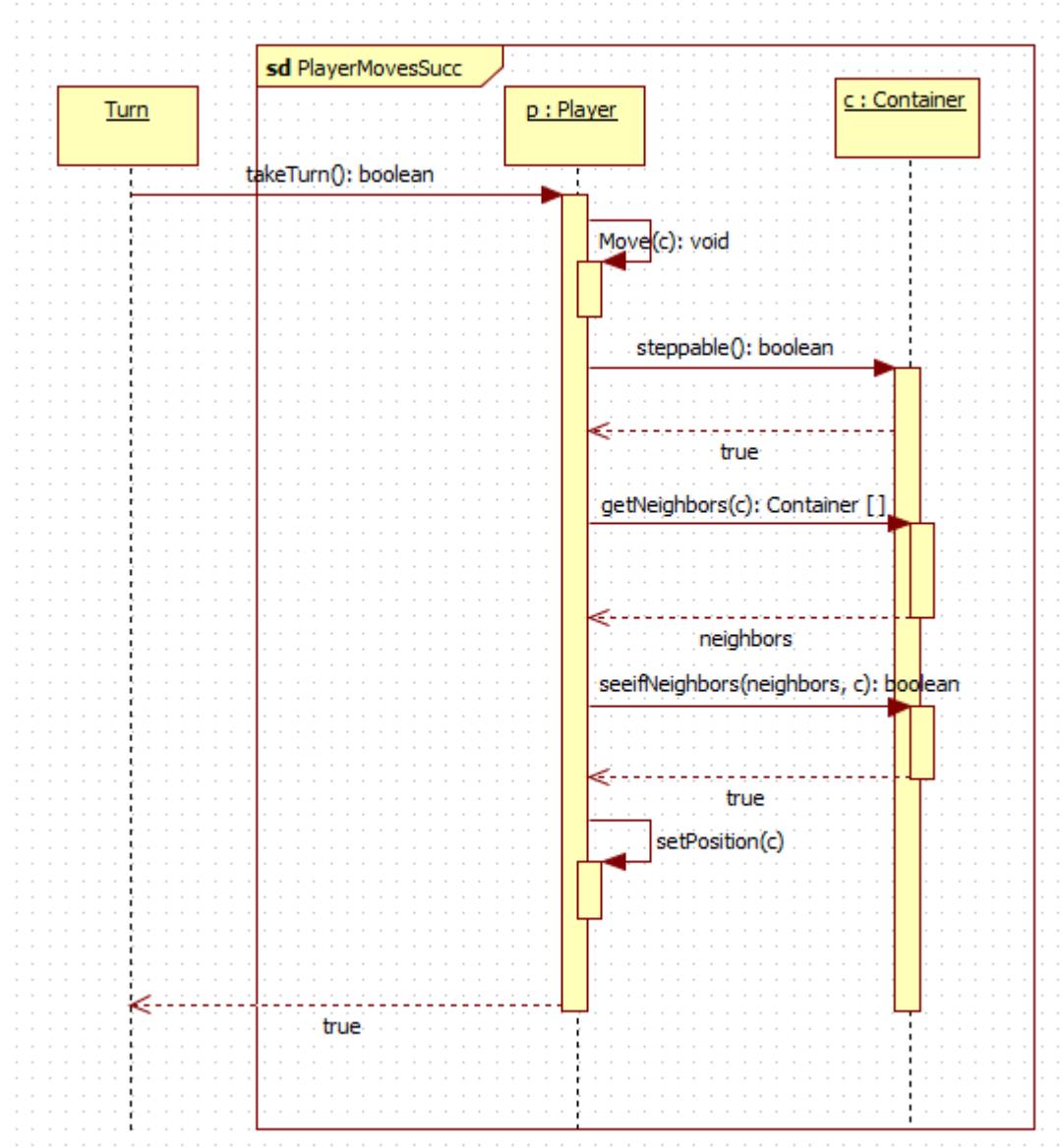
### 3.4.4.1 Player Moves Successfully

Komment: Először szeretném tisztázni, hogy Player Moves mind a 3 lehetséges forgatókönyvet (scenario) elkészítettük, mivel ez olvashatóbbá és érthetőbbé teszi a feladatunkat. takeTurn() jelen esetben a játékos mozgásának kiváltása.

Az elsőben azt fogjuk leírni, amikor a játékosnak sikeresen sikerült mozognia.

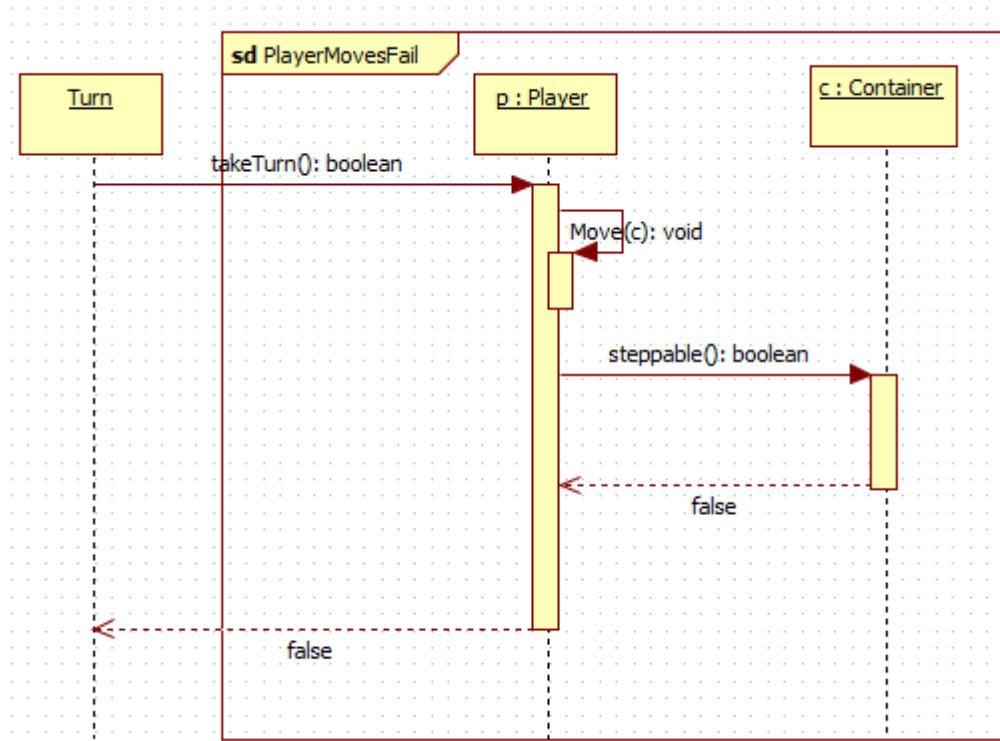
Másodikban a Player hamis(false) kap a steppable() metódustól, mert a játékos nem tud odafelé mozogni.

Harmadikban pedig azt, amikor a seeifNeighbours metódustól hamis(false) ad vissza, mert a kiválasztott objektum nem szomszédos, és ezért nem tud oda mozogni.



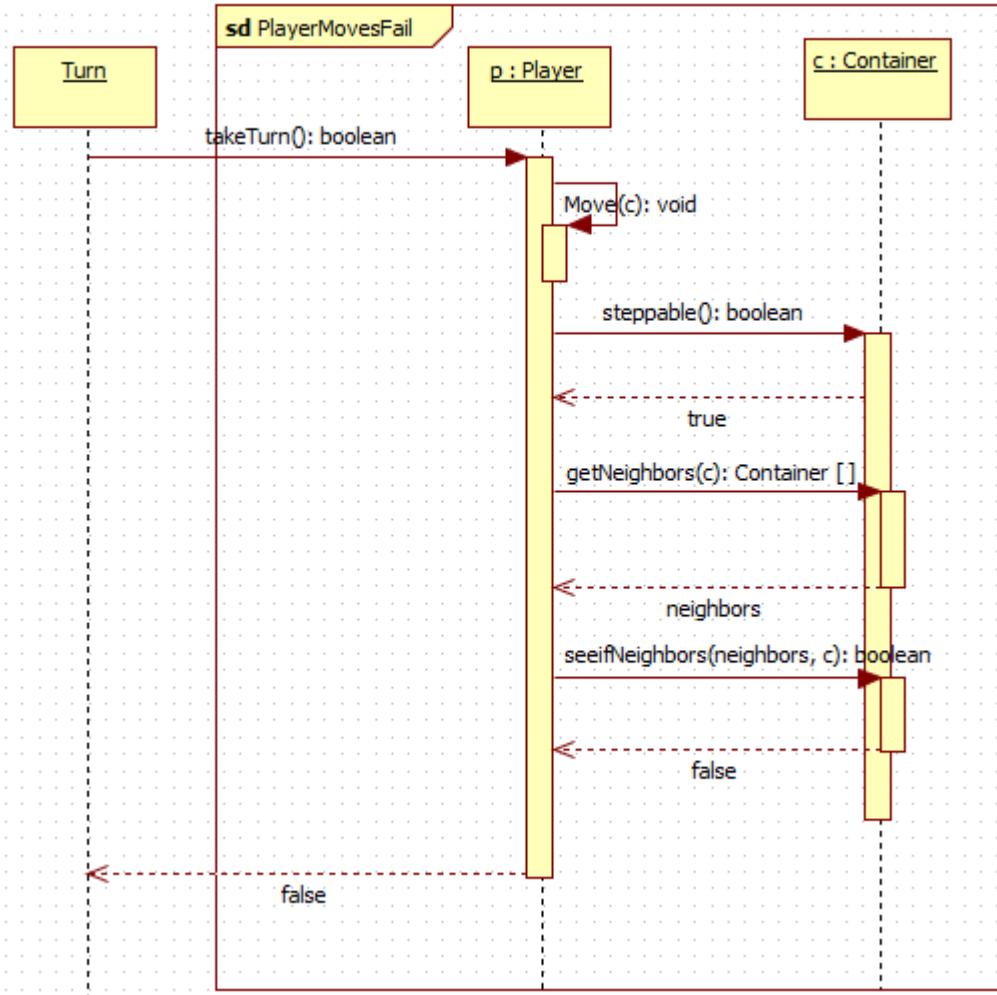
Komment: A Player elindítja a mozgást, majd a steppable() hívásával információt kér a Containertől a kiválasztott elemről, a sikeres eredmény után a játékos ellenőrzi, hogy az elem szomszédos-e, ha igen (ebben az esetben igen), akkor a Player új pozíciót állít be a setPosition(c)

### 3.4.4.2 Player Moves Fail I



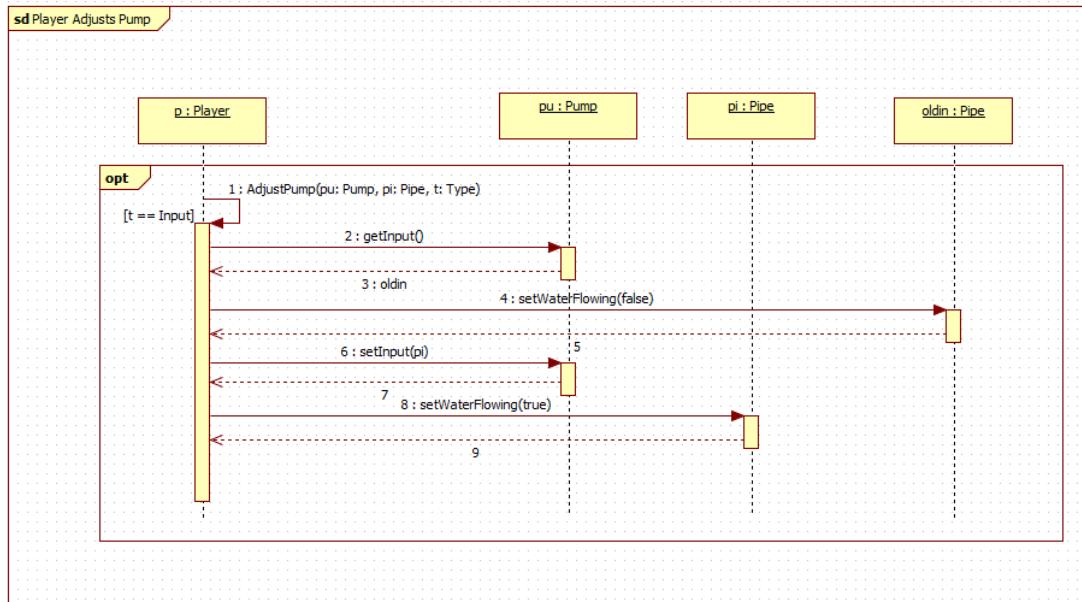
Komment: A Player elindítja a mozgást, majd a steppable() hívásával információt kér a Containertől a kiválasztott elemről, ha hamis(false) kapni, akkor a mozgás nem lehetséges.

### 3.4.4.3 Player Moves Fail II



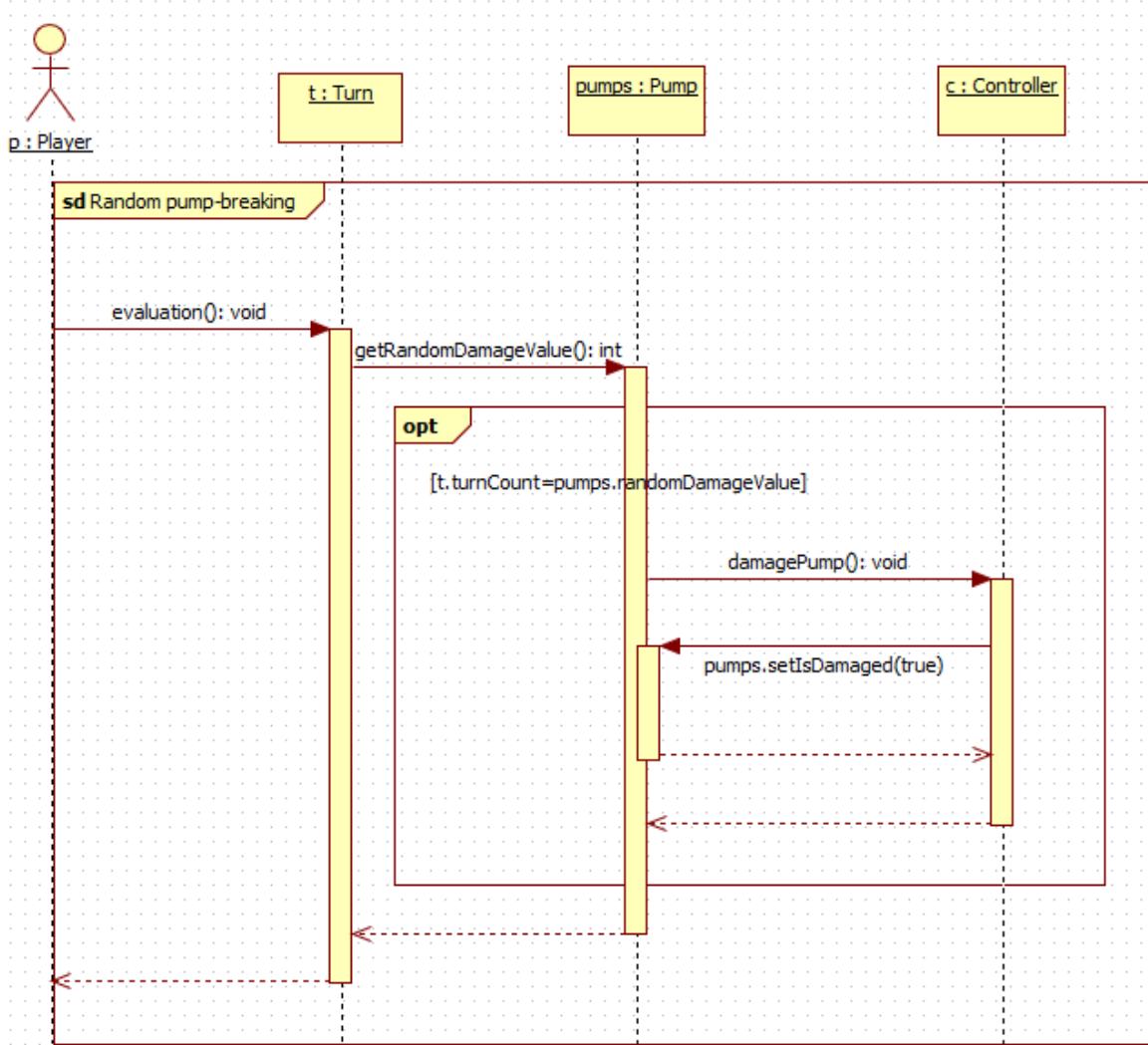
Komment: A Player elindítja a mozgást, majd a steppable() hívásával információt kér a Containertől a kiválasztott elemről, a sikeres eredmény után a játékos ellenőrzi, hogy az elem szomszédos-e, ha nem (ebben az esetben úgy), akkor a mozgás nem lehetséges.

### 3.4.5 Player Adjusts Pump



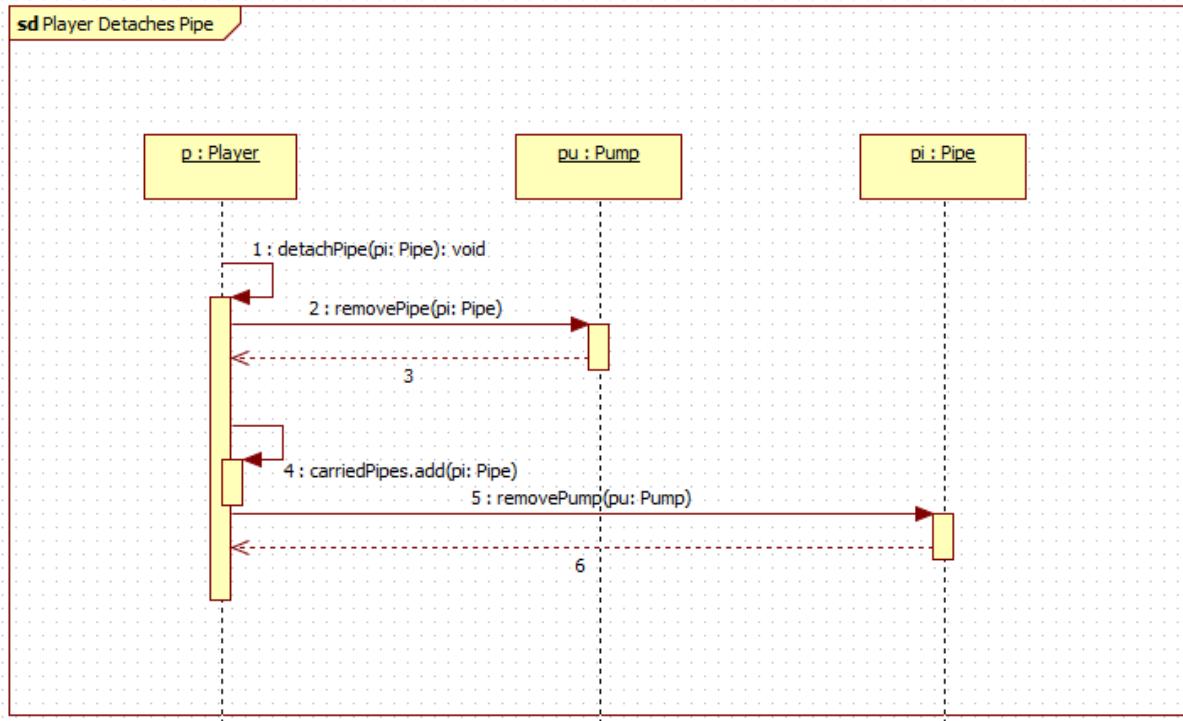
Komment: A játékos átállítja, hogy az adott pumpánál mely csőből mely csőbe folyjon a víz. Az `AdjustPump` metódus type paramétere jelzi, hogy a játékos a be- vagy kimeneti csövet szeretné átállítani. Ekkor a `getInput/getOutput` metódussal lekérdezzük a pumpától, hogy melyik cső volt eddig a be- vagy kimenet, ezután le kell tiltani az eddig be vagy kimenetet (`setWaterFlowing(false)`) és az újat pedig engedélyezni kell (`setWaterFlowing(true)`).

### 3.4.6 Random pump-breaking



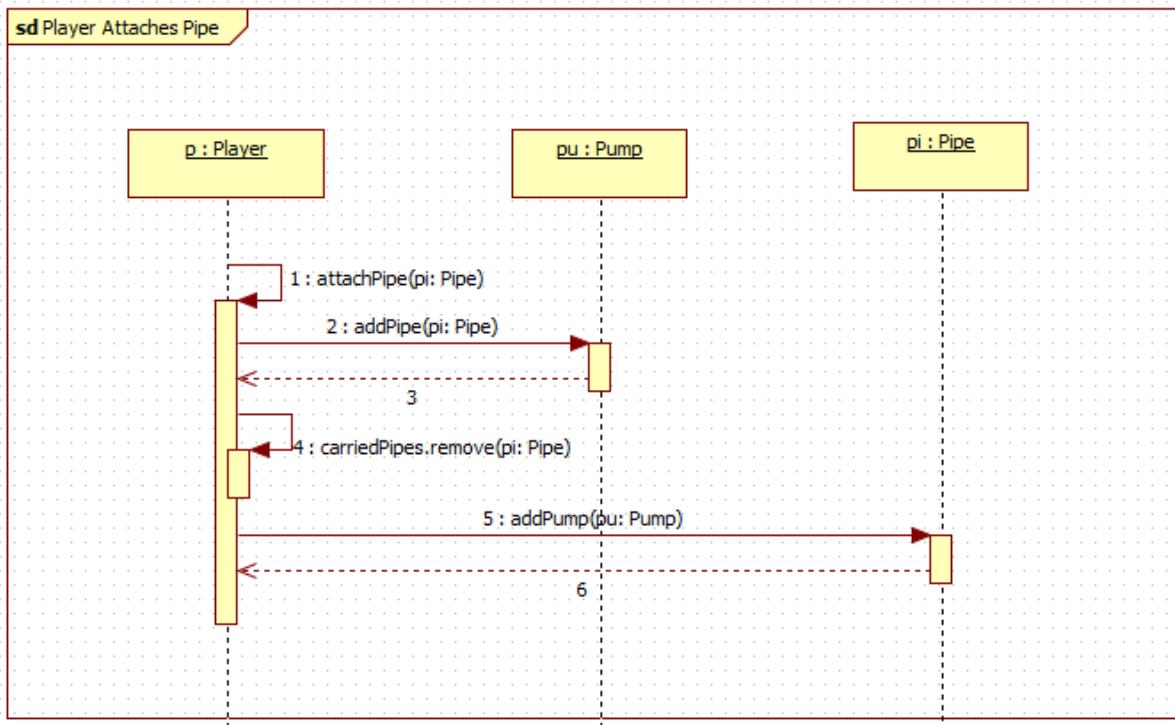
Komment: minden pumpához még a játék elején generáltunk egy randomDamageValue értéket (ezt most nem szemléltetjük, mivel ez egy random érték). minden kör után ezt az értéket lekérdezzük, és ha ez egyezik az körök számával, akkor azt a pumpát elrontjuk és az isDamaged értékét igazra változtatjuk. Ezzel az látható, hogy így véletlen időközönként elromlik egy pumpa.

### 3.4.7 Player Detaches Pipe



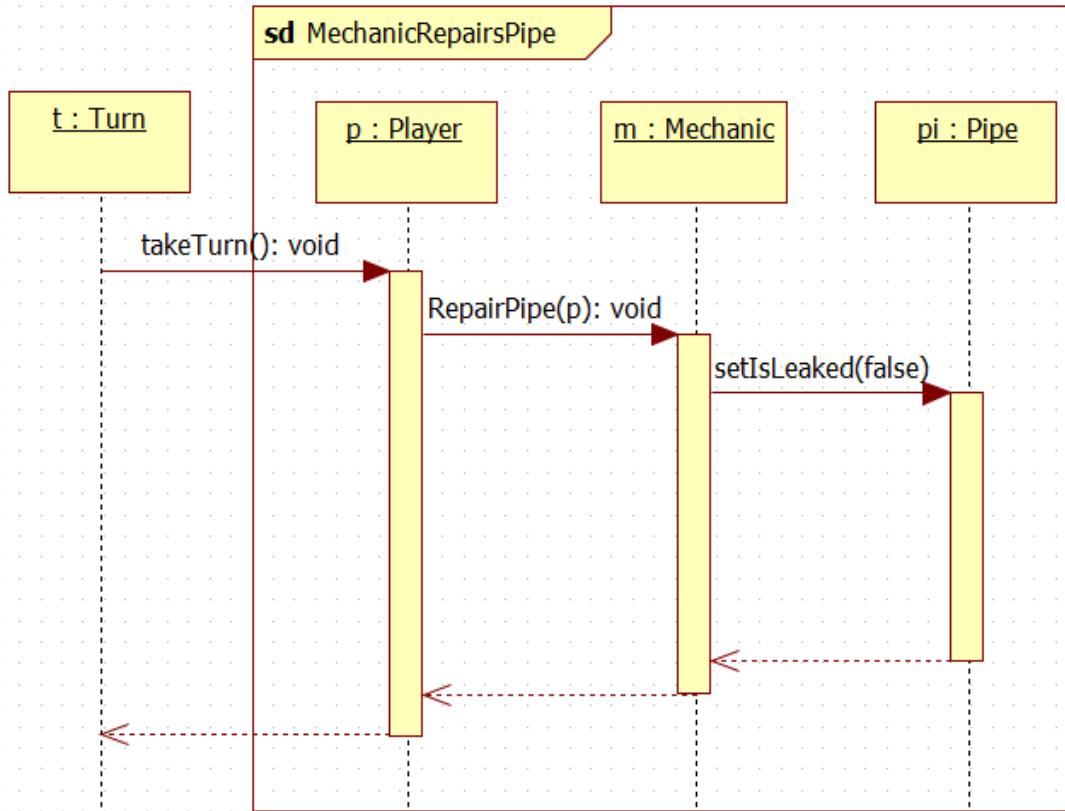
Komment: A játékos megszünteti a kapcsolatot egy cső és egy pumpa között(leválasztja a pumpáról a csövet). Ekkor a pumpától elvesszük a csövet a `removePipe` metódussal, majd a játékos `carriedPipes` attribútumához adjuk hozzá, majd a csőtől is eltávolítjuk a pumpát a `removePump` metódus meghívásával.

### 3.4.8 Player Attaches Pipe



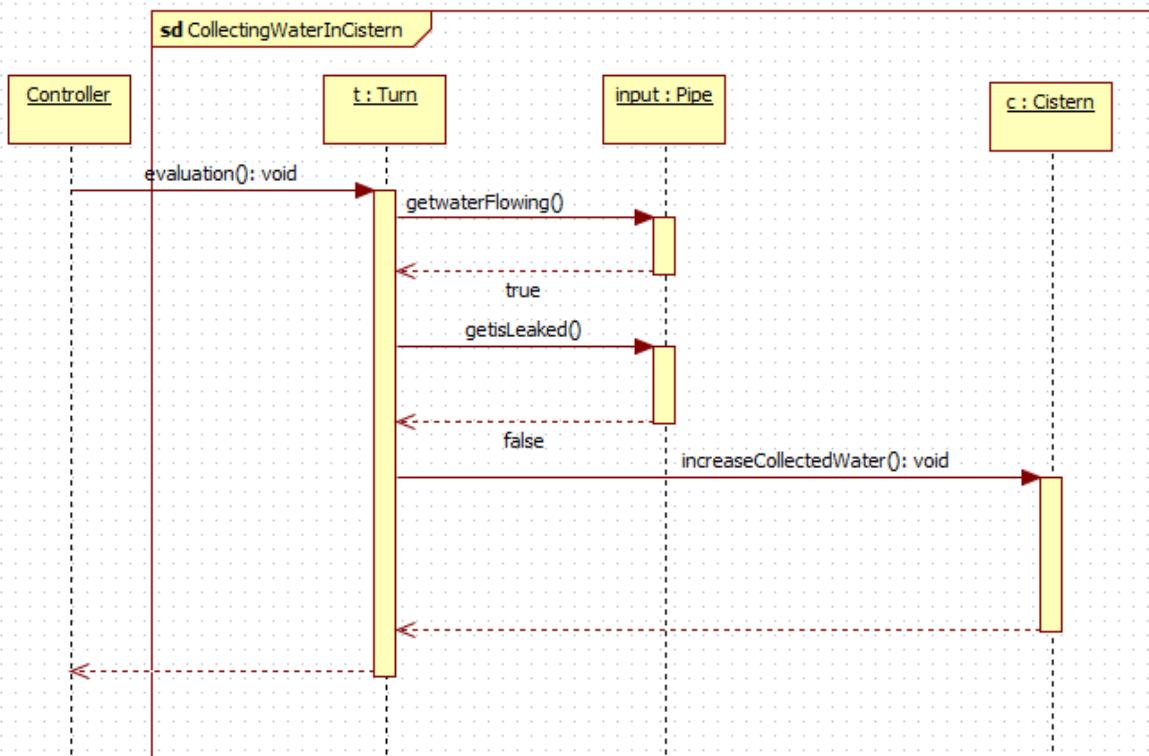
Komment: A játékos egy pumpához hozzákapcsol egy csövet. A pumpához hozzáadjuk a csövet az `addPipe` metódussal, a játékos `carrierPipes` attribútumából eltávolítjuk a csövet, végül a csőhöz hozzáadjuk azt a pumpát, melyre rákapcsoltuk (`addPump` metódus).

### 3.4.9 Mechanic Repairs Pipe



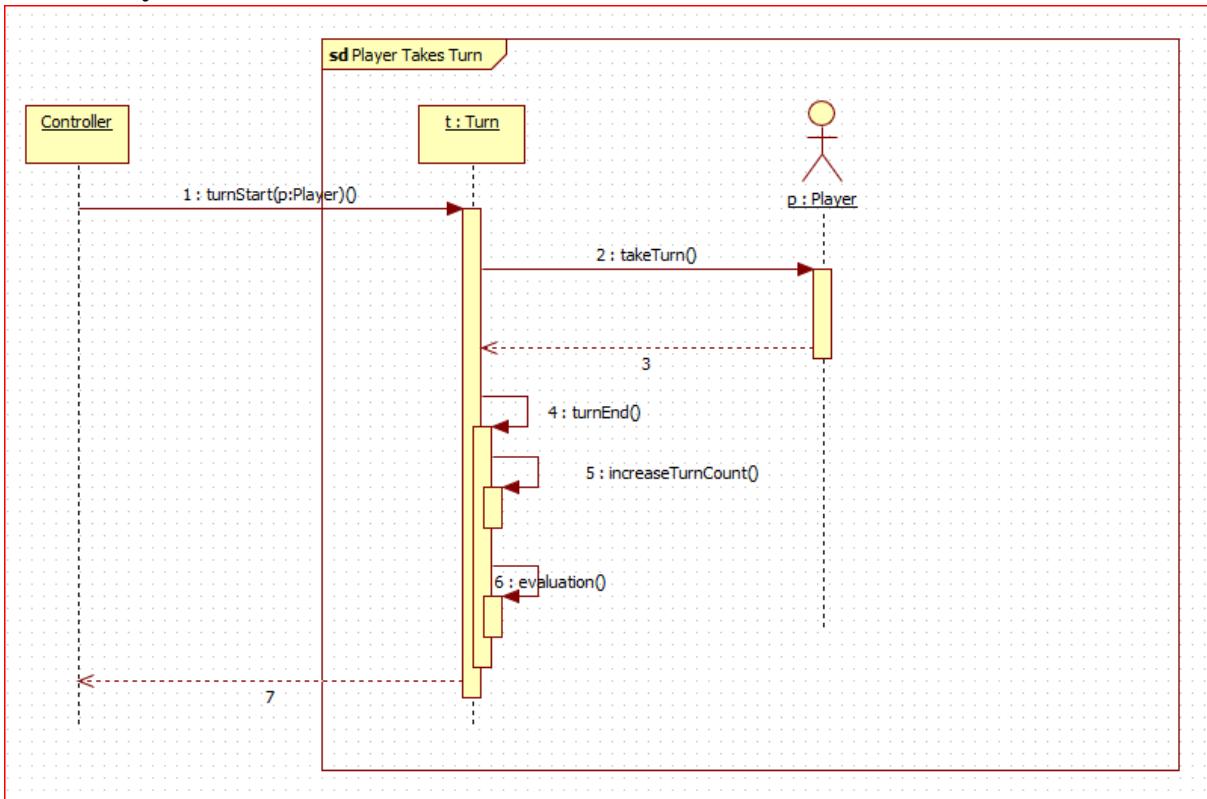
Komment: Feltételezzük hogy a Mechanik csőnél van és az lyukas és folyik benne víz. A Mechanik ekkor elhasználja a körét, arra hogy megjavítson egy Pipe-t, annak az isLeaked értékét false-ra állítva.

### 3.4.10 Add Collected Water



Komment: A ciszterna collectedWater attribútumának értékét növeli. Ehhez a ciszternába bemenő csőtől lekérdezzük, hogy törött-e (`getIsLeaked` metódus), ha nem törött, akkor növeljük a collectedWater attribútum értékét. Itt az előfeltevés, hogy a szekvencia diagramon megadott cső az a ciszterna bemeneti csöve, illetve a cső nem lyukasztott és víz folyik benne.

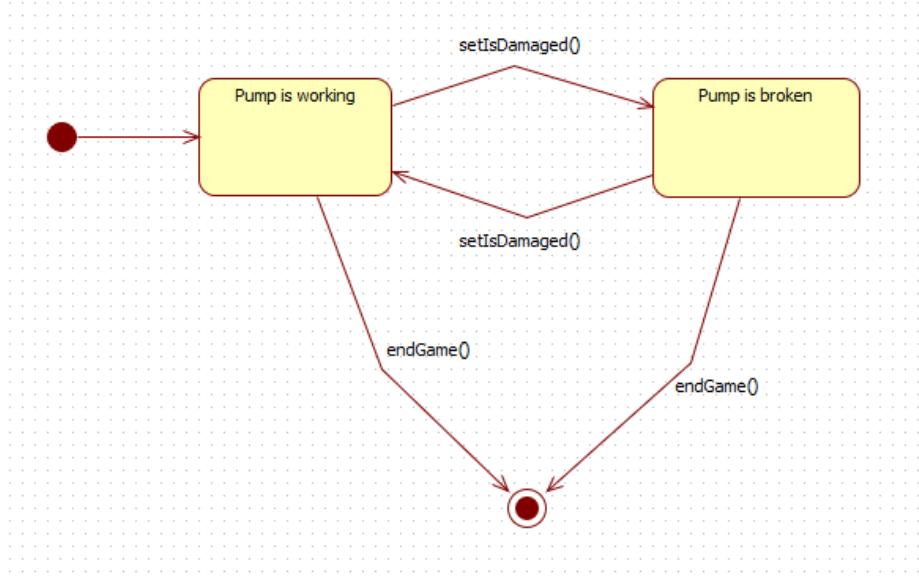
### 3.4.11 Player Takes Turn



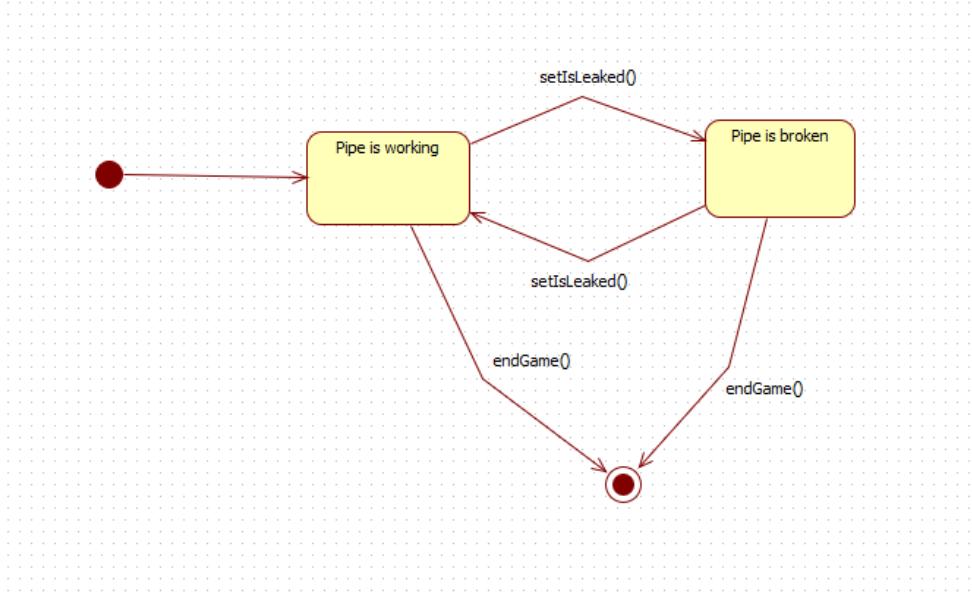
Komment: A játékos lép. A Turn osztály `turnStart` metódusát meghívjuk, amely meghívja a paraméterül kapott játékos `takeTurn()` metódusát. A játékos ekkor végrehajtja a lépést, amit szeretne, a függvény visszatérése után a `turnEnd` metódus hívódik meg, melyben megnöveljük az eddig lejátszott körök számát (`increaseTurnCount`), majd frissítjük a pályán lévő objektumokat (`evaluation`).

## 3.5 State-chartok

### 3.5.1 Pump State



### 3.5.2 Pipe State



### 3.6 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2023.03.25. 12:30	0.5 óra	Réti Ádám Tisza Miklós Kopach Artem Czifra Barnabás Molnár Márton	Értekezlet. Feladatok kiosztása. Feladatok megtervezése, különböző elemek megvitatása
2023.03.25 13:00	1 óra	Molnár Márton	3.4.4, 3.4.7, 3.4.8, 3.4.10, 3.4.11 Szekvenciadiagramok leírásának elkészítése.
2023.03.25 13:00	2 óra	Réti Ádám	3.4.6, 3.4.9 Szekvenciadiagramok újratervezése
2023.03.25 13:00	1 óra	Czifra Barnabás	3.4.5, 3.4.6, 3.4.9 Szekvenciadiagramok leírása
2023.03.22 14:00	1 óra	Tisza Miklós Réti Ádám	Értekezlet: 3.2 Osztálydiagram újratervezése
2023.03.25 12:00	4 óra	Tisza Miklós	Tevékenység: 3.2 Osztálydiagram változtatása 3.3 Osztály leírások változtatása 3.4.4 Szekvencia diagramok változtatása, létrehozása 3.4.10 Szekvencia diagram újratervezése
2023.03.25 12:00	4 óra	Kopach Artem	Tevékenység: 0. Új cím sablont, nyomtatás és beadás 3.4.1, 3.4.2, 3.4.3, 3.4.4.1, 3.4.4.2, 3.4.4.3 Szekvenciadiagramok leírása 3.4.1 - 3.4.11 hibákat találtam a diagramokban, átnéztem leírását, segített a diagramok újratervezésében

# Szkeleton tervezése

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

**Réti Ádám -  
csapatvezető**

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)      ket.vill.adam@gmail.com  
(E1LX0J)      tisza.miklos.16@gmail.com  
(NX9GA4)      barna.czifra@gmail.com  
(KLM60O)      molnar.marton.hun@edu.bme.hu  
(JQBOLI)      kopach.artem@edu.bme.hu

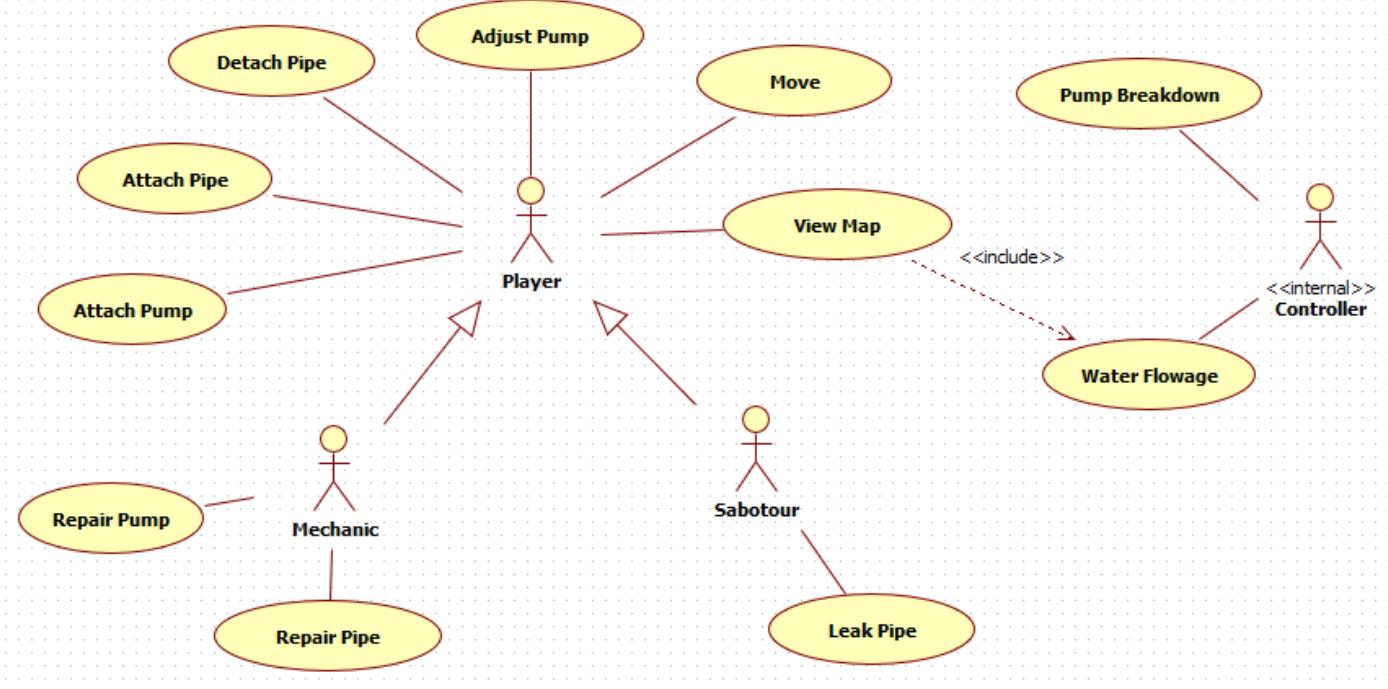
2023.04.02

## 5. Szkeleton tervezése

### 5.1 A szkeleton modell valóságos use-case-ei

Az use-case diagram a rendszer felhasználói és az azokkal való interakciót ábrázolja, és segít megérteni a rendszer működését.

#### 5.1.1 Use-case diagram



### 5.1.2 Use-case leírások

<b>Cím</b>	<b>Move</b>
<b>Leírás</b>	A szerelők és a szabotőrök csak a csöveken és a pumpákon haladhatnak. Kettő különböző játékos nem tud egyazon csövön állni.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékosok a csöveken és pumpákon haladva mozognak
<b>Kivétel Forgatókönyv</b>	A játékos egy olyan csőre próbálna mozogni, amelyen már valaki halad, ekkor a játékos nem tud azon a csövön közlekedni
<b>Alternatív Forgatókönyv</b>	A játékosok a pumpánál már megkerülhetik egymást

<b>Cím</b>	<b>View Map</b>
<b>Leírás</b>	A játékosok látják a pályán lévő csöveket és pumpákat
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	<ol style="list-style-type: none"> <li>1. A rendszer kirajzolja a csövek és pumpák aktuális állapotát</li> <li>2. A játékos megtekinti a csövek és pumpák aktuális állapotát</li> </ol>

<b>Cím</b>	<b>Adjust Pump</b>
<b>Leírás</b>	Minden pumpánál külön-külön állítható, hogy melyik csőből melyik másikba szállítson, pumpáljon vizet. A játékos csak akkor változtathatja a pumpa bemeneti csövét amennyiben azon keresztül víz nem folyik
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos átállíthatja, hogy melyik csőből melyik csőbe pumpáljon vizet a pumpa
<b>Kivétel forgatókönyv</b>	A játékos egy megsérült pumpát akarna átállítani azonban a pumpa sérültsége miatt ez nem lehetséges

<b>Cím</b>	<b>Attach Pipe</b>
<b>Leírás</b>	A játékos egy meglévő aktív elemhez(amelyen éppen tartózkodik) hozzácsatlakoztat egy csövet
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos hozzá csatlakoztat egy csövet azon aktív elemhez, amelyen éppen áll
<b>Kivétel forgatókönyv</b>	Az adott aktív elemhez nem lehet további csöveket csatlakoztatni. Ekkor nem történik meg a cső csatlakoztatása.
<b>Alternatív forgatókönyv</b>	Az adott csövet egy pumpához (egy pumpa típusú aktív elemhez) csatlakoztatjuk

<b>Cím</b>	<b>Detach Pipe</b>
<b>Leírás</b>	A játékos lecsatlakoztat egy csövet a csőrendszerből
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos sikeresen lecsatlakoztatja az aktív elemhez tartozó csövet
<b>Kivétel forgatókönyv</b>	A játékos egy olyan csövet próbál lecsatlakoztatni, ami két aktív elem között van.

<b>Cím</b>	<b>Attach Pump</b>
<b>Leírás</b>	A szerelők a ciszternáknál magukhoz tudnak venni új pumpát is, amit egy cső közepén tudnak elhelyezni. A csövet ehhez ketté kell vágni, és a két végét a pumpához kell csatlakoztatni.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	A játékos egy csövet kettévágva a keletkezett két szabadvégű cső szabad végéhez hozzá csatlakoztatja a pumpát

<b>Cím</b>	<b>Repair Pump</b>
<b>Leírás</b>	A szerelő játékosok javítják meg az elromlott pumpákat
<b>Aktorok</b>	Mechanic
<b>Forgatókönyv</b>	A szerelő megjavítja az elromlott pumpát
<b>Kivétel forgatókönyv</b>	A szerelő olyan pumpát akar megjavítani amely nincsen elromolva

<b>Cím</b>	<b>Repair Pipe</b>
<b>Leírás</b>	A szerelő játékosok javítják meg a kilyukasztott csöveget
<b>Aktorok</b>	Mechanic
<b>Forgatókönyv</b>	A szerelő megjavítja a kilyukasztott csövet
<b>Kivétel forgatókönyv</b>	A szerelő olyan csövet próbál megjavítani amely nincsen kilyukasztva

<b>Cím</b>	<b>Leak Pipe</b>
<b>Leírás</b>	A szabotőr játékosok kilyukasztják a csöveget
<b>Aktorok</b>	Saboteur
<b>Forgatókönyv</b>	A szabotőr kilyukasztja a csövet
<b>Kivétel forgatókönyv</b>	A szabotőr olyan csövet próbál kilyukasztani amely már ki van lyukasztva

<b>Cím</b>	<b>Pump Breakdown</b>
<b>Leírás</b>	A pumpák véletlen időközönként el tudnak romlani
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A pumpa meghibásodik

<b>Cím</b>	<b>Water Flowage</b>
<b>Leírás</b>	A víz folyását jelzi, ezt a gép irányítja
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A sivatagon keresztül csöveken keresztül szállítódik a víz
<b>Alternatív forgatókönyv</b>	A víz a hegyi forrásból származik, ebben véges mennyiségű víz van
<b>Alternatív forgatókönyv</b>	Ha egy csövön folyik át víz és a csövet kilyukasztják a csőből a víz szivárogni kezd
<b>Alternatív forgatókönyv</b>	Egy pumpán keresztül megy egy csőből egy másik csőbe a víz

## 5.2 A szkeleton kezelői felületének terve, dialógusok

A program a standard inputon várja a felhasználó parancsait. A főmenüben számok begépelésével érhetők el a menüpontok. Abban az esetben, amikor egyéb inputot vár a program, annak a formátumát jelzi a standard outputon. A főmenüben a következő menüpontok találhatók:

- Move
- View Map
- Adjust Pump
- Attach Pipe
- Detach Pipe
- Repair Pump
- Repair Pipe
- Leak Pipe
- Pump Breakdown
- Water Flowage

A következő táblázat tartalmazza a program bemeneteit és az azokhoz tartozó kimeneteket:

Bemenet	Kimenet
1. (Move) 2. Y/N (Is steppable true?) 3. Y/N (Is seeIfNeighbors true?)	takeTurn has started  Move has started  Steppable has started  Steppable has finished. Return value: true/false  getNeighbors has started  getNeighbors has finished. return value: "Neighbors[]"  seeIfNeighbors has started  seeIfNeighbors has finished. Return value: true/false  Move has finished.  takeTurn has finished. Return value: true/false
2. (View Map)	evaluation has started evaluation has finished

3. (Adjust pump)	<p>AdjustPump has started</p> <p>getInput/getOutput has started</p> <p>getInput/getOutput has finished. Return: oldin/oldout</p> <p>setWaterFlowing has started</p> <p>setWaterFlowing has finished</p> <p>etInput/setOutputt has started</p> <p>setInput/setOutput has finished</p> <p>setWaterFlowing ha started</p> <p>setWaterFlowing has finished</p> <p>AdjustPump has returned</p>
4. (Attach Pipe)	<p>attachPipe has started</p> <p>addPipe has started</p> <p>addPipe has returned</p> <p>addPump has started</p> <p>addPump has returned</p> <p>attachPipe has returned</p>
5. (Detach Pipe)	<p>detachPipe has started</p> <p>removePipe has started</p> <p>removePipe has returned</p> <p>removePump has started</p> <p>removePump has returned</p> <p>detachPipe has returned</p>
6. Repair Pump	<p>repairPump() has started</p> <p>IsDamaged: true</p> <p>SetIsBroken: false</p> <p>IsDamaged: false</p> <p>repairPump() has finished</p>

7. Repair Pipe	takeTurn() has started repairPipe() has started SetIsLeaked: false repairPipe() has finished takeTurn() has finished
8. Leak Pipe Is the pipe leaked(Y/N)	LeakPipe has started  getIsLeaked has started  getIsLeaked has returned. Return value: true/false  setIsLeaked has started  setIsleaked has returned  LeakPipe has returned
9. Pump Breakdown	evaluation has started SetIsDamaged(): true evaluation has finished
10. Water Flowage	evaluation has started waterFlow has started eval has started setInputState has started setInputState has finished eval has finished makeHistory has started makeHistory has finished evaluation has finished

### 5.3 Szekvencia diagramok a belső működésre

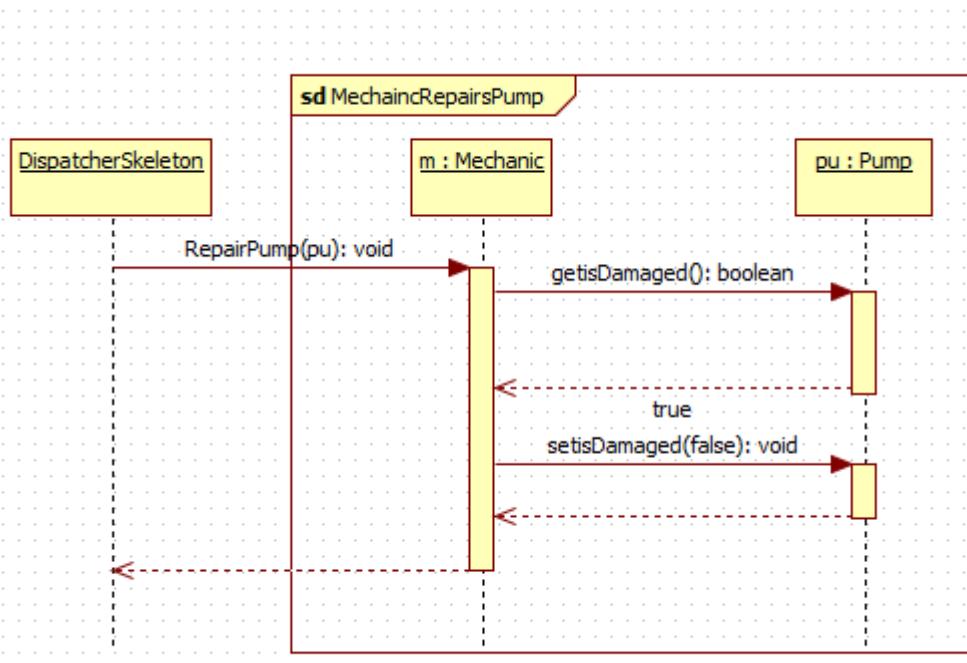
**Megjegyzés:** A következő szekcióban az alábbi rövidítés a következőt jelenti:

**DS->Dispatcher Skeleton**

**Első fő objektum: Player (, amelyben lesznek Mechanic és Saboteur)**

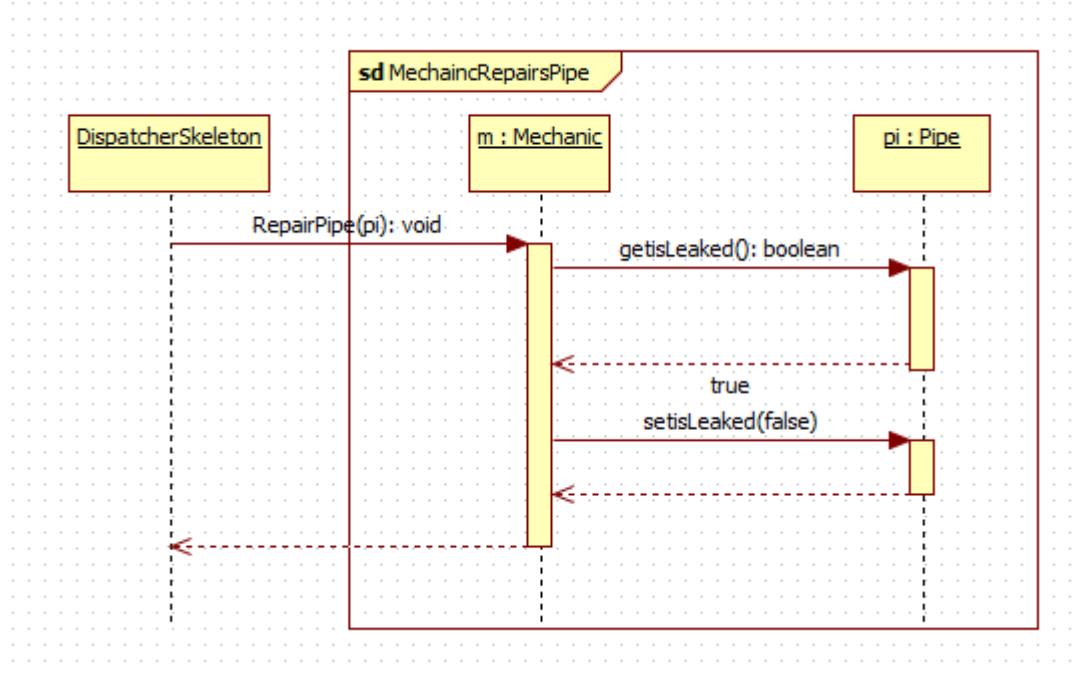
**Komment:** Úgy döntöttünk, hogy a Mechanic és a Saboteur egy közös Player objektum példányaként ábrázolod, mivel sok hasonló tulajdonságuk és viselkedésük van a Use-Case diagramában (és a játékban). Mindketten emberi játékosok, akik képesek a csöveken mozogni és a csőrendszerrel interakcióba lépni, de különböző célokkal és stratégiák. Lesznek sequence diagramok, amelyik bemutatják ezen objektumok (Mechanic és Saboteurs) egyes funkciót.

### 5.3.1 Mechanic repairs pump ( Use-Caseben Mechanic - Repair Pump)



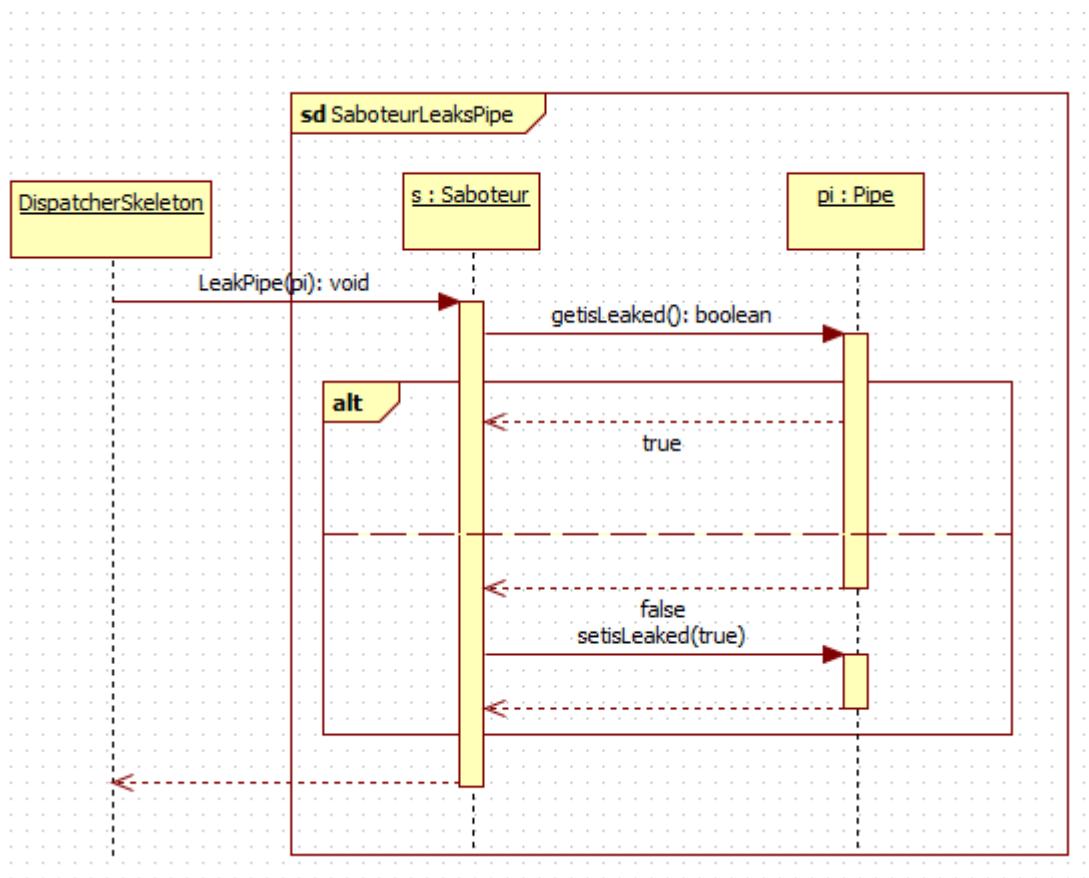
**Komment:** A DispatcherSkeleton osztályunk meghívja az (m) Mechanic objektumra a RepairPump(pu) függvényt amely elsősorban lekérdezi a (pu) Pump objektum isDamaged attribútumát (annak érdekében, hogy olyan pumpát ne tudjon megjavítani a szerelő amely pumpa alapjáraton nem sérült). Erre a lekérdezésre megkapjuk a pumpa állapotát és ha true értéket kapunk egyszerűen a pumpa isDamaged attribútumát egy setter függvénnnyel átállítjuk.

### 5.3.2 Mechanic repair pipe (Use-Caseben Mechanic - Repair Pipe)



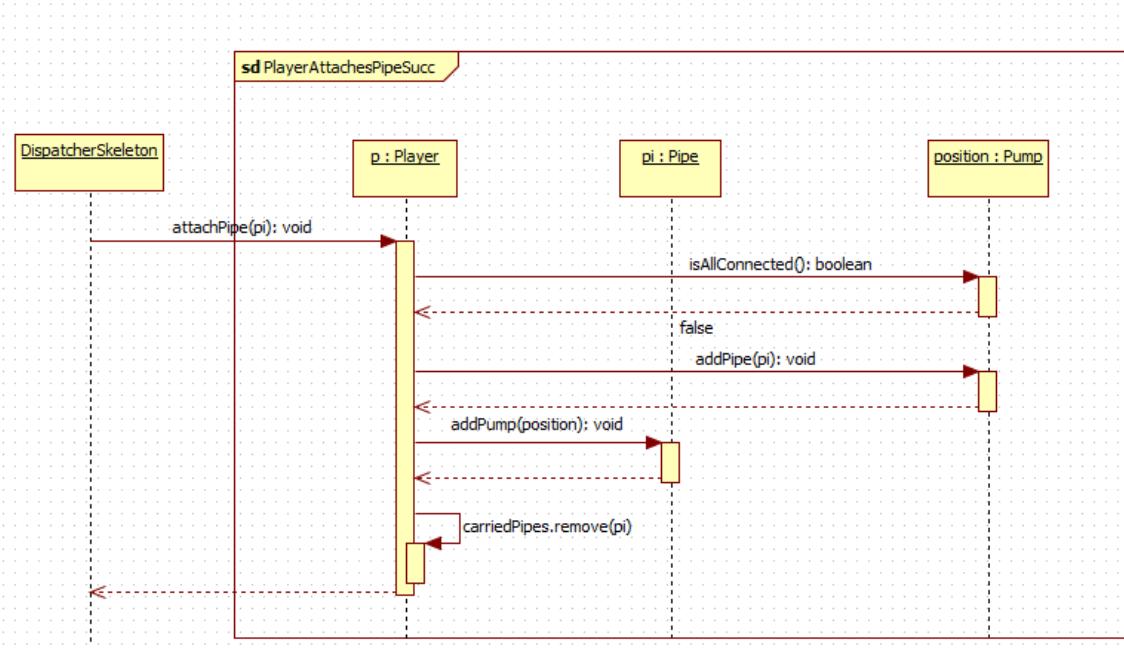
**Komment:** Hasonlóan az előző szekvencia diagramhoz a DispatcherSkeleton osztályunk meghívja az (m) Mechanic objektumra a RepairPipe(pi) függvényt amely elsősorban lekérdezi a (pi) Pipe objektum isLeaked attribútumát (annak érdekében, hogy olyan csövet ne tudjon megjavítani a szerelő amely cső alapjáraton nem sérült). Erre a lekérdezésre megkapjuk a cső állapotát és ha true értéket kapunk egyszerűen a cső isLeaked attribútumát egy setter fügvénnyel átállítjuk.

### 5.3.3 Saboteur leak pipe (Use-Caseben Player - Leak Pipe)



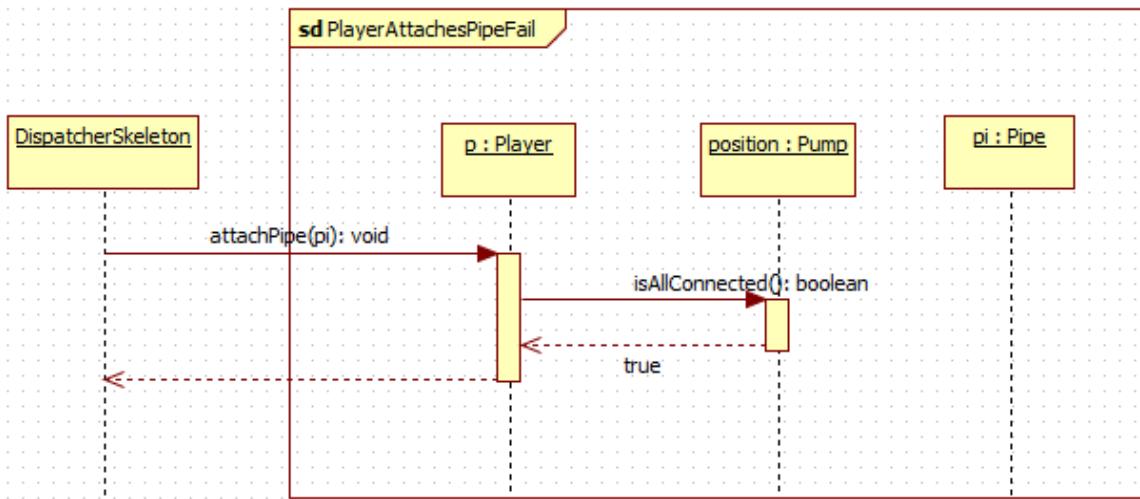
**Komment:** Ebben a szekvencia diagramban tüntejük fel a szabotőr csőlékelési akcióját. A `LeakPipe(pi)` függvényt meghívja a DS (Dispatcher Skeleton) osztályunk. Ezt követően lekérdezi a cső aktuális állapotát (azaz a cső éppen lyukas vagy sem) amely azért szükséges számunkra, hogy megtudjuk, hogy érdemes-e bármilyen akciót végeznünk az adott objektumon. Amennyiben false értékkel tér vissza a getter függvényünk egy setter metódus hívásban átalítjuk a (pi) Pipe isLeaked attribútumának értékét true-ra

### 5.3.4 Player attach pipe Successful (Use-Caseben Player - Attach Pipe)



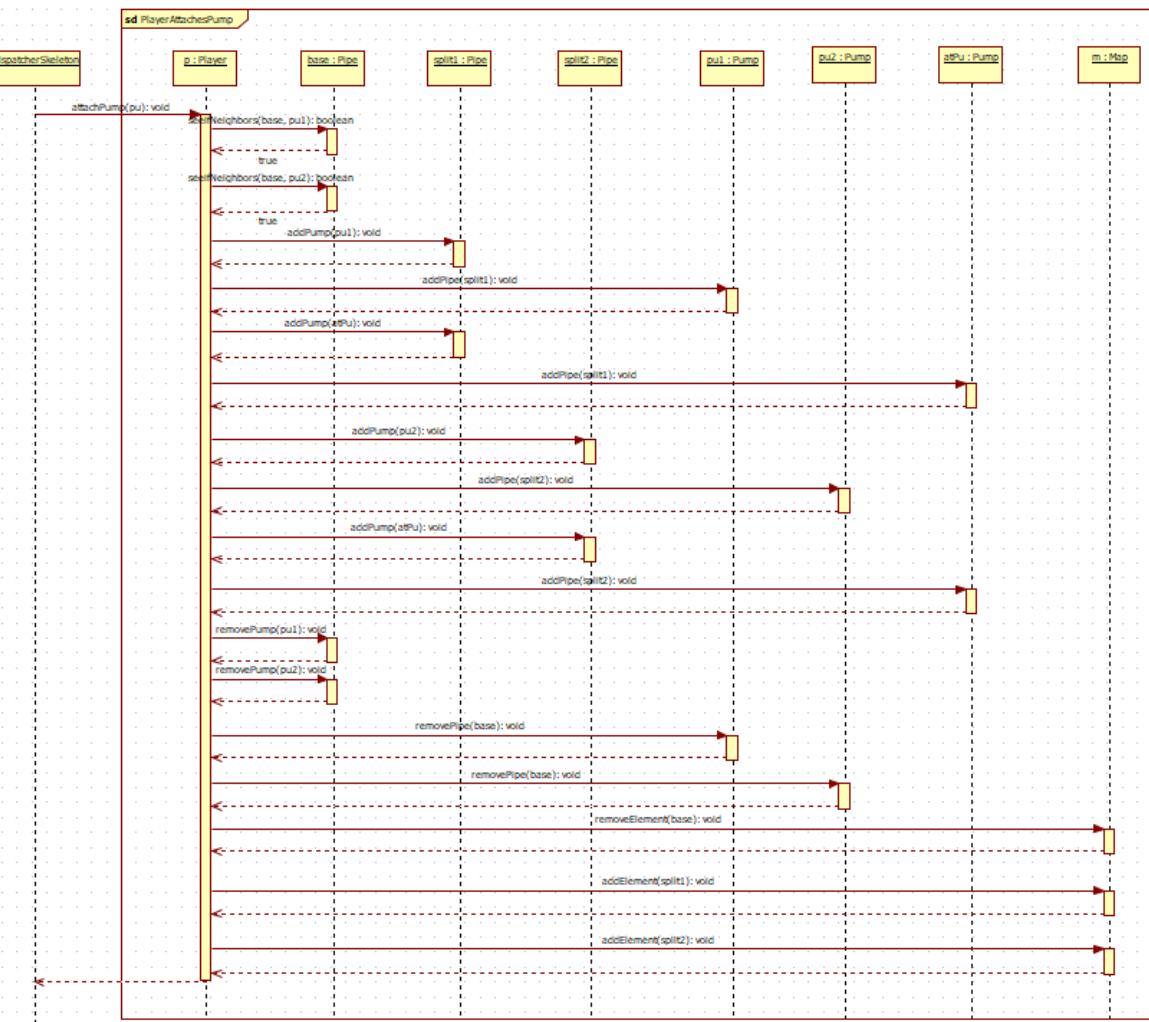
**Komment:** Ebben a szekvencia diagramban tüntejük fel a játékosok/játékos csőillesztési akcióját. A DS meghívja a (p) Player-re az `attachPipe(pi)` függvényt, ezután ezen függvényen belül lekérdezzük egy `isAllConnected()` metódus segítségével a (position) Pump objektumtól, hogy az összes csatlakoztatási helyre csatlakoztatva van-e cső. Amennyiben nem (azaz `false` értéket ad vissza) a pumpa cső listájához egy `addPipe(pi: Pipe)` függvény segítségével hozzáadjuk a csövet a pumpához, majd ez egy `addPump(pu: Pump)` függvénytel megte tesszük a csőnél is (tehát mind a csőhöz, mind a pumpához hozzáadtuk az egyiket). Legvégül a játékos `carriedPipes` listájából kivesszük az éppen lerakott cső elemét.

### 5.3.4.2 Player attach pipe Fail (Use-Caseben Player - Attach Pipe)



**Komment:** Az előző szekvencia diagram egy kivétel forgatókönyve ahol azért nem történik meg a cső pumpához illesztése mivel az `isAllConnected()` függvény egy hamis (azaz egy `boolean false`) értéket adott vissza, ezzel jelezve, hogy a pumpához már a maximális számú cső tartozik ezért nem lehet egy újabbat hozzáilleszteni.

### 5.3.5 Player attaches pump (Use-Caseben Player - Attach Pump)

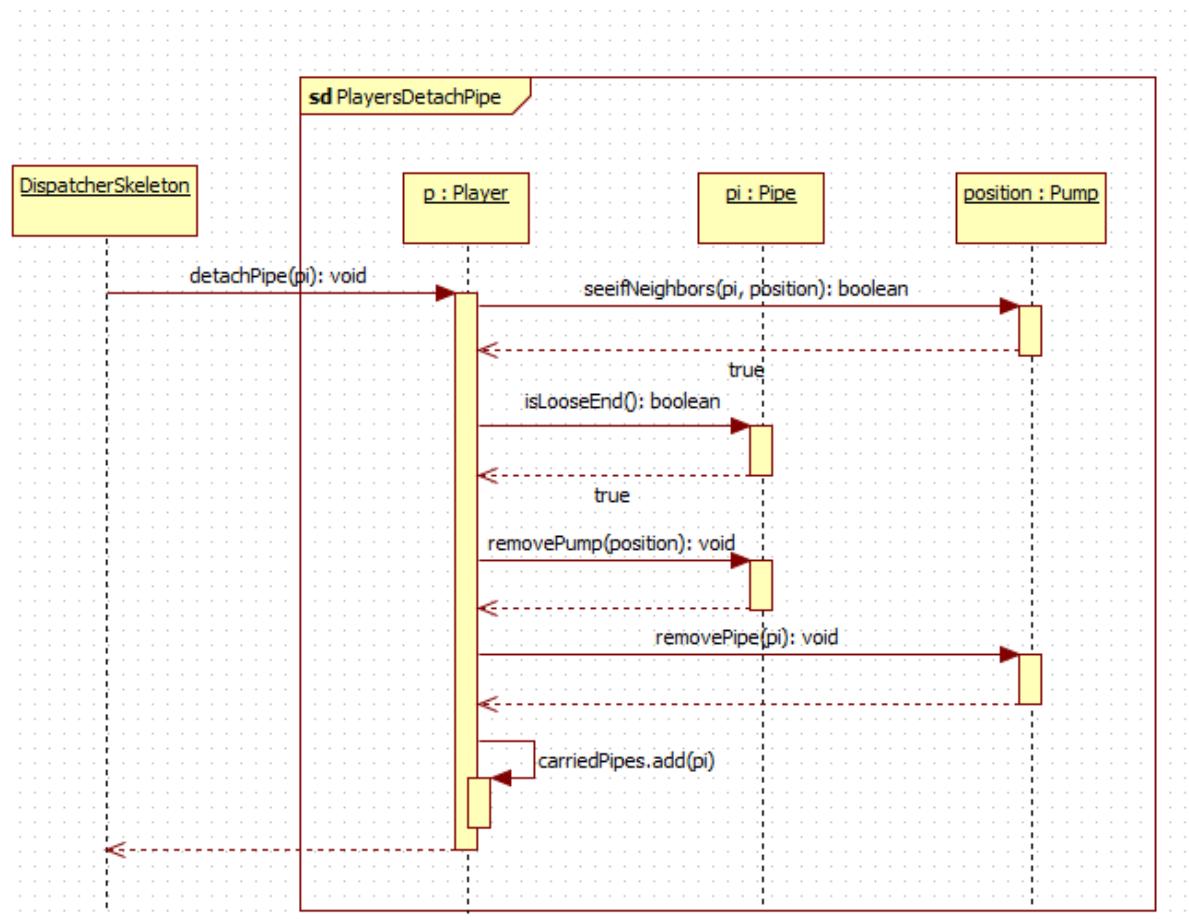


**Komment:** Ezen szekvencia diagram mutatja meg az osztályok egymással való kommunikációt egy pumpa illesztés akciónál. A pumpa illesztésének folyamatát (most itt éppen) a DS kezdeményezi egy attachPump(pu) függvénytel. A pumpát azon csőhöz fogjuk illeszteni amelyen éppen a játékos áll, jelen esetben (base) Pipe objetumunkhoz. Az első fázisa az illesztésnek a (base) Pipe felosztása illetve a környező pumpák lekérdezése (ezt egy getNeighbor() függvény hívásánál kapjuk meg azonban az egyszerűség kedvéért most itt csak lekérdeztünk kettő adott pumpához hogy szomszédja-e (base) Pipe-nak). Mivel az illesztés úgy megy végbe, hogy a játékos félre vágha egy csövet ezért itt létre kell hoznunk két másik cső objektumot jelen esetben (split1) Pipe és (split2) Pipe objektumokat. Ezek létrehozása után egyenként meg kell tennünk az illesztéseket (példaként kövessük végig (split1) Pipe objektumunkat):

- A (split1) Pipe objektum pumpa lista-jába hozzáadjuk az újonnan beillesztett pumpát (atPu) illetve az eredeti csőhöz (base) tartozó valamely szomszédját (jelen esetben (pu1) csövet)
- Ezután értelemszerűen hozzáadjuk (pu1) és (atPu) pumpák cső listájához (split1) csövet

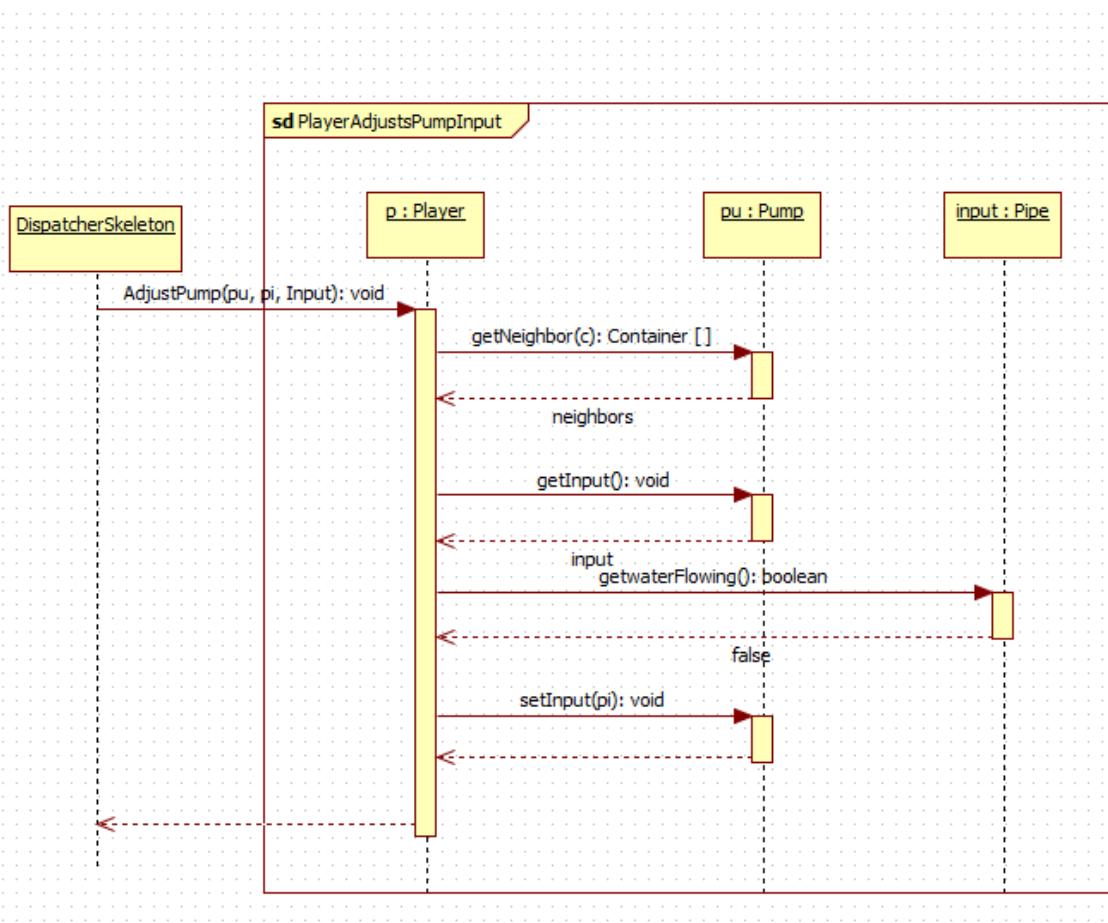
Teljesen hasonlóan ugyanezt megcsináljuk (split2) csőnél is (csak a (base) cső másik szomszédjával). Legvégül kiszedjük (base) Pipe-ot a szomszédos pumpák csőlistájából majd kiszedjük (removeElement(base)) a Map osztályunk konténer listájából és hozzáadjuk a kettő újonnan született csövet a listához (addElement(split1), addElement(split2))

### 5.3.6 Player detach pipe (Use-Caseben Player - Detach Pipe)



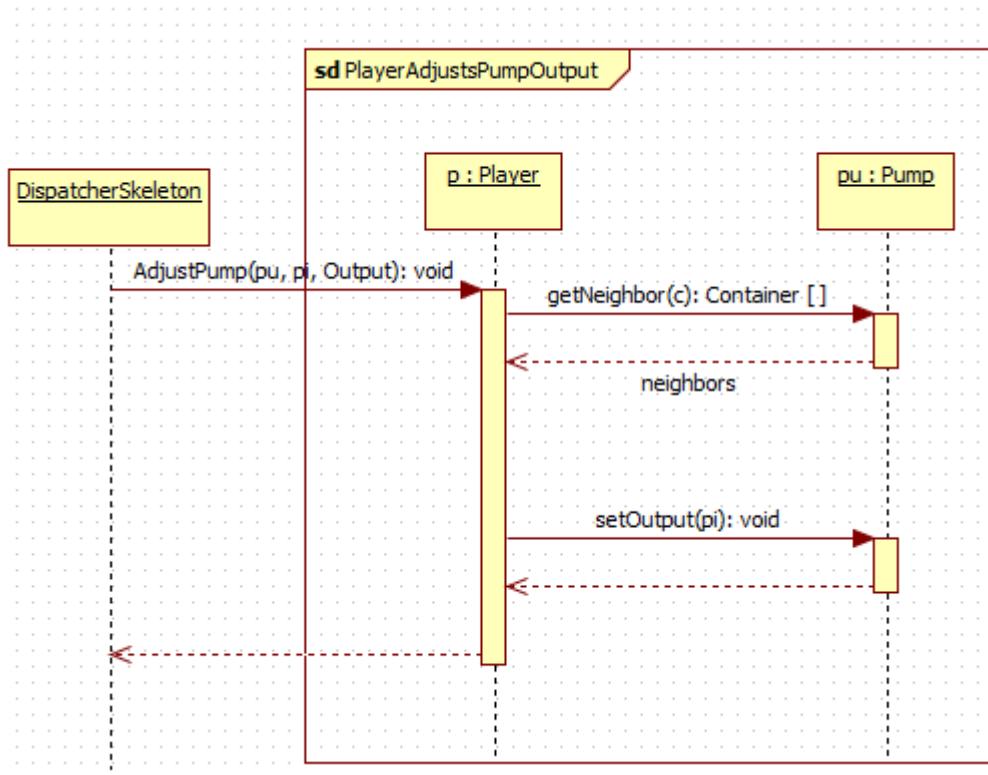
**Komment:** Ebben a szekvencia diagramban a játékos (p) Player csőleillesztési akcióját követjük végig. A DS meghívja a `detachPipe(pi)` metódusát (p) Player-nek amelyben a következő metódus hívások történnek meg: `seeifNeighbors(pi, position)` függvény segítségével megvizsgáljuk , hogy a két vizsgált objektum szomszédos-e egymással, ez visszaad egy boolean értéket jelen esetben `true` (`false` esetet felesleges megvizsgálni hiszen azon esetben nem történik semmi és visszatér teljesen a `detachPipe(pi)` függvény). Ezt követően a (pi) Pipe-ra meghívunk egy olyan függvényt amely egy logika kifejezést visszadvva jelzi, hogy az adott cső szabadvégű cső-e (`isLooseEnd()`). Amennyiben a függvény visszatérési értéke `true` akkor elkezdhetjük elvágezni a leillesztését a csőnek. Először is cső pumpa listájából kiszedjük a (position) pumpát majd ugyanezt fordítva megtesszük a pumpával is. Legvégül a játékos által hordozott csövek listájába beillesztjük az éppen leválasztott csövet.

### 5.3.7.1 Player adjust pump Input (Use-Caseben Player - Adjust Pump)



**Komment:** Ebben a szekvencia diagramban egyszerűen azt követjük végig, hogy a játékos a pumpa Input csövét hogyan is állítja át egy másik csőre. A DS meghívja (p) Player objektumra az `AdjustPump(pu, pi, Input)` függvényt. Ezt követően a függvényen belül lekérdezzük (pu) pumpa szomszédjainak listáját (Ez csak abból a szempontból fontos, hogy az `AdjustPump` argumentumában megadott (pi) Pipe szomszédja-e (pu) Pump-nak más különben nem foglalkozunk tovább ezzel és visszatérünk). Amennyiben az argumentumban megadott cső ténylegesen szomszédja (pu) Pump-nak lekérdezzük, hogy a (pu) Pump mostani Inputjában folyik-e éppen víz (amennyiben folyik víz benne azaz a `getWaterFlowing()` true értékkel tér vissza nem tudjuk megváltoztatni az inputot, mivel ez egy fontos feltevés volt a use-case leírásoknál is). Legvégül az argumentumban kapott (pi) csővet egy setter segítségével megtesszük (pu) Pump inputjává.

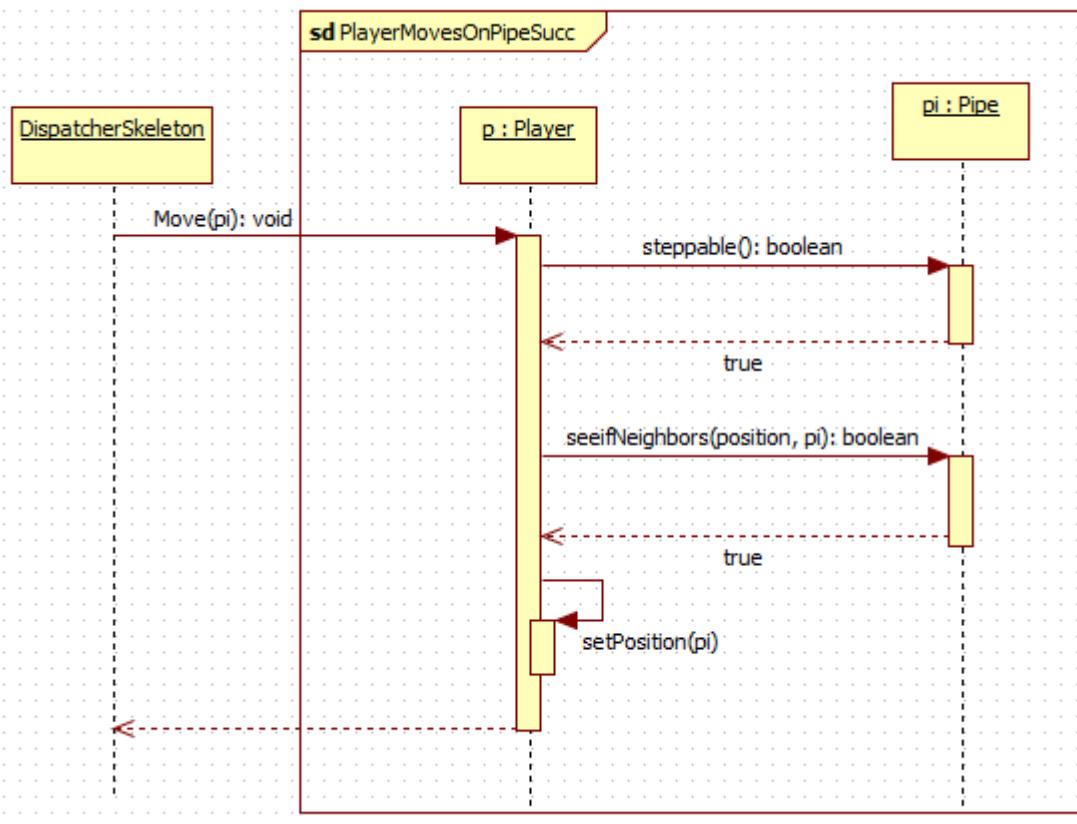
### 5.3.7.2 Player adjust pump Output (Use-Caseben Player - Adjust Pump)



**Komment:** Ennek a szekvencia diagramnak a leírása egy ponttól elvée teljes mértékben megegyezik az előző leírásával. Itt az Output csövet változtatjuk meg de mivel nincsenek számunkra nem kedvező feltételek az Output cső változtatására ezért egyszerűen csak megvizsgáljuk a két elem szomszédosságát majd amennyiben szomszédosak egymással a megadott csövet a pumpa Outputjának tesszük.

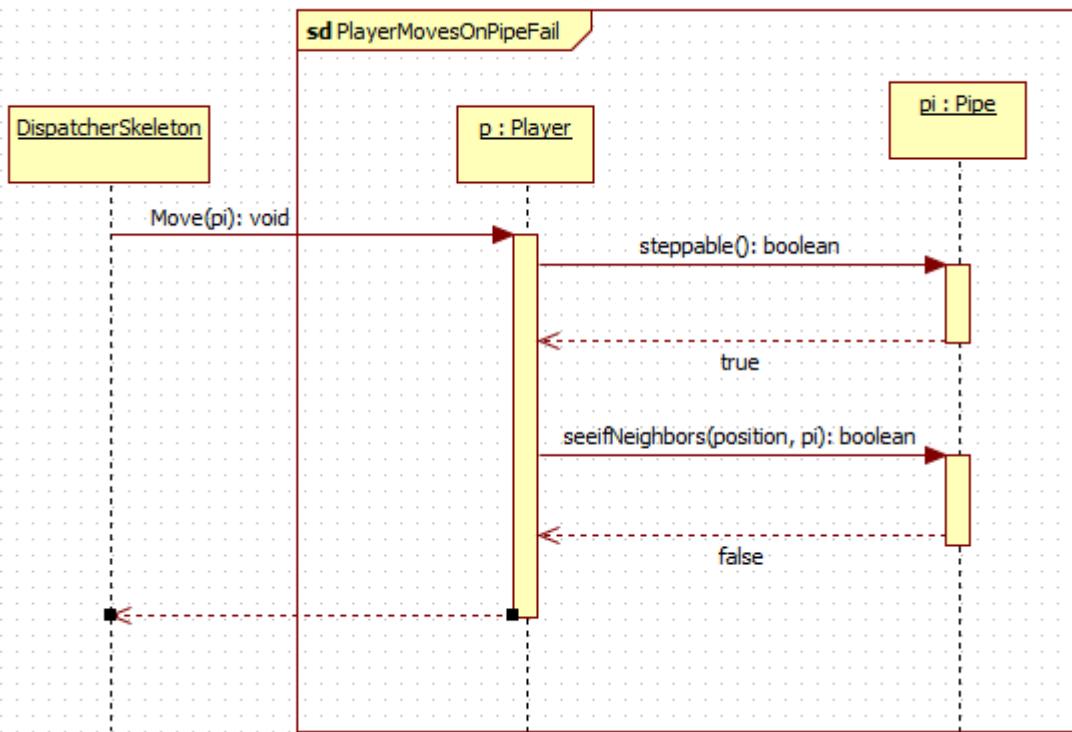
### 5.3.8 Player Move

#### 5.3.8.1 Player Moves To Pipe Successfully (Use-Caseben Player - Move - MoveToPipe)



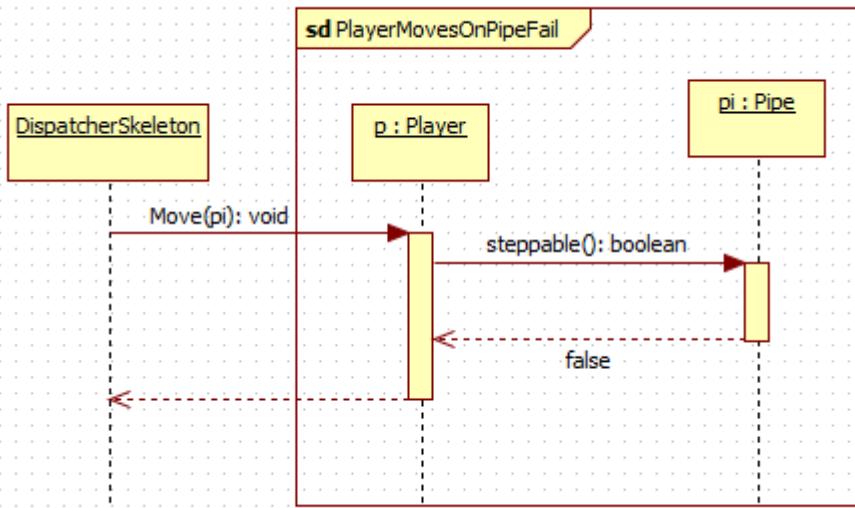
**Komment:** Először a DispatcherSkeleton Move(pi) kezdeni, utána a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben minden igaz, akkor Player sikerül lépni, ezzel új setPosition(pi) hívja.

### 5.3.8.2 Player Moves To Pipe Fail (Use-Caseben Player - Move - MoveToPipe)



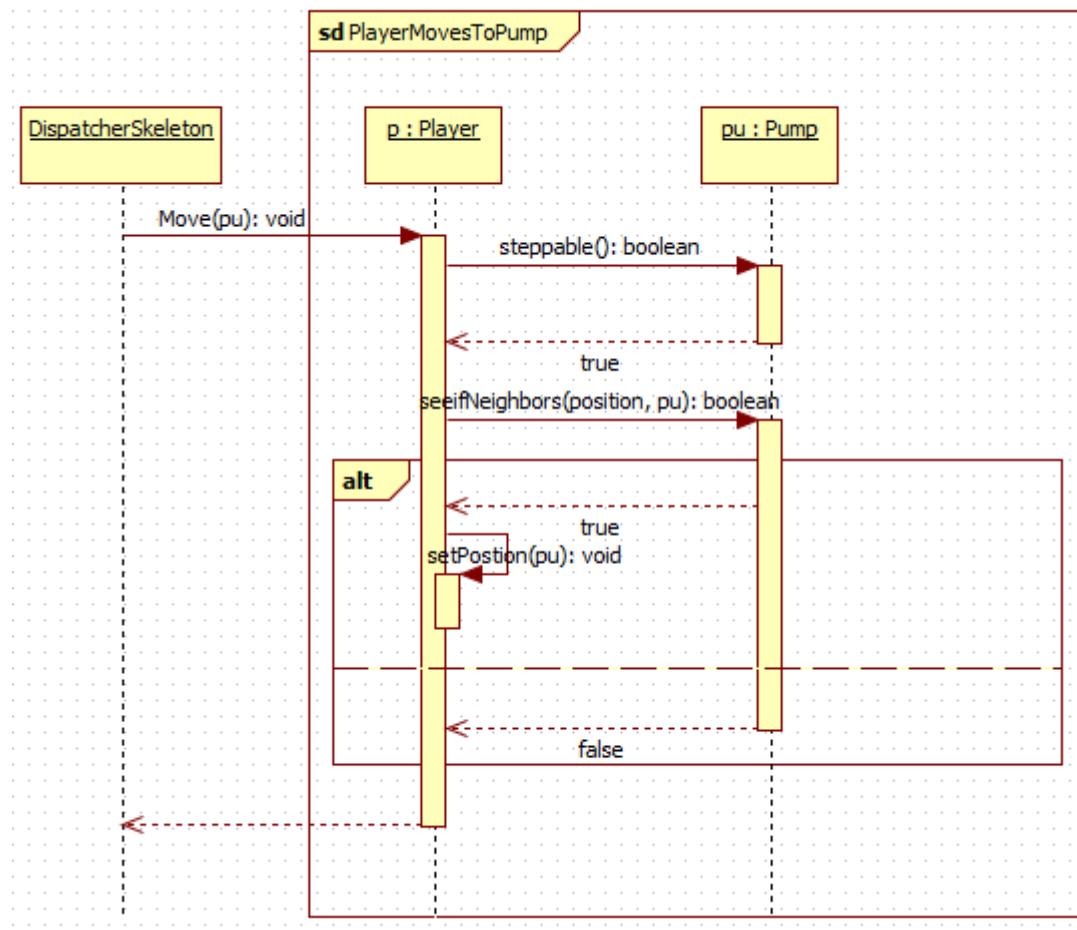
**Komment:** Először a DispatcherSkeleton `Move(pi)` kezdeni, utána a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben `steppable` igaz visszaadni, de `Pipe` nem szomszédja, ezért Player nem tud oda menni.

### 5.3.8.3 Player Moves To Pipe Fail (Use-Caseben Player - Move - MoveToPipe)



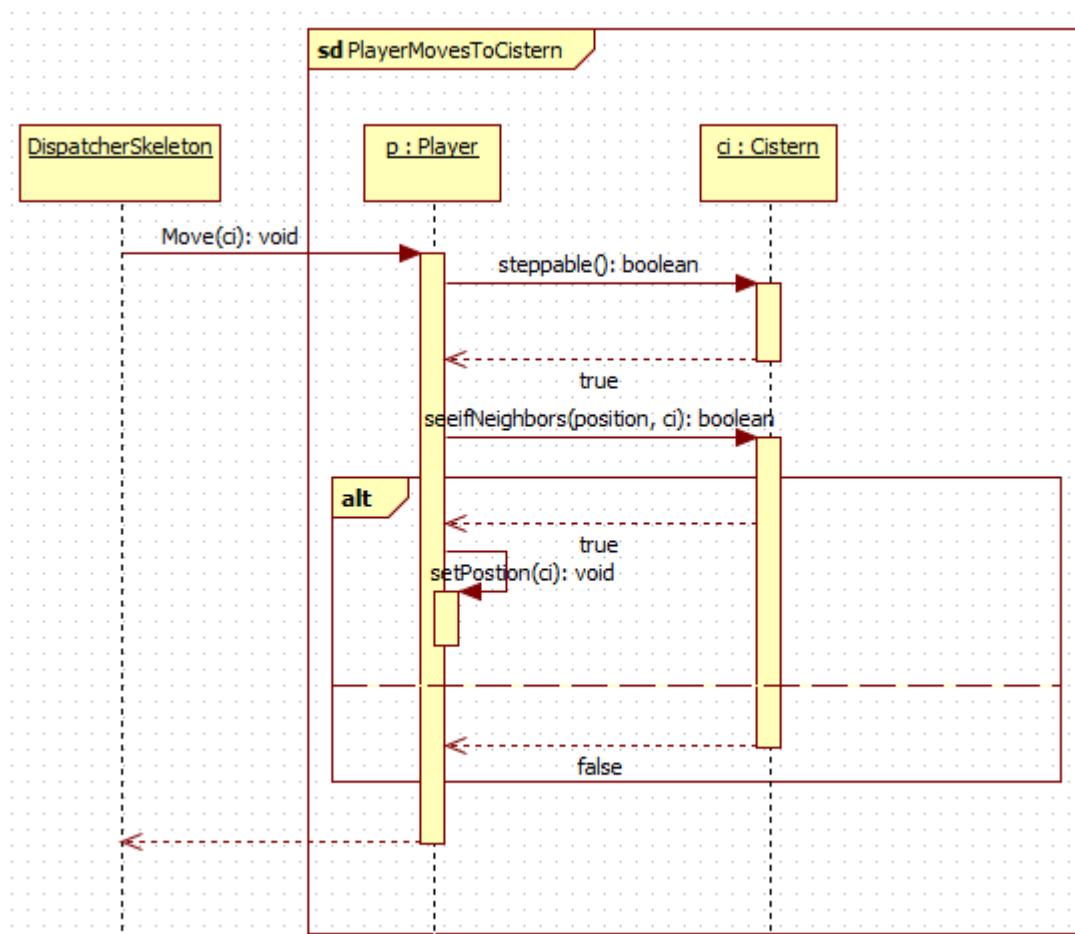
**Komment:** Először a DispatcherSkeleton `Move(pi)` kezdeni, utána a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja . Ebben az esetben `steppable` hamis visszaadni, ezért Player nem tud menni erre az elemre.

### 5.3.8.4 Player Moves To Pump (Use-Caseben Player - Move - MoveToPump)



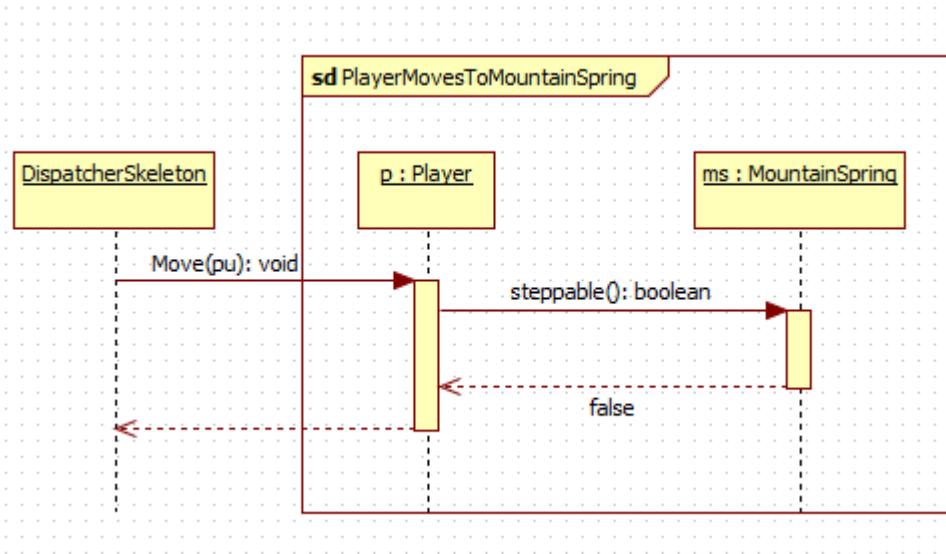
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja, majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben alt kezde, ha Pump szomszédja, akkor Player lehet oda menni és `setPosition` csak berakni új pozíciót, ha nem, akkor nem tud lépni, mert ez az aktív eleme, nem szomszédja

### 5.3.8.5 Player Moves To Cistern (Use-Caseben Player - Move - MoveToCistern)



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben alt kezde, ha Cistern szomszédja, akkor Player lehet oda menni és setPosition csak berakni új pozíciót , ha nem, akkor nem tud lépni, mert ez az aktív eleme, nem szomszédja

### 5.3.8.6 Player Moves To Mountain Spring (Use-Caseben Player - Move)

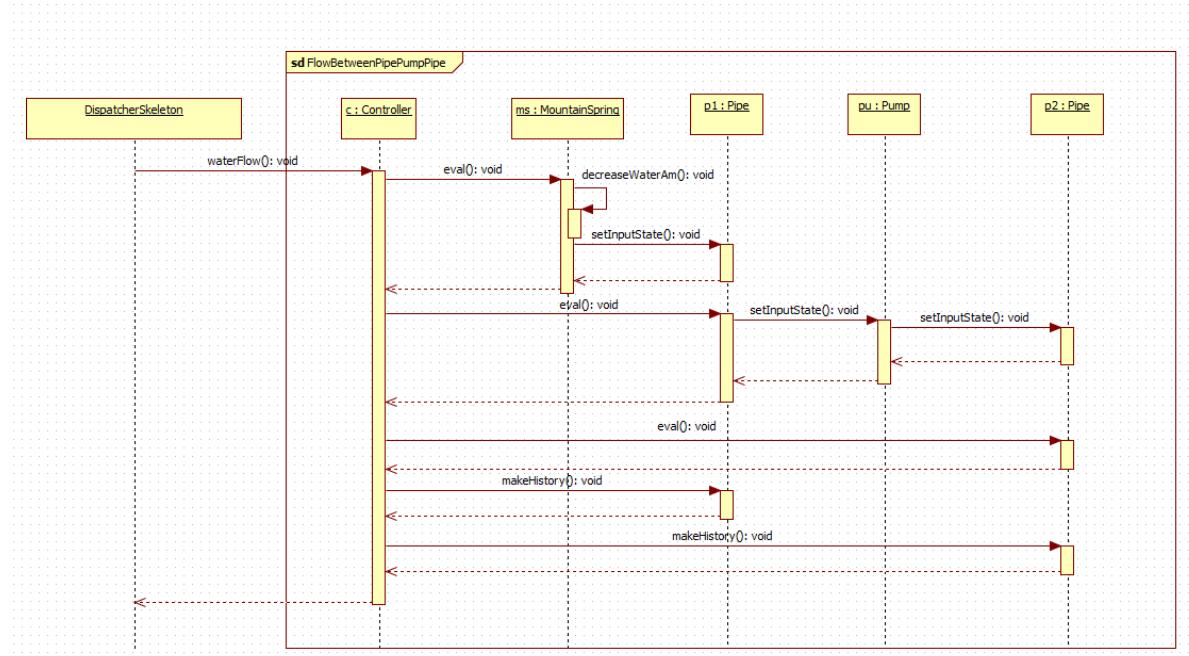


**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben steppable hamis (Mert ez a MountainSpring visszaadni, ezért Player nem tud menni erre az elemre

## Második fő objektum: Controller

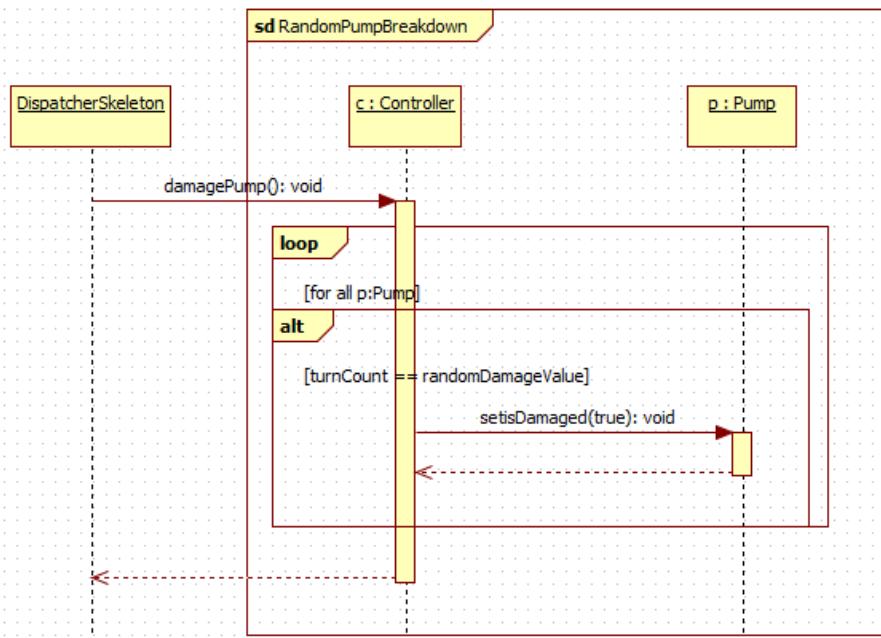
**Komment:** A Controller ez a fontos objektum része a játéknak és a Use-Case diagramnak. A Controller a játékmenet irányításáért, a Players akcióinak kezeléséért és a játék állapotának frissítéséért. Controller felelős Use-Case diagramában a Water Flowage a csőrendszerben és Pump breakdown folyamatok

### 5.3.9 Flow between Pipe-Pump-Pipe



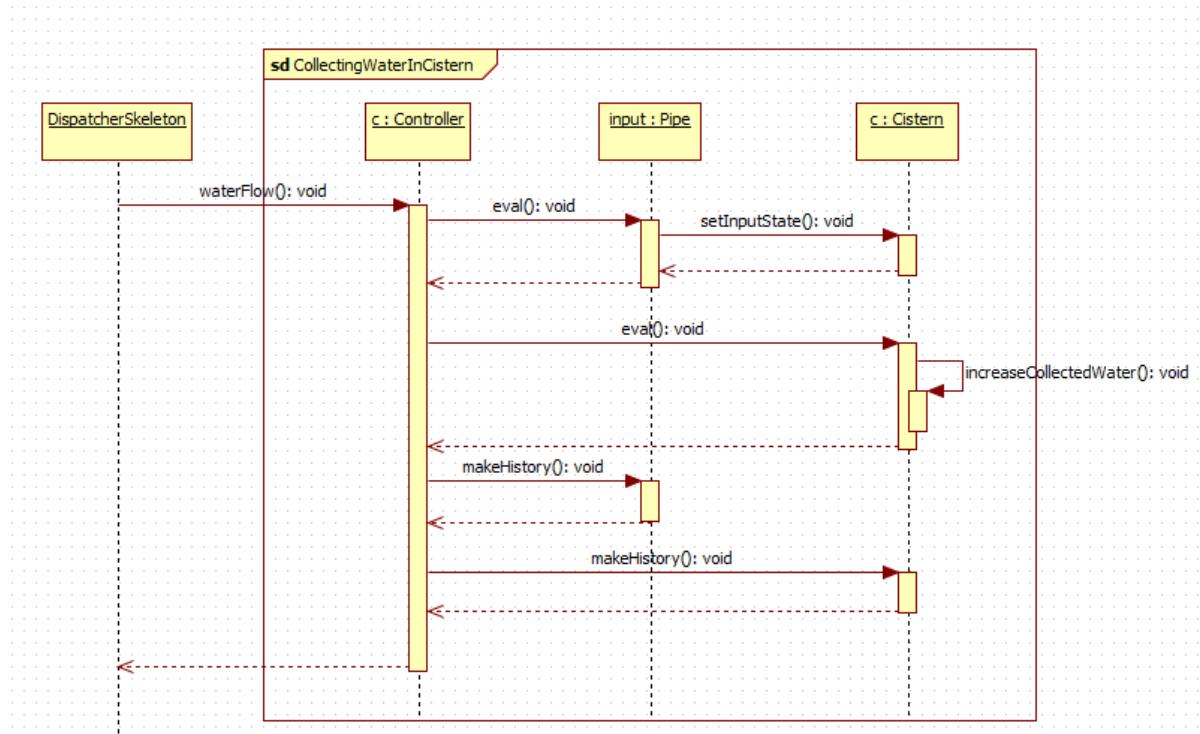
**Komment:** Ennél a szekvenica diagramnál nem a víz pályán való teljes mozgását hanem egy kis részének logikáját ábrázoltam. A DS mehívja a `waterFlow()` függvényt amely elindítja a víz mozgásának folyamatát. Ezt követően meghívódik a (ms) Mountain Spring `eval()` függvénye amely forrásból való víz csökkenését értékeli ki (igazából leginkább csak a `decreaseWaterAm()` meghívásáért felelős ennél az osztálynál). A forrás `eval()` függvénye meghívja a Mountain Spring output attribútumában lévő (p1) Pipe `setInputState()` függvényét (röviden ez a függvény a Pipe objektum (`InputState: boolean [2]`) attribútum második értékét változtatja meg amely a víz folyásának időbeliségeért felelős). Ezeket a kiértékelésket követően a függvények visszatérnek egészen a `waterFlow()` függvényig ahonnan a következő aktív elem kiértékelésére kerül sor (p1) és ennek is meghívjuk az `eval()` függvényét. Ez az `eval()` függvény felelős a folyás "logikájáért". Ezen a függvényen belül meghívjuk a következő aktív elem ((pu) pumpa) `setInputState()` függvényét amely jelen esetben a egyszerűen csak a következő cső `setInputState()`-jét hívja meg (itt ez azért lényeges mivel ezzel lehet kikerülni azt, hogy a pumpa 1 időegységet elvegyen a folyásból, így csak "átfolyik" a pumpán a víz 0 időegység alatt. Az összes kiértékelést követően a meghívódik a cső elemekre a `makeHistory()` függvény amely majd a következő körben lévő állapotok miatt lesz egy fontos dolog.

### 5.3.10 Random pump-breaking (Use-Caseben Controller - Pump Breakdown)



**Komment:** Itt a véletlenszerű elromlását írjuk le a pumpának. Végigmegyünk a Map osztály konténer listájában lévő összes Pump elemen és amely elemek a randomDamageValue-ja megegyezik az eltelt körök számával akkor az adott pumpa elromlik

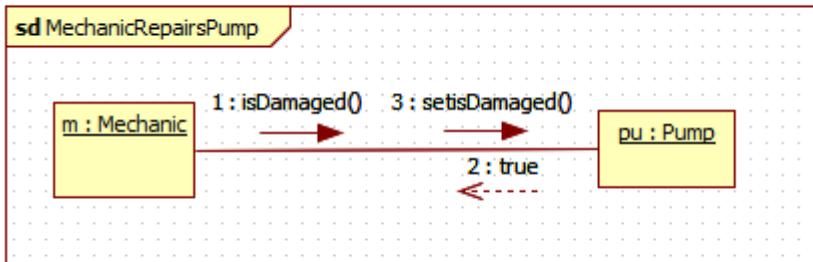
### 5.3.11 Collecting water in Cistern ( Controller - Water Flowage)



**Komment:** Ebben a szekvencia diagramban a Ciszternába beérkező víz akcióját követjük végig. Maga a folyamat teljesen megegyezik az 5.3.9-es szekvencia diagrammal egy eltéréssel, hogy amikor a következő aktív elem amire meghívjuk az eval() metódust a ciszterna, akkor ez a kiértékelés a ciszternába bemenő víz mennyiségrért felelős csak..

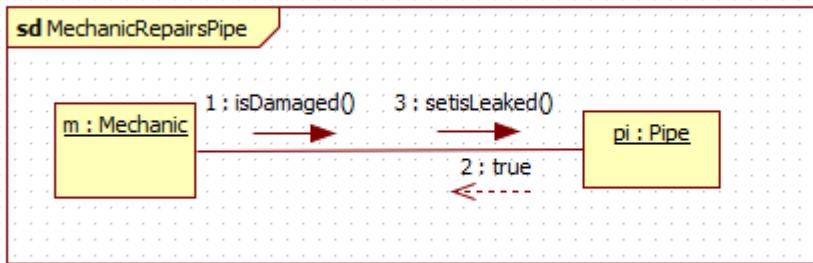
## 5.4 Kommunikációs diagramok

### 5.4.1 Mechanic Repairs Pump



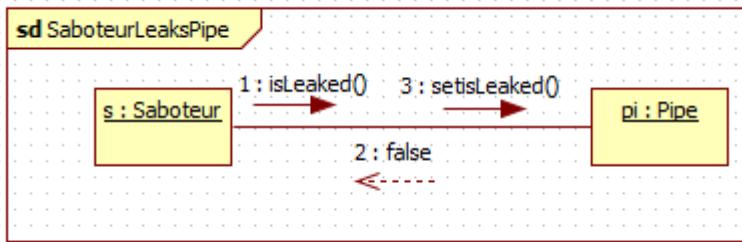
**Komment:** A Mechanic ellenőrzi, hogy a Pump sérült-e (igaz), utána hívja a setIsDamaged(false) parancsot. Az utolsó függvénye az isDamaged == false legyen, akkor Pump megjavítják

### 5.4.2 Mechanic Repairs Pipe



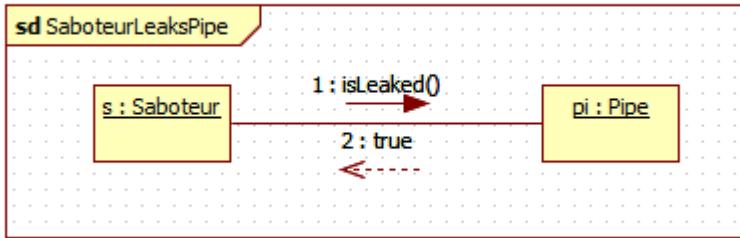
**Komment:** A Mechanic nézd meg, hogy a Pipe szivárog-e(igaz), utána hívja a setIsLeaked(false) parancsot. Az utolsó függvénye az isLeaked == false legyen, akkor Pipe megjavítják

### 5.4.3 Saboteur leaks pipe I



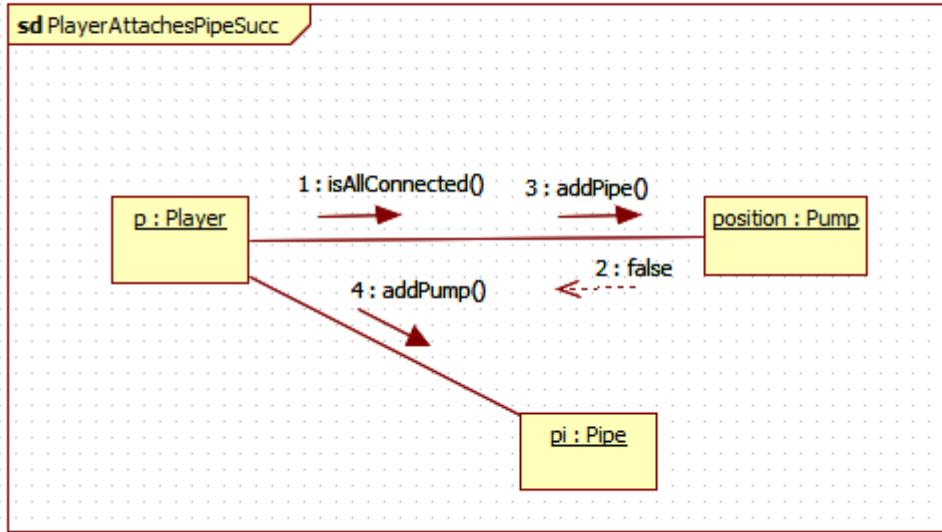
**Komment:** Saboteur ellenőrzi Pipe attribútum isLeaked, ha nem, akkor Saboteur hívás setIsLeaked() és nem a Pipe szivárgás

#### 5.4.3.1 Saboteur leaks pipe II



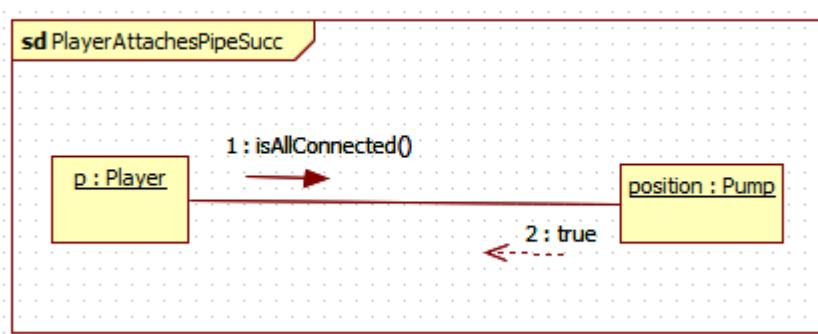
**Komment:** Saboteur ellenőrzi Pipe attribútum isLeaked, ha igen, Saboteur (happy) se csinál

#### 5.4.4.1 Player Attaches Pipe Successful



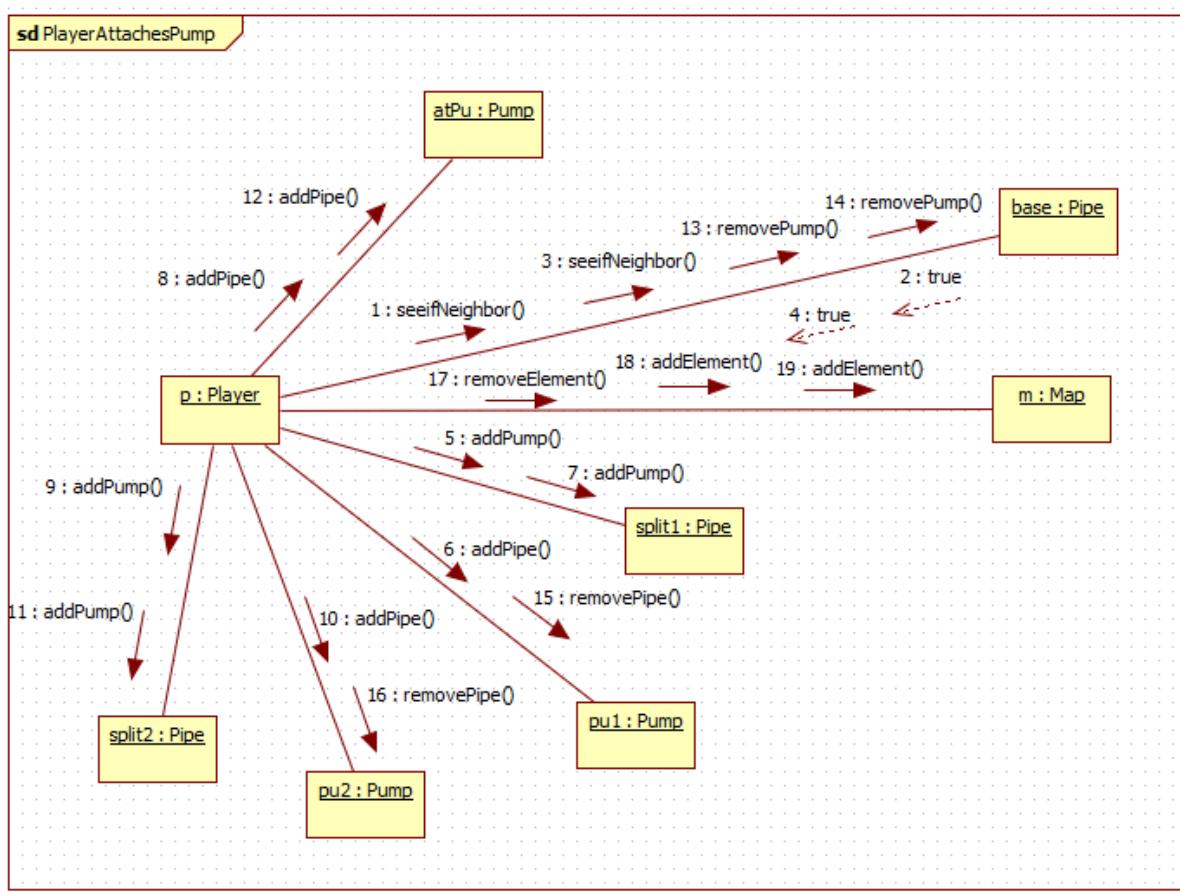
**Komment:** Először a Player az isAllConnected hívásával ellenőrzi, hogy minden Pipe csatlakoztatva van-e. Ha nem, akkor a Player az addPipe hívásával sikeresen csatlakoztatja az új Pipe a rendszerhez, és a szivattyúhoz csatlakoztatja.

#### 5.4.4.2 Player Attaches Pipe Fail



**Komment:** Először a Player az isAllConnected hívásával ellenőrzi, hogy minden Pipe csatlakoztatva van-e. Ha igen, akkor a Player nem tud csinálni attachPipe

### 5.4.5 Player Attaches Pump



**Komment:** Az első fázisa az illesztésnek a (base) Pipe felosztása illetve a környező pumpák lekérdezése (ezt egy getNeighbor() függvény hívásánál kapjuk meg azonban az egyszerűség kedvéért most itt csak lekérdeztünk kettő adott pumpához hogy szomszédja-e (base) Pipe-nak). Mivel az illesztés úgy megy végbe, hogy a játékos félbe vág egy csövet ezért itt létre kell hoznunk két másik cső objektumot jelen esetben (split1) Pipe és (split2) Pipe objektumokat. Ezek létrehozása után egyenként meg kell tennünk az illesztéseket (példaként kövessük végig (split1) Pipe objektumunkat):

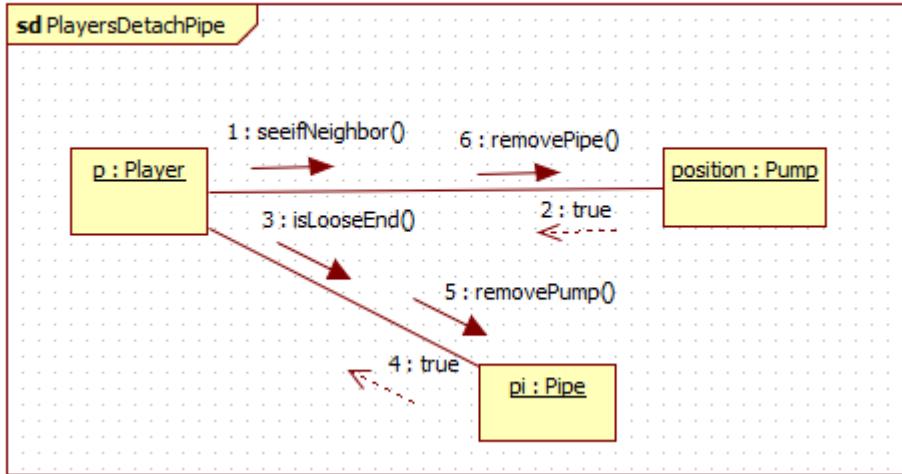
-A (split1) Pipe objektum pumpa listájába hozzáadjuk az újonnan beillesztett pumpát (atPu) illetve az eredeti csőhöz (base) tartozó valamely szomszédját (jelen esetben (pu1) csövet)

-Ezután értelemszerűen hozzáadjuk (pu1) és (atPu) pumpák cső listájához (split1) csövet

Teljesen hasonlóan ugyanezt megcsináljuk (split2) csőnél is (csak a (base) cső másik szomszédjával). Legvégül kiszedjük (base) Pipe-ot a szomszédos pumpák csőlistájából majd

kiszedjük (removeElement(base) a Map osztályunk konténer listájából és hozzáadjuk a kettő újonnan született csövet a listához (addElement(split1), addElement(split2)).

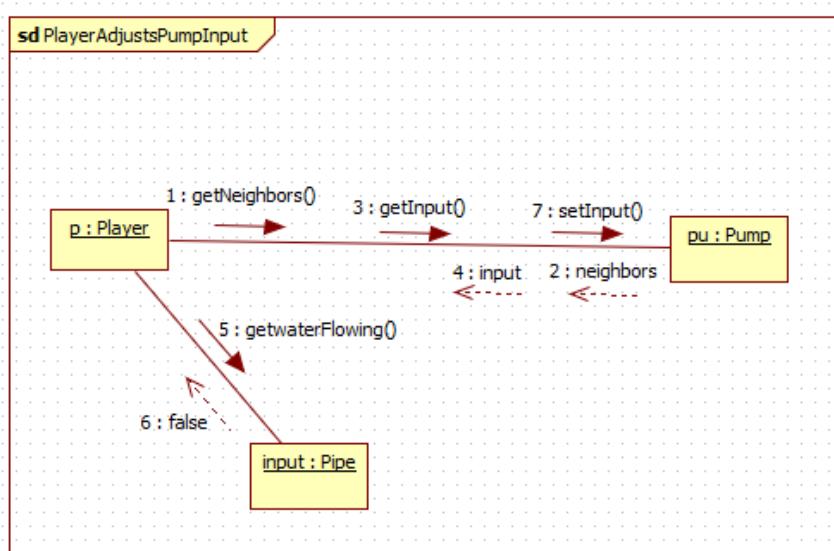
#### 5.4.6 Player Detach Pipe



**Komment:** Először is a Player látja a szomszédokat a seeIfNeighbours (true) segítségével, ha a Pump vannak szomszédai, a Player eltávolítja a végét az isLooseEnd() segítségével és

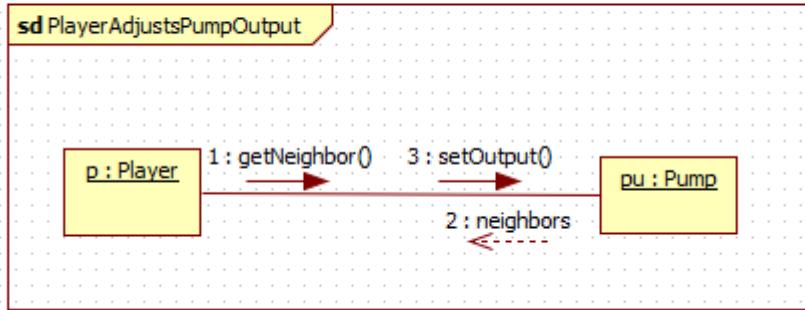
természetesen eltávolítjuk a Pump is a removePump() segítségével, csak ezután a Player készen áll a Pipe eltávolítására a rendszerből.

#### 5.4.7.1 Player Adjusts Pump Input



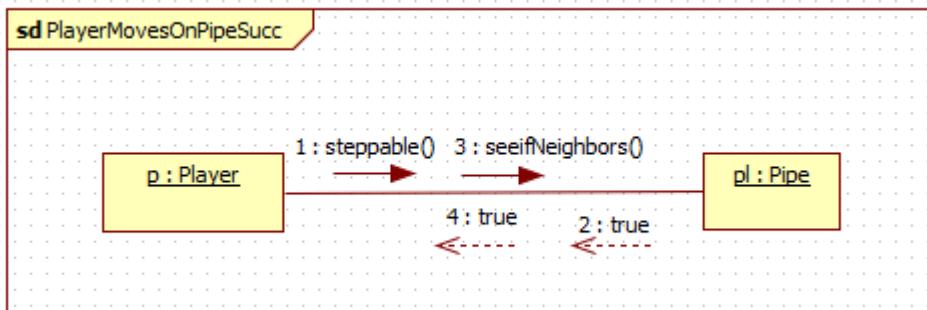
**Komment:** A Player átállítja, hogy az adott pumpánál mely csőből mely csőbe folyjon a víz. A Player megkérdezi a Pump szomszédait getNeighbours() hívja , kéri a bemeneteket getInput() hívja , setInput() hívja meghatározza, hogy honnan fog folyni a víz.

### 5.4.7.2 Player Adjusts Pump Output



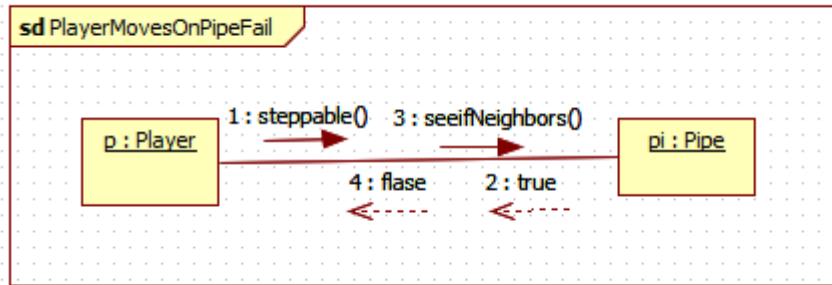
**Komment:** A Player átállítja, hogy az adott pumpánál mely csőből mely csőbe folyjon a víz. Player megkérdezi a Pump szomszédait `getNeighbours()` hívja, `setOutput()` hívja meghatározza, hogy hova fog folyni a víz.

### 5.4.8.1 Player Moves On Pipe Successful



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben minden igaz, akkor Player sikerül lépni .

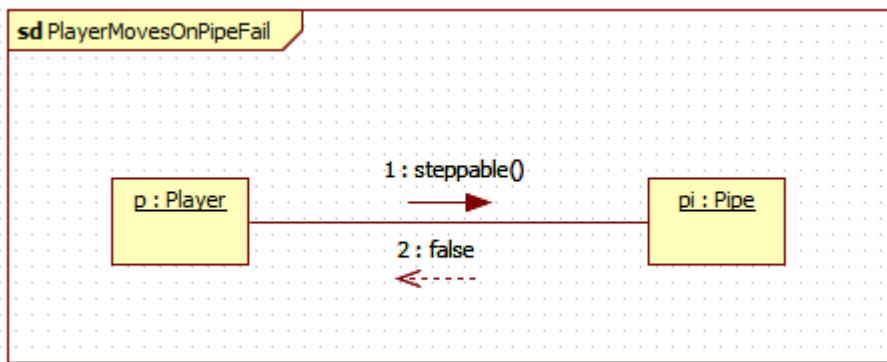
### 5.4.8.2 Player Moves On Pipe Fail I



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja.

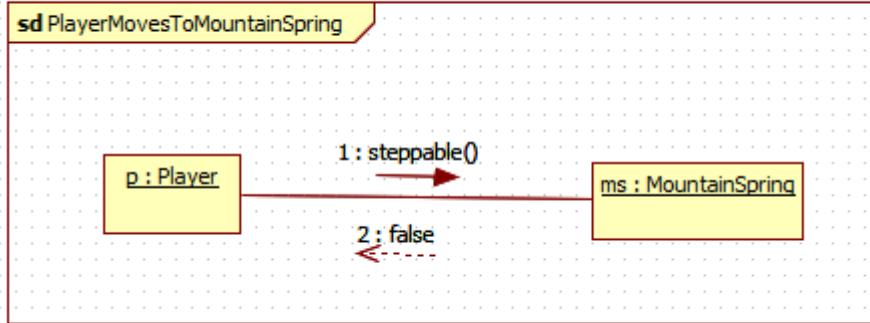
Ebben az esetben steppable igaz visszaadni, de Pipe nem szomszédja, ezért Player nem tud oda menni.

### 5.4.8.3 Player Moves on Pipe Fail II



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja . Ebben az esetben steppable hamis visszaadni, ezért Player nem tud menni erre az elemre

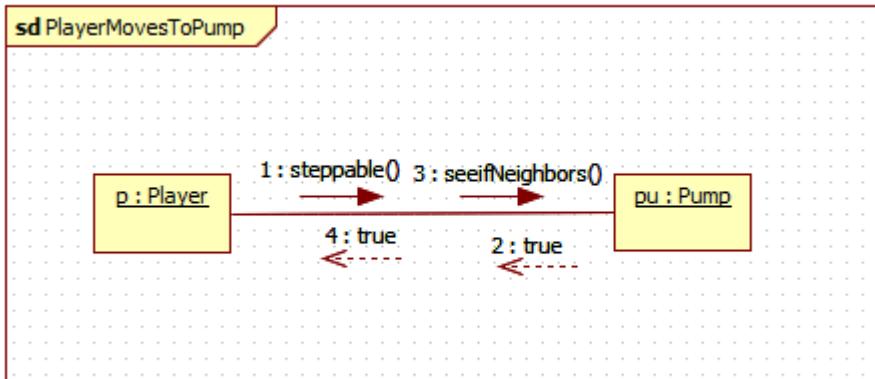
#### 5.4.8.4 Player Move To Mountain Spring



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja.

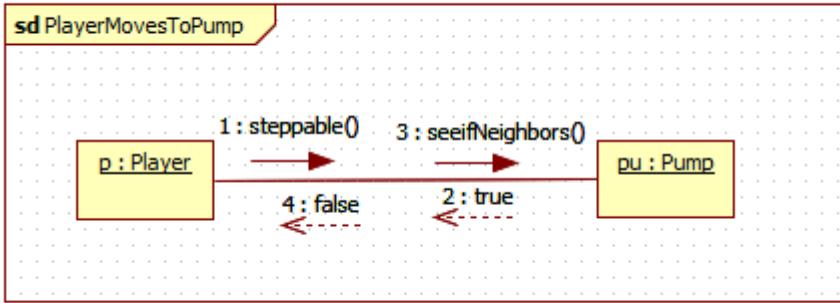
Ebben az esetben steppable hamis (Mert ez a MountainSpring) visszaadni, ezért Player nem tud menni erre az elemre

#### 5.4.8.5 Player Moves To Pump



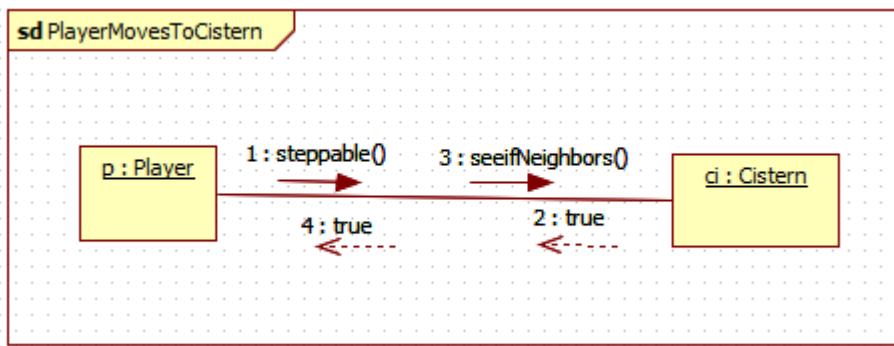
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben minden igaz, akkor Player sikerül lépni (mert ez a Pump eleme)

#### 5.4.8.6 Player Moves To Pump



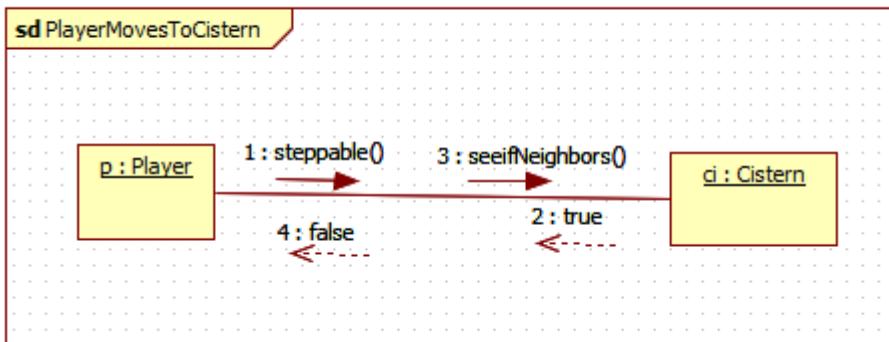
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben steppable igaz, de ez nem szomszédja, ezért a Player nem sikerül lépni.

#### 5.4.8.7 Player Moves To Cistern



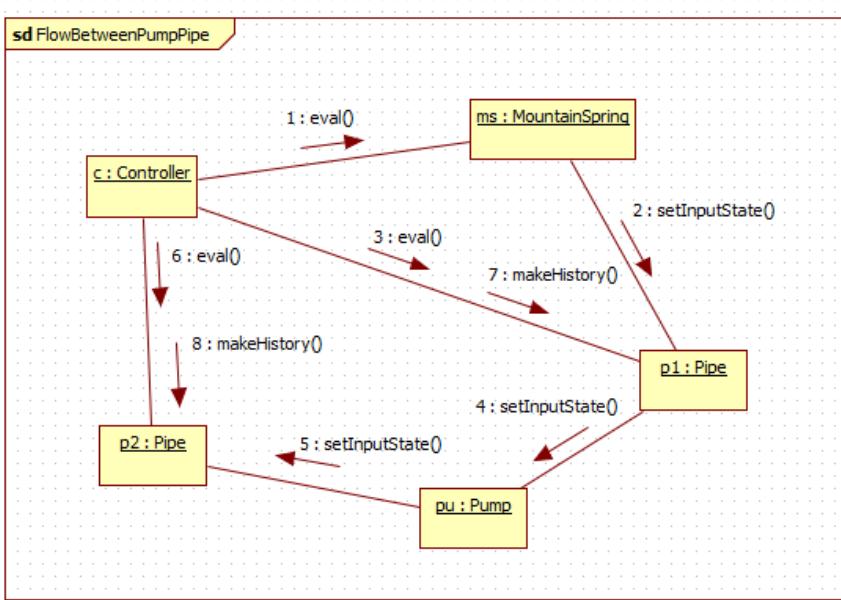
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben minden igaz, Player sikerül lépni.( ez a Cistern eleme)

### 5.4.8.8 Player Moves To Cistern



**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre steppable() hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e seeifNeighbours() hívja. Ebben az esetben steppable igaz, de ez nem szomszédja, ezért a Player nem sikerül lépni

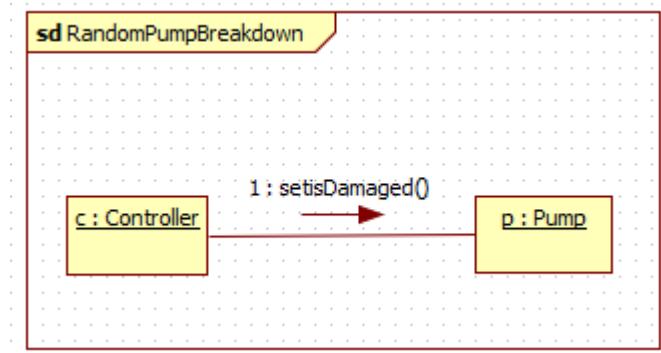
### 5.4.9 Flow Between Pump Pipe



**Komment:** Mountain Spring eval() függvénye amely forrásból való víz csökkenését értékeli ki (igazából leginkább csak a decreaseWaterAm() meghívásáért felelős ennél az osztálynál). A forrás eval() függvénye meghívja a Mountain Spring output attribútumában lévő (p1) Pipe setInputState() függvényét (röviden ez a függvény a Pipe objektum (InputState: boolean [2]) attribútum második értékét változtatja meg amely a víz folyásának időbeliségéért felelős). Ezeket a kiértékelésket követően a függvények visszatérnek egészen a waterFlow() függvényig ahonnan a következő aktív elem kiértékelésére kerül sor (p1) és ennek is meghívjuk az eval() függvényét. Ez az eval() függvény felelős a folyás "logikájáért". Ezen a függvényen belül meghívjuk a következő aktív elem ((pu) pumpa) setInputState() függvényét amely jelen esetben a egyszerűen csak a következő cső setInputState()-jét hívja meg (itt ez

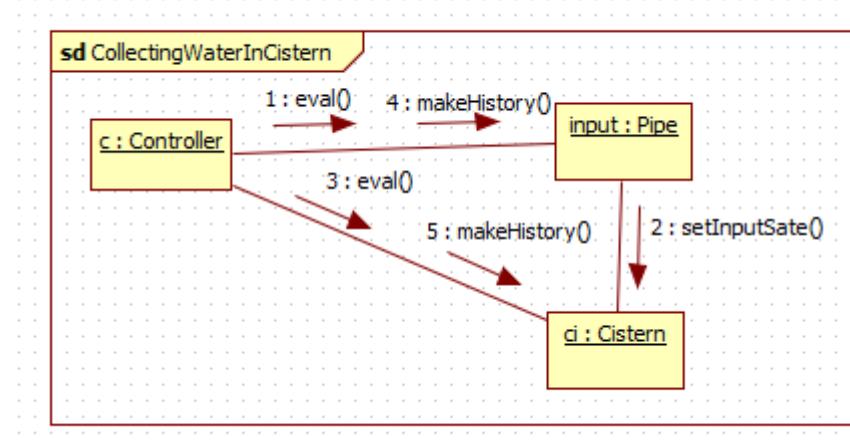
azért lényeges mivel ezzel lehet kikerülni azt, hogy a pumpa 1 időegységet elvegyen a folyásból, így csak “átfolyik” a pumpán a víz 0 időegység alatt. Az összes kiértékelést követően a meghívódik a cső elemekre a makeHistory() függvény amely majd a következő körben lévő állapotok miatt lesz egy fontos dolog.

#### 5.4.10 Random Pump Breakdown



**Komment:** Controller random Pump Break random időben egy függvényhívás Pumpához setisDamaged()

#### 5.4.11 Collecting Water in Cistern



**Komment:** Maga a folyamat teljesen megegyezik az 5.4.9-es diagrammal egy eltéréssel, hogy amikor a következő aktív elem amire meghívjuk az eval() metódust a ciszterna, akkor ez a kiértékelés a ciszternába bemenő víz mennyiségeknek növeléséért felelős csak..

**Napló**

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résznevők</b>	<b>Leírás</b>
2023.04.01. 12:30	0,5 óra	Molnár Márton Réti Ádám Czifra Barnabás Tisza Miklós Kopach Artem	Értekezlet: Feladatok kiosztása, megbeszélése
2023.04.01 13:00	1.5 óra	Réti Ádám Tisza Miklós	Értekezlet: 5.1 Use-Case diagram módosítása, leírások újragondolása
2023.04.01 13:00	1.5 óra	Molnár Márton Czifra Barnabás	Értekezlet: 5.2 A szkeleton kezelői felületének terve, dialógusok
2023.04.01 13:00	1.5 óra	Kopach Artem	Tevékenység: 5.3 A játékot fő osztályokra osztotta, kommentek, beraktam és csinálok új diagramokat
2023.04.02 13:00	5 óra	Tisza Miklós	5.3 Szekvencia diagramok készítése, leírása
2023.04.02 13:00	4 óra	Réti Ádám	5.4 Szekvencia diagramok alapján kommunikációs diagramok elkészítése
2023.04.02 13:00	3.5 óra	Kopach Artem	Tevékenység: 0. Új cím sablont, nyomtatás és beadás 5.3 írt megjegyzéseket a játékos mozgásához hibákat keresve 5.4 írta az összes megjegyzést a diagramokra, hibákat keresve

# Szkeleton elkészítése

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

### Réti Ádám - csapatvezető

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)      ket.vill.adam@gmail.com  
  
(E1LX0J)      tisza.miklos.16@gmail.com  
(NX9GA4)      barna.czifra@gmail.com  
(KLM600)      molnar.marton.hun@edu.bme.hu  
(JQBOLI)      kopach.artem@edu.bme.hu

2023.04.15

## 6. Szkeleton beadás

### 6.1 Fordítási és futtatási útmutató

#### 6.1.1 Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
Cistern.java	6.8 KB	2023.04.15 0:58	Cistern osztály
Container.java	5.5 KB	2023.04.15 0:58	Container osztály
MountainSpring.java	4.2 KB	2023.04.15 0:58	MountainSpring osztály
Pipe.java	8.6 KB	2023.04.15 0:58	Pipe osztály
Pump.java	8 KB	2023.04.15 0:58	Pump osztály
Controller.java	4.2 KB	2023.04.15 0:58	Controller osztály
MyException.java	138 B	2023.04.15 0:58	MyException osztály
Map.java	2 KB	2023.04.15 0:58	Map osztály
Type.java	362 B	2023.04.15 0:58	Type enumeráció
Mechanic.java	714 B	2023.04.15 0:58	Mechanic osztály
Player.java	4.2 KB	2023.04.15 0:58	Player osztály
Saboteur.java	622 B	2023.04.15 0:58	Saboteur osztály
DispatcherSkeleton.java	28.3 KB	2023.04.15 0:58	DispatcherSkeleton osztály. A skeleton program vezérlését és a szekvenciák futtatását megvalósító osztályt tartalmazza.
Main.java	3.2 KB	2023.04.17 03:48	A DispatcherSkeleton függvényeit hívja meg a main-ben.

## 6.1.2 Tesztek leírás

Itt a következő részben a program sikeres futtatása után, a menüben megjelenő tesztelendő operációk elvárt működéseit és kimeneteit fogjuk részletezni. Itt a gombok lenyomása alatt, a konzolban írt választott operáció szám utáni enter lenyomását értjük.

Példa elvárt inputra:

```
[15]. Collecting water in cistern  
[0]. Kilépés  
8  
AdjustPump has started
```

### 6.1.2.1 Mechanic repairs pump

A menüben az 1-es gomb megnyomásával egy szerelő pumpa megjavítását szimuláljuk.

Idetartozó szekvencia diagram: **MechanicRepairsPump**

A teszt a MechanicRepairsPump() metódus meghívásával kezdődik, amely inicializálja a Map objektumot az initializeTable() metódus segítségével. Úgy tűnik, hogy a Map objektum tárolja a tartály adatait a csőrendszerben. Ezután egy új Mechanic objektum jön létre a Map objektum első tárolójával, mint kezdeti pozícióval, amely a csőrendszer karbantartásáért és javításáért felelős vízvezeték-szerelőt képviseli. A módszer ezután kiad egy üzenetet, hogy a szerelő megkezdte a szivattyú javítását, és megjeleníti a szivattyú isDamaged tulajdonságát a javítás megkezdése előtt. Ezután a Mechanic objektum RepairPump() metódusa meghívásra kerül, amely megpróbálja megjavítani a szivattyút. Ha a javítás sikeres, a rendszer sikeres üzenetet ad ki, ellenkező esetben hibaüzenet jelenik meg. A teszt ezen része a "Mechanic repairs pump" forgatókönyv szimulációját valósítja meg egy szekvencia diagram segítségével.

Elvárt kimenet:

```
Mechanic repairs pump has started  
Pump isDamaged before repair: true  
RepairPump is called  
RepairPump has returned  
Pump repair was successful :)  
Pump isDamaged after repair: false  
Mechanic repairs pump has finished
```

### 6.1.2.2 Mechanic repairs pipe

A menüben az **2-es gomb** megnyomásával egy szerelő cső megjavítását szimuláljuk.

Idetartozó szekvencia diagram: **MechanicRepairsPipe**

Ez a teszt azt szimulálja, hogy egy szerelő javít egy csövet a csővezetékrendszerben. Az "initializeTable()" metódus segítségével inicializál egy Map objektumot, és létrehoz egy Mechanic objektumot egy Pipe objektummal a javítandó cső ábrázolására. A módszer ezután a Pipe objektum isLeaked tulajdonságát true-ra állítja a szivárgás szimulálására, meghívja a Mechanic objektum "RepairPipe()" metódusát a javítás szimulálására, és az eredmény alapján sikeres vagy hibaüzenetet ír ki. Végül a módszer kiírja az isLeaked tulajdonság állapotát a javítás után, és egy üzenettel zárul, amely jelzi, hogy a szerelő befejezte a cső javítását. A tesztnak ezt a részét a "Mechanic Repairs Pipe szimuláció" nevű szekvencia diagram ábrázolja.

Elvárt kimenet:

```
Mechanic repairs pipe has started
Pipe isLeaked before repair: true
RepairPipe is called
RepairPipe has returned
Pipe repair was successful :)
Pipe isLeaked after repair: false
Mechanic repairs pump has finished
```

### 6.1.2.3 Saboteur leaks pipe

A menüben az **3-as gomb** megnyomásával egy szabotőr cső rongálását szimuláljuk.

Idetartozó szekvencia diagram: **SaboteurLeaksPipe**

A teszt a SaboteurLeaksPipe() metódus meghívásával kezdődik, amely egy szabotőr által okozott csővezeték-szivárgást szimulál. A módszer inicializálja a térképet az initializeTable() metódus segítségével, létrehoz egy Saboteur objektumot egy konténerrel paraméterként, és a konténer csővezetékekének szivárgott tulajdonságát false-ra állítja. Ezután a Saboteur objektum LeakPipe() metódusa meghívásra kerül a csővezeték szivárgásának szimulálására. A teszt ellenőrzi a tartály csövének szivárgott tulajdonságát, hogy megállapítsa, sikeres volt-e a szabotázs, és végül kiadja a szivárgott tulajdonságot és a szimuláció végét jelző üzenetet. A tesztnak ez a része egy szekvencia diagramot valósít meg a Saboteur leaks Pipe szimulációhoz.

Elvárt kimenet:

```
Saboteur leaks pipe has started
Pipe isleaked before sabotage: false
LeakPipe is called
LeakPipe has returned
Pipe sabotage was succesful :)
Pipe isleaked after sabotage: true
Saboteur leaks pipe has finished
```

#### 6.1.2.4 Player attach pipe

A menüben az **4-es gomb** megnyomásával egy játékos csőillesztését szimuláljuk. Idetartozó szekvencia diagram: **PlayerAttachPipeFail/Succ**

Ez a kód azt szimulálja, hogy a játékos egy csövet csatlakoztat a csőrendszerhez. Létrehoz egy új Player objektumot a 7th pipe konténerben, hozzáad egy Pipe objektumot az átvihető csövekhez, és meghívja az "attachPipe()" metódust. Ha a csatolás sikeres volt, akkor egy üzenet jelenik meg, amely ezt közli, ha nem, akkor egy hibaüzenet jelenik meg. A tesztnak ez a része egy szekvencia diagramot valósít meg a Player attach Pipe szimulációhoz.

Elvárt kimenet:

```
Player attach pipe successful has started
attachPipe is called
attachPipe has returned
Player attach pipe successful was successful :)
Player attach pipe successful has finished
```

#### 6.1.2.5 Player attaches pump

A menüben az **5-ös gomb** megnyomásával egy játékos pumpaillesztését szimuláljuk.

Idetartozó szekvencia diagram: **PlayerAttachesPump**

A teszt a PlayerAttachesPipe() metódus meghívásával kezdődik, amely inicializál egy Map objektumot, létrehoz egy Player objektumot a térkép harmadik tárolójával, mint kezdőpozíció, és megpróbál egy új Pump objektumot csatolni a Player pozíciójához. A kód kiírja a metódus kezdetét és végét, valamint a szivattyú csatolás sikerét vagy sikertelenségét jelző üzeneteket is. A módszer egy MyException nevű egyéni kivételt dob. A tesztnak ez a része egy szekvencia diagramot valósít meg a Player attaches pump szimulációhoz.

Elvárt kimenet:

```
Player attaches pump has started
attachPump is called
Player attached pump successfully :)
Player attaches pump has finished
```

### 6.1.2.6 Player detach pipe

A menüben az **6-s gomb** megnyomásával egy játékos cső leillesztését szimuláljuk. Idetartozó szekvencia diagram: **PlayerDetachPipe**

A teszt a PlayerAttachesPump() metódus meghívásával kezdődik, amely inicializál egy Map objektumot, létrehoz egy Player objektumot a térkép harmadik konténerével, mint kezdőpozícióval, és megpróbál egy új Pump objektumot csatolni a Player pozíójához. A kód kiírja a metódus kezdetét és végét, valamint a szivattyú csatolás sikerét vagy sikertelenségét jelző üzeneteket is. A módszer egy MyException nevű egyéni kivételt dob.

Elvárt kimenet:

```
Player detach pipe has started
detachPipe is called
detachPipe has returned
Player has detached pipe successfully :)
Player detach pipe has finished
```

### 6.1.2.7 Player adjust pump Input

A menüben az **7-s gomb** megnyomásával egy játékos pumpa inputjának átállítását szimuláljuk. Idetartozó szekvencia diagram: **PlayerAdjustPumpInput**

A teszt a PlayerAdjustPumpInput() metódus meghívásával kezdődik, amelyben a játékos karakter beállítja a szivattyú bemeneti nyílását. A metódus inicializálja a játéktérképet és a játékos pozíóját, kiírja az eredeti szivattyú beömlőnyílást, majd meghívja az adjustPump metódust, hogy a beömlőnyílást a megadottá változtassa. A kód ezután ellenőrzi, hogy a beállítás sikeres volt-e, és ennek megfelelően kiír egy sikeres vagy sikertelen üzenetet.

Elvárt kimenet:

```
Player adjust pump Input has started
Original input of pump: container.Pipe@4b67cf4d
adjustPump is called
adjustPump has returned
Input adjustment successful :)
Input successfully got adjusted to: container.Pipe@7ea987ac
Player adjust pump Input has finished
```

### 6.1.2.8 Player adjust pump Output

A menüben az **8-s gomb** megnyomásával egy játékos pumpa outputjának átállítását szimuláljuk. Idetartozó szekvencia diagram: **PlayerAdjustPumpOutput**

A teszt a PlayerAdjustPumpOutput() metódus meghívásával kezdődik, amellyel a játékos beállíthatja a játékban a szivattyú kimenetét. Inicializálja a játéktérképet és a játékost, majd meghívja a játékos adjustPump() metódusát, hogy megkísérelje a beállítást. Siker esetén a módszer egy sikeres üzenetet ad ki az új teljesítményértékkal együtt, ha nem, akkor egy hibaüzenetet.

Elvárt kimenet:

```
AdjustPump has started
Original output of pump: container.Pipe@12a3a380
adjustPump is called
adjustPump has returned
adjustPump Output successful :)
Output successfully got adjusted to: container.Pipe@29453f44
AdjustPump has finished
```

### 6.1.2.9 Player moves to pipe successfully

Menüben a **9-es gomb** megnyomásával egy játékos csövön való sikeres mozgatását szimuláljuk. Idetartozó szekvencia diagram: **PlayerMovesOnPipeSucc**

Ez a függvény egy játékost mozgat egy csőhöz (Pipe) egy adott térképen. A függvény először inicializál egy térképet (Map) a "initializeTable()" metódussal, majd létrehoz egy játékost, aki a térképen található első tartályban (Container) van. Ezután kiír néhány üzenetet a konzolra a függvény állapotáról, majd megpróbálja mozgatni a játékost a csőhöz..

A függvény a játékost a szomszédos tartályok közül az elsőhöz mozgatja, majd ellenőrzi, hogy a játékos a cél csövön (Pipe) mozgott-e. Ha igen, akkor kiírja a sikeres mozgást az üzenetek között, különben kiírja a sikertelen mozgást. Végül a függvény kiírja az állapotot, hogy befejeződött.

Elvárt kimenet:

```
PlayerMovesOnPipeSuc has started
Pipe we want to move onto: container.Pipe@4b67cf4d
Move is called
Move has returned
Player moves to Pipe successful :)
Player moved successfully onto: container.Pipe@4b67cf4d
PlayerMovesOnPipeSuc has finished
```

### 6.1.2.10 Player moves to pipe fail

Menüben a **10-es gomb** megnyomásával egy játékos csövön való sikeres mozgatását szimuláljuk. Idetartozó szekvencia diagram: **PlayerMovesOnPipeFail**

Ez a függvény szintén egy játékos mozgatását végzi el egy csőre az adott játékban, de ezúttal az a célja, hogy a játékos mozgatása sikertelen legyen, mivel a kiválasztott cső már foglalt. A függvény először inicializál egy térképet, majd létrehoz egy játékost az első tartályban.

Ezután megkeresi a játékos jelenlegi tartályának összes szomszédját, és az első szomszédos csőre próbálja mozgatni a játékost. Azonban mielőtt a játékos mozgatása megtörténne, a függvény beállítja az adott csőt foglaltnak. Azután a játékos mozgatását megpróbálja végrehajtani, de mivel a cél cső már foglalt, a játékos mozgása nem lehet sikeres. A try-catch blokk segítségével a függvény kezeli a dobott MyException kivételt. Végül a függvény ellenőrzi, hogy a játékos sikerült-e mozgatni a cél csőre, és az eredményt a konzolra írja ki.

Elvárt kimenet:

```
PlayerMovesOnPipeFail has started
Pipe we want to move onto: container.Pipe@7ea987ac
Given pipe is currently occupied so Move to that pipe must fail
Move is called
exception.MyException: The Pipe is clearly not steppable
Move has returned
Player moves to Pipe failed :)
Player stayed on: container.Pump@12a3a380
PlayerMovesOnPipeSuc has finished
```

### 6.1.2.11 Player moves to pump

Menüben a **11-es gomb** megnyomásával egy játékos pumpához való sikeres mozgatását szimuláljuk. Idetartozó szekvencia diagram: **PlayerMovesToPump**

A függvény inicializál egy térképet, majd létrehoz egy játékost a térkép második tartályában. Ezután megkeresi a játékos jelenlegi tartályának összes szomszédját, és az első szomszédos pumpára próbálja mozgatni a játékost. Azután a függvény a játékos mozgatását végrehajtja, majd ellenőrzi, hogy a játékost sikerült-e a pumpára mozgatni. A függvény az eredményt a konzolra írja ki, majd befejezi a futását.

Elvárt kimenet:

```
Player moves to pump has started
Pump the player wants to move onto: container.Pump@29453f44
Move is called
Move has returned
Move to Pump successful :)
Player successfully moved onto: container.Pump@29453f44
Player moves to pump has finished
```

### 6.1.2.12 Player moves to cistern

Menüben a **12-es gomb** megnyomásával egy játékos ciszternához való sikeres mozgatását szimuláljuk. Idetartozó szekvencia diagram: **PlayerMovesToCistern**

Ez a függvény egy játékos mozgatását végzi el egy ciszterna tartályra az adott játékban. Először inicializál egy térképet, majd létrehoz egy játékosat a térkép kilencedik tartályában. Ezután a függvény kiírja a konzolra a ciszterna tartályt, ahova a játékos akar mozogni. Azután a függvény a játékos mozgatását végrehajtja a ciszterna tartályra, majd ellenőrzi, hogy sikerült-e a mozgatás. Ha igen, akkor a függvény a sikeres mozgatást jelzi a konzolon, és kiírja a játékos új pozícióját. Ha nem sikerült a mozgatás, akkor a függvény a sikertelenséget jelzi a konzolon.

Elvárt kimenet:

```
Player moves to cistern has started
The Cistern that the player wants to move onto: container.Cistern@5cad8086
Move is called
Move has returned
Move to Cistern successful :)
Player sucessfully moved onto: container.Cistern@5cad8086
Player moves to cistern has finished
```

### 6.1.2.13 Player moves to mountain spring

Menüben a **13-es gomb** megnyomásával egy játékos hegyi forráshoz való hibás mozgatását szimuláljuk. Idetartozó szekvencia diagram: **PlayerMovesToMountainSpring**

Ez a függvény a játékban a játékos áthelyezését végzi el a hegyi forráshoz, ha az lehetséges. A függvény először inicializálja a játéktáblát, majd a játékosat a megadott kezdő pozícióra. Ezután a függvény megpróbálja áthelyezni a játékosat a hegyi forrás pozíciójára. Ez a művelet sikertelen, mivel mi már a feladat megvalósításával kapcsolatban úgy döntöttünk korábban, hogy a hegyi forráshoz a lépés nem lehetséges.

Elvárt kimenet:

```
Player moves to MountainSpring has started
MountainSpring the player wants to move onto: container.MountainSpring@6e0be858
Move is called
exception.MyException: Not even next to it
Move to MountainSpring failed :)
Mission failed successfully, position is: container.Pipe@61bbe9ba
Player moves to MountainSpring has finished
```

### 6.1.2.14 Random pump breaking

Menüben a **14-es gomb** megnyomásával egy vízellátó rendszerben található szivattyúk meghibásodását szimulálunk. Idetartozó szekvencia diagram: **RandomPumpBreakdown**

A függvény inicializál egy térképet, majd egy vezérlőt hoz létre és hozzárendeli a térképet.

Ezután a függvény for ciklusa 20-szor fut, és minden iterációban meghívja a vezérlő damagePump() metódusát, amely véletlenszerűen meghibásít egy szivattyút (ezt már korábban említettük, hogy mi alapján hibásodik meg). A sleep() metódus használata miatt a függvény hívása szünetet tart a szál számára, hogy az alkalmazás megjelenítse a szimuláció változásait. A függvény végeztével az alkalmazás jelzi, hogy az összes szivattyú meghibásodott.

Egy példa kimenetre (mindig más körben romlanak el a pumpák, ezért többféle kimenet is lehet):

```
Pump breaking sequence initiated!
Current turnCount:1
Current turnCount:2
Current turnCount:3
Current turnCount:4
Current turnCount:5
Current turnCount:6
Current turnCount:7
Current turnCount:8
Current turnCount:9
Current turnCount:10
Current turnCount:11
Current turnCount:12
Current turnCount:13
Current turnCount:14
Naww this pump got damaged: container.Pump@610455d6
Naww this pump got damaged: container.Pump@511d50c0
Current turnCount:15
Current turnCount:16
Current turnCount:17
Current turnCount:18
Naww this pump got damaged: container.Pump@60e53b93
Current turnCount:19
Current turnCount:20
Pumps all up and gone!
```

### 6.1.2.15 Collecting water in cistern

Menüben a **15-ös gomb** megnyomásával szimuláljuk a víznek a mozgását egészen 7 “körön” keresztül. Idetartozó szekvencia diagram: **FlowBetweenPipePumpPipe** és

**CollectingWaterInCistern.** A szimuláció abból áll, hogy végigmenve az összes konténeren meghívunk rájuk egy kiértékelő függvényt. Ez a kiértékelő függvény mondja meg, hogy milyen feltételek alapján, hogyan mozog a víz amelyet a setInputState() függvény segítségével szemléltetünk. A kimenetben megjelenik egy ‘Before’ és egy ‘After’ kimenet is ezt az kívánja szemléltetni, hogy miféle képpen történt változás az adott konténer “állapotában”. Mindez végett az eval() függvények mellett meghívjuk a konténerekhez tartozó makeHistory() függvényt amely a “naplázást” valósítja meg.

Segítő az kimenetek értelmezésére:

-Before: MountainSpring inputStatjének első illetve második eleme:

container.MountainSpring@61bbe9ba: false,false → ez azt mutatja meg, hogy a Mountain Spring-ben “nem folyik a víz” ez igazából számunkra mégnem releváns adat hiszen Mountain Springnél majd a későbbiekben minden {true, true} lesz az inputState-ünk mivel a setInputState-je ezt valósítja meg

-After: MountainSpring inputStatjének első illetve második eleme:

container.MountainSpring@61bbe9ba: true,true Water remaining in Mountain Spring: 49 → itt látható hogy megtörtént a setInputState() és minden értéke az inputState tömbnek true, emellett még ott van az is, hogy mennyi víz maradt a Mountain Spring-be (tehát látszik, hogy folyik a víz

-Végül: After: Pipe inputStatjének első illetve második eleme:

container.Pipe@12a3a380: false,true → ez a Pipe setInputState()-jében történt változás (alapjáraton a konténerek {false, false} inputState-tel jönnek létre). Itt az setInputState()-je a Pipe-nak megváltoztatta az inputState[1] true értékre, ezzel jelezve, hogy víz folyik benne. A többi kimenet is ehhez hasonlóan történik

## Elvárt kimenet:

### **6.1.2.16 Kilépes**

Menüben a **0-s gomb** megnyomásával lehet kilépni a menüből.

### 6.1.2 és 6.1.3: Fordítás és Futtatás

**Komment:** itt lehet találni pontosan User-Guide, amelyik válaszolni pontosan lehet fordítani és futtatni kódot.

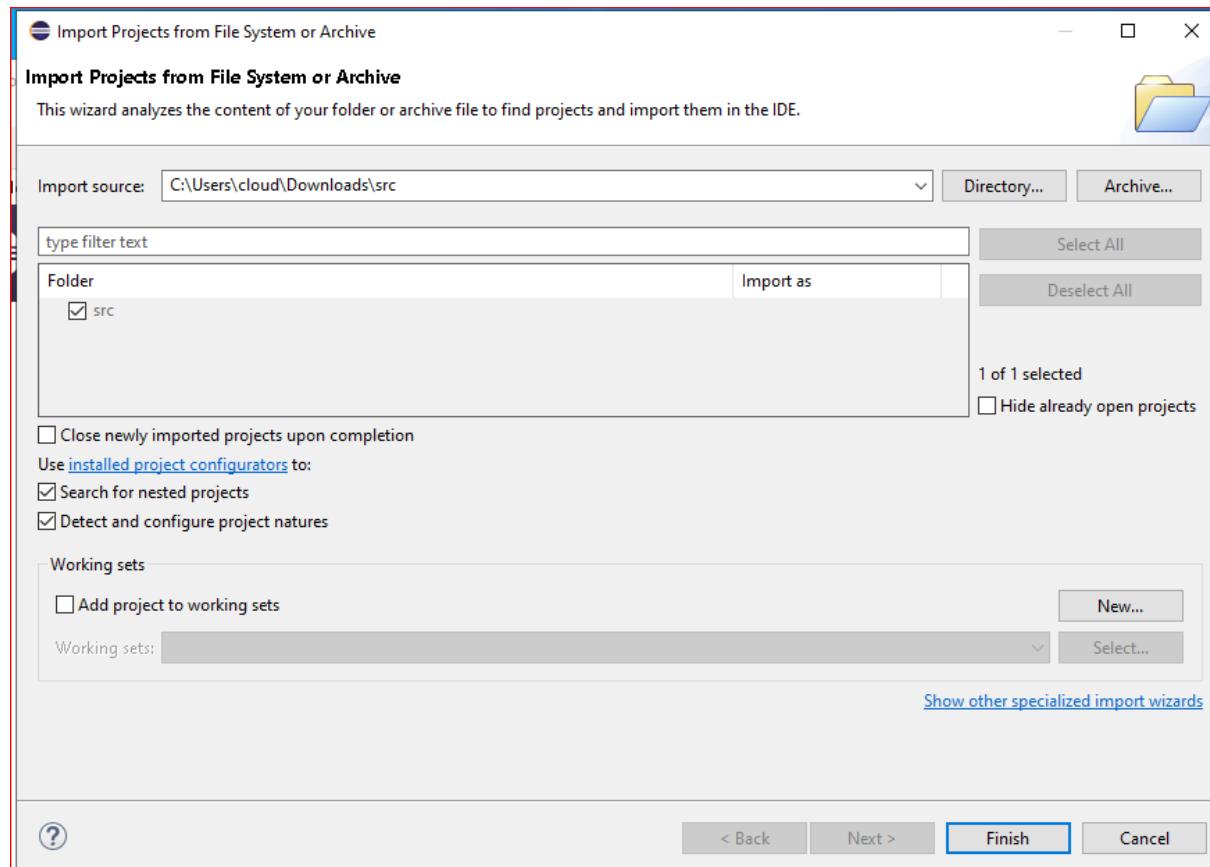
**Először, van egy GitHub repository a csapatunk számára**

**GitHub link:** [https://github.com/kopach-artem/Pipes\\_gods\\_of\\_jar.git](https://github.com/kopach-artem/Pipes_gods_of_jar.git)

**Eclipse IDE:**

1. Töltsé le a projekt mappát olyan fájlokkal, mint .java (File -> open Projects from FileSystems->Directory->src folder)
2. A kódgeneráláshoz szükséges összes csomagot benne van ( Packages, amelyik kell sikeresen fordításnak: container, controller, exception, map, player, skeleton)
3. Amikor az Eclipse-ben megnyomja a "Futtatás"(Run) gombot, a Java forráskódot bytecode-be (azaz .class fájlokká) fordítja, majd a Java Virtual Machine (JVM) végrehajtja. Ez a folyamat automatikusan történik a háttérben, és az Eclipse Java Development Tools (JDT) pluginja kezeli. De, ha szeret külön csinálni azt, akkor lehet előtt Build nyomni és utána, amikor "Fordítás" succeed lesz -> "Futtatás"(Run) gombot nyomja.
4. Megjegyzés a 3. ponthoz: Az ezekben a csomagokban felsorolt fájlokat minden le kell fordítani és futtatni kell.

Saját tesztfuttatásunk az Eclipse IDE-vel Java fejlesztőknek - 2022-06:



The screenshot shows the Eclipse IDE interface with the Main.java file open in the editor. The code prints a sequence of operations from 1 to 15, representing a sequence of events in a game. Below the code, the Console view displays the same sequence of operations. A status bar at the bottom indicates the application is running on Java 8.

```

package skeleton;
import exception.MyException;
public class Main
{
    static void printOperations()
    {
        System.out.println("\nVálassza ki a kívánt szekvenciát!");
        System.out.println("1. Mechanic repairs pump");
        System.out.println("2. Mechanic repairs pipe");
        System.out.println("3. Saboteur leaks pipe");
        System.out.println("4. Player attach pipe");
        System.out.println("5. Player attaches pump");
        System.out.println("6. Player detach pipe");
        System.out.println("7. Player adjust pump Input");
        System.out.println("8. Player adjust pump Output");
        System.out.println("9. Player moves to pipe successfully");
        System.out.println("10. Player moves to pipe fail");
        System.out.println("11. Player moves to pump");
        System.out.println("12. Player moves to cistern");
        System.out.println("13. Player moves to mountain spring");
        System.out.println("14. Random pump breaking");
        System.out.println("15. Collecting water in cistern");
    }
}

```

### CMD Windows (javac - java commands only):

1. A megbízás feltételei szerint a csapatunknak gondoskodnia kell arról, hogy a kódot a Linux operációs rendszer konzoljáról futtassuk a szokásos javac és java parancsok használatával. Ezt a lehetőséget sikeresen teszteltük és megvalósítottuk a <https://niif.cloud.bme.hu/> "Windows 10 20H2 - JDK-Eclipse-WSU" segítségével.
2. A kódot az rdesktop vm.niif.cloud.cloud.bme.hu:17760 -u cloud -p 8hr7AiF2B8 -f parancssal lehet futtatni a Linux rendszerek termináljáról (Mi például Ubuntu 20.04-et használtunk). Azonban a Windows rendszerről is elvégezhető ez a művelet.
3. Ezután klónozzuk a kódunkat erről a linkről a számítógépünkre, vagy töltük le a .zip archívumot. Ezután menjünk a projekt mappába, nyissuk meg a konzolt, majd az alábbi útvonalon használhatjuk **a parancs cmdben fordításnak:**  
**javac container\Cistern.java container\Container.java  
container\MountainSpring.java container\Pipe.java container\Pump.java  
container\Pump.java controller\Controller.java  
exception\MyException.java map\Map.java player\Mechanic.java  
player\Player.java player\Saboteur.java player\Type.java  
skeleton\DispatcherSkeleton.java skeleton>Main.java**
4. Miután a fordítási folyamat befejeződött, futtassuk a Java Virtual Machine (JVM) programot a java parancs futtatásával, amelyet a fő metódust tartalmazó osztályfájl neve követ, esetünkben a Main. Ez létrehozza a bináris futtatható kódot, amely bármely olyan rendszeren futtatható, amelyen kompatibilis JVM van telepítve.. Ezután futtassuk az unalmas fájlt **a parancs futtatásának:**  
**java skeleton.Main**  
és kapunk egy üdvözlő menüt a választásokkal és tesztekkel a csontvázunkhoz (részletesebb menü működése fentebb, az Eclipse IDE működésének leírásában található).

Saját tesztfuttatásunk a cmdben:

Fordítás és Futtatás sikerül.

```

Ubuntu20 (Снимок 2) [Running] - Oracle VM VirtualBox
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows (Version 10.0.19042.802)
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\cloudf\Downloads\src>java plac container\Cistern.java plac container\MountainSpring.java plac container\Pipe.java plac container\Hump.java plac controller\Controller.java exception\HyException.java map\Map.java
Hello! I'm a God of jar csapat skeleton program!
Válassza ki a kívánt szekvenciat!
1. Mencende repairs pump
2. Player moves to pipe
3. Saniteur leaks pipe
4. Player attach pipe
5. Player moves to pump
6. Player detach pipe
7. Player attaches pump
8. Player adjusts pump
9. Player moves to pipe Output
10. Player moves to pipe full
11. Player moves to pump
12. Player moves to cistern
13. Player breaks mountain spring
14. Handles pump breaking
15. Collecting water in cistern
16. KillGods

```

### Megjegyzés:

Fontos megjegyezni, hogy a 2. követelményben leírt környezetet biztosítjuk, amely magában foglalja az Eclipse IDE használatát és a Windows parancssorát a javac és java parancsokkal, amelyek az egyetlen szükséges eszközök a bináris futtatható kód forrásfájlok ból történő generálásához. A kód kompatibilitásának és hordozhatóságának biztosítása érdekében minden más kiegészítő eszközt vagy függőséget kerülni kell.

Szinte bármilyen fejlesztői környezetben futtathatod, mert a csapatunk készített egy praktikus GitHub tárolót, ahol láthatod a kódváltoztatási fát, és könnyen klónozhatod a számítógépedre.

## 6.2 Értékelés

Tag neve	Tag neptun	Munka százalékban
Molnár Márton	KLM60O	18.75
Kopach Artem	JQBOLI	18.75
Réti Ádám	AO7JX4	18.75
Czifra Barnabás	NX9GA4	18.75
Tisza Miklós	E1LX0J	25

### 6.3 Napló

Kezdet	Időtartam	Résznevők	Leírás
2023.04.15 12:00	1 óra	Réti Ádám Tisza Miklós Czifra Barnabás Kopach Artem Molnár Márton	Értekezlet: Feladatak kiosztása.
2023.04.15 13:00	6 óra	Réti Ádám	Tevékenység: Néhány függvény megírása (pl.takePipe,takePump,isLooseEnd..) Pump konstruktörának megírása Main osztály elkészítése
2023.04.15 - 2023.04.16 .	6 óra	Kopach Artem	Tevékenység: Létrehozott egy GitHub tárolót, összekapcsolta a csapatot. Írt különböző konstruktörököt, pl. Pipe, Cistern és mások (további részletek a GitHubon található commitok alapján értékelhetők). Dolgozott, segített és javított számos menüpontot a 4-8-as menüpontok közül. Emellett egy Eclipse IDE projektet is futtatott, létrehozta a cloud.bme-t és ott futtatta a kódot. Megírta a 6.1.2-6.1.3. pontokat. JavaDOC kommenteket írtam. Címoldal sablont megcsináltam.
2023.04.15 13:00	2 óra	Molnár Márton Czifra Barnabás	Értekezlet: DispatcherSkeleton osztály elkészítése
2023.04.16 13:00	1 óra	Molnár Márton	Tevékenység DispatcherSkeleton osztály javítása
2023.04.16 14:30	2 óra	Molnár Márton	Tevékenység: Osztályok kommentezése
2023.04.16 12:30	16 óra	Tisza Miklós	Tevékenység: Főbb függvények megvalósítása, hozzájuk illő JavaDoc elkészítése, DispatcherSkeleton javítása
2023.04.17 1:00	3 óra	Réti Ádám Kopach Artem	Értekezlet: Dokumentáció elkészítése

# Prototípus koncepciója

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

**Réti Ádám -  
csapatvezető**

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

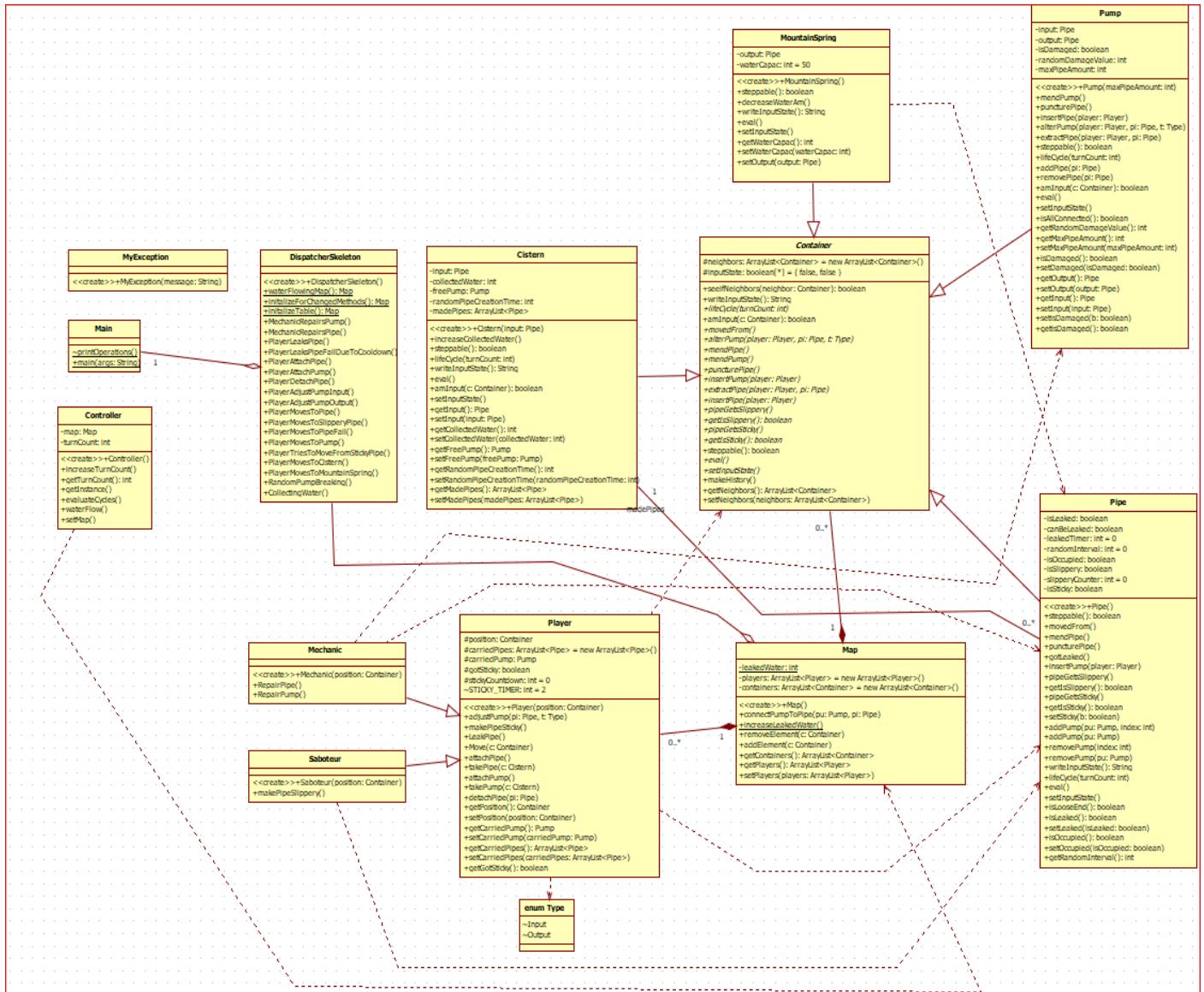
(AO7JX4)      ket.vill.adam@gmail.com  
(E1LX0J)      tisza.miklos.16@gmail.com  
(NX9GA4)      barna.czifra@gmail.com  
(KLM60O)      molnar.marton.hun@edu.bme.hu  
(JQBOLI)      kopach.artem@edu.bme.hu

2023.04.22

## 7. Prototípus koncepciója

### 7.0 Változás hatása a modellre

#### 7.0.1 Módosult osztálydiagram



### 7.0.2 Új vagy megváltozó metódusok

**Komment:** Először is szeretném kiemelni, hogy a játékban bekövetkezett változások elégé globálisak voltak, új metódusokkal és részben megváltozott koncepcióval érintik a fő szuperosztályainkat. Ezért úgy döntöttünk, hogy célszerű lenne a szuperosztályokban bekövetkezett változásokat a változásokra és újításokra vonatkozó megjegyzéssel együtt bemutatni. Elsőként a Player és a Container(*abstract*) szuperosztályokat tekintjük át. Bennük az összes változást meg fogjuk nézni, megjegyzésekkel ellátva.

#### Osztály: Player

Metódus	Leírás
+<<create>> Player(position:Container)	Ez a konstruktur Playernek, amelyben mi tudunk felállítani positiont
+adjustPump(p: Pipe, t: Type)	Most ebben a függvényben adjustPump eset csak kezdi. pu : Pump most oda nem kell küldeni.
+makePipeSticky()	Új függvény. Ez a függvény csinálni cső, amin Player áll Sticky.
+LeakPipe()	Új függvény. Ez a függvény csinálni cső, amin Player áll lyukasztani, mert "Szerelő is tud lyukasztani."
+Move(c : Container)	Változni logiká, mert most lehet, hogy cső, amin áll, rövid időre csúszóssé tudja tenni. Vagy cső, amin állnak, rövid időre ragadóssé tudják tenni.
+attachPipe()	Változni logiká, mert most attachPipe csak a függvény, amelyik csak kezdi eset és néz, hogy Playernek van carriedPipes (!getCarriedPipes.isEmpty()). Meghívni insertPipe(p: Player) : void függvényt.
+takePipe(c : Cistern)	Ha van cső Cisternában, amelyben Player áll, akkor ez függvény tud adni új csöt carriedPipes List-be
+attachPump()	A jelenlegi pozíójához adja hozzá a Pumpot.

+takePump(c : Cistern)	Elveszi a Cisterntől a szabad Pumpot.
+detachPipe(pi : Pipe)	Felveszi a paraméterül kapott csövet a jelenlegi pozíójából.
+getPosition() : Container	Visszaadni Player position (Container), amelyben áll
+setPosition(position : Container)	Új játékos pozíció beállítása a konténeren
+getCarriedPipes() : ArrayList<Pipe>	getter CarriedPipesnek
+setCarriedPipes(carriedPipes : ArrayList<Pipe>)	setter CarriedPipesnek
+getCarriedPump() : Pump	getter carriedPumpnak
+setCarriedPump(pu : Pump)	setter carriedPumpnak

### Osztály: Saboteur

Metódus	Leírás
+makePipeSlippery()	A szabotőrhöz tartozó metódus amely segítségével egy csövet (amelyen éppen áll) csúszóssá tud tenni

### Osztály: Mechanic

Metódus	Leírás
+RepairPipe()	Megjavítja a pozíónál lévő csövet (getPosition() használ )
+RepairPump()	Megjavítja a pozíójánál lévő pumpát (getPosition() használ ).

**Osztály: Container (Abstract osztály, ezért leírunk hol implementál minden metódust)**

Metódus	Leírás
+writeInputState() : String	Ez a függvény felelős a waterFlow() függvényhez kapcsolódó kiiratásokra
+lifeCycle(int turnCount)	Ezt a függvényt implementálja a Pump és Cistern osztályok A függvény azért felelős, hogy egy adott idő után megtörténjen valami a container objektumunkkal
+amInput()	Ezt a függvényt implementálja a Pump és Cistern osztályok A függvény egy boolean értéket ad vissza abból adódóan, hogy az argumentumban megadott Container inputja-e az adott Containernek (this)
+movedFrom()	Ezt a függvényt implementálja a Pipe osztály A függvény azt a feladatot látja el, hogy amikor a játékos ellép egy Container-ről (Pipe-ról), beállítsa, hogy a Pipe nem occupied
+alterPump(player: Player, pi :Pipe, t : Type)	Ezt a függvényt a Pump osztály valósítja meg. Ez a függvény felelős a pumpa output illetve inputjának átállításáért
+mendPipe()	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős a cső megjavításáért
+mendPump()	Ezt a függvényt a Pump osztály valósítja meg. Ez a függvény felelős a pumpa megjavításáért
+puncturePipe()	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős a cső meglékeléséért
+insertPump(player : Player)	Ezt az függvényt a Pipe osztály valósítja meg. Ez az függvény felelős pumpa csőhöz való illesztéséért

+extractPipe(player : Player, pi : Pipe)	Ezt a függvényt a Pump osztály valósítja meg Ez a függvény felelős a csőhöz tartoző szabadvégű csőnek leillesztéséért
+insertPipe(player : Player)	Ezt a függvényt a Pump osztály valósítja meg Ez a függvény felelős a cső pumpához illesztéséért
+pipeGetsSlippery()	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős a cső csúszóssá válak
+getIsSlippery() : boolean	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős visszaadni csúszóssá-e Pipe (getter)
+pipeGetsSticky()	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős a cső ragadóssá válak
+getIsSticky() : boolean	Ezt a függvényt a Pipe osztály valósítja meg. Ez a függvény felelős visszaadni ragadóssá-e Pipe (getter)
+steppable() : boolean	Ezt a függvényt megvalósítja a Pump, Pipe, Cistern illetve MountaiSpring osztályok Ez a függvény visszaad egy boolean értéket az alapján, hogy az adott Container-re lehet-e lépni
+eval()	Ezt a függvényt megvalósítja a Pipe, Cistern, MountainSpring Ez a függvény felelős a víz mozgásáért való kiértékelésekért
+setInputState()	Ezt a függvényt megvalósítja a Pump, Pipe, Cistern illetve MountaiSpring osztályok Ez a függvény felelős az inputState megváltoztatásáért, ez felelős a víz tényleges mozgásáért
+makeHistory()	Az evaluation-t (kiértékelést) követően az inputState értékeit megváltoztatjuk

+getNeighbors() : ArrayList<Container>	Visszatér a neighbors attribúmmal
+seeIfNeighbors(neighbors : ArrayList<Container>)	Beállítja a neigbors attribútumot a paraméterként kapotttra

**Osztály: Map**

Metódus	Leírás
+<<create>>Map()	Map osztály konstruktora
+connectPumpToPipe(pu : Pump, pi : Pipe)	Ez a függvény felelős a paraméterként kapott pumpát és csövet egymáshoz csatlakoztatja.
+increaseLeakWater()	Ez a függvény felelős növeli a leakedWater attribútum értékét.
+removeElement(c : Container)	Ez a függvény felelős eltávolít egy Container a játéktéről
+addElement(c : Container)	Ez a függvény felelős hozzáad egy Containert a játéktérhez
+getContainers() : ArrayList<Container>	Ez a függvény felelős visszatér a containers attribútum értékével.
+getPlayers() : ArrayList<Player>	Ez a függvény felelős visszatér a player attribútum értékével.
+setPlayers(players : ArrayList<Player>)	Ez a függvény felelős beállítja a player attribútumot a paraméterként kapotttra.

**Osztály: DispatcherSkeleton**

Metódus	Leírás
+<<create>>DispatcherSkeleton()	DispatcherSkeleton osztály konstruktora
+waterFlowingMap() : Map	Inicializál egy pályát a víz mozgatásának bemutatásához
+initializeForChangeMethods() : Map	Ez a függvény a módosított metódus inicializálója a programban.
+initializeTable() : Map	Inicializál egy pályát a programhoz tartozó alkatrészek bemutatásához

**Osztály: MyException**

Metódus	Leírás
+<<create>>MyException(String message)	MyException osztály konstruktora, amelyben ahová fárasztó hiba vagy incidens üzenetet küldünk

**Osztály: Main**

Metódus	Leírás
+printOperations()	Print játék (teszt) menüt
+main(args: String)	Program main, ahol fogunk hívni függvényeket DispatcherSkeletonból

**Osztály: Controller**

Metódus	Leírás
+<<create>>Controller()	Controller osztály konstruktora
+getInstance() : Controller	függvény, amely létrehozni, ha nincs
+evaluateCycles()	Ebben a függvényben történik meg a pumpák megrontása, avagy elrontása Egyszerűen csak végigmegyünk a pálya konténerjein és mindenki meghívjuk a lifeCycle() függvényt (amely függvényre majd azon példányok felelnek csak akik tudnak)
+increaseTurnCount()	növeli a játéka paraméterét turnCount
+getTurnCount() : int	getter turnCountnak
+waterFlow()	Ez a függvény hívni eval() minden containernek és makeHistory() is.
+setMap(map : Map)	Leállítjuk a paraméterben kapott pályát a Controller pályájára

**Osztály: <<enum>> Type**

Input, Output vannak benne.

Komment:A pumpa módosításának típusát határozza meg(azaz, hogy az AdjustPump(pi: Pipe, type: Type) függvénynél bemeneti csövet vagy kimeneti csövet változtatunk, amennyiben bemeneti csövet akkor a type értéke Input, amennyiben pedig kimeneti csövet a type értéke Output)

## Szekvencia-diagramok

**Komment:** Ebben a részben nemcsak módosított szekvencia-diagramok, hanem új szekvencia-diagramok is lesznek. **Az új szekvencia diagramokat a számok alapján láthatja:**

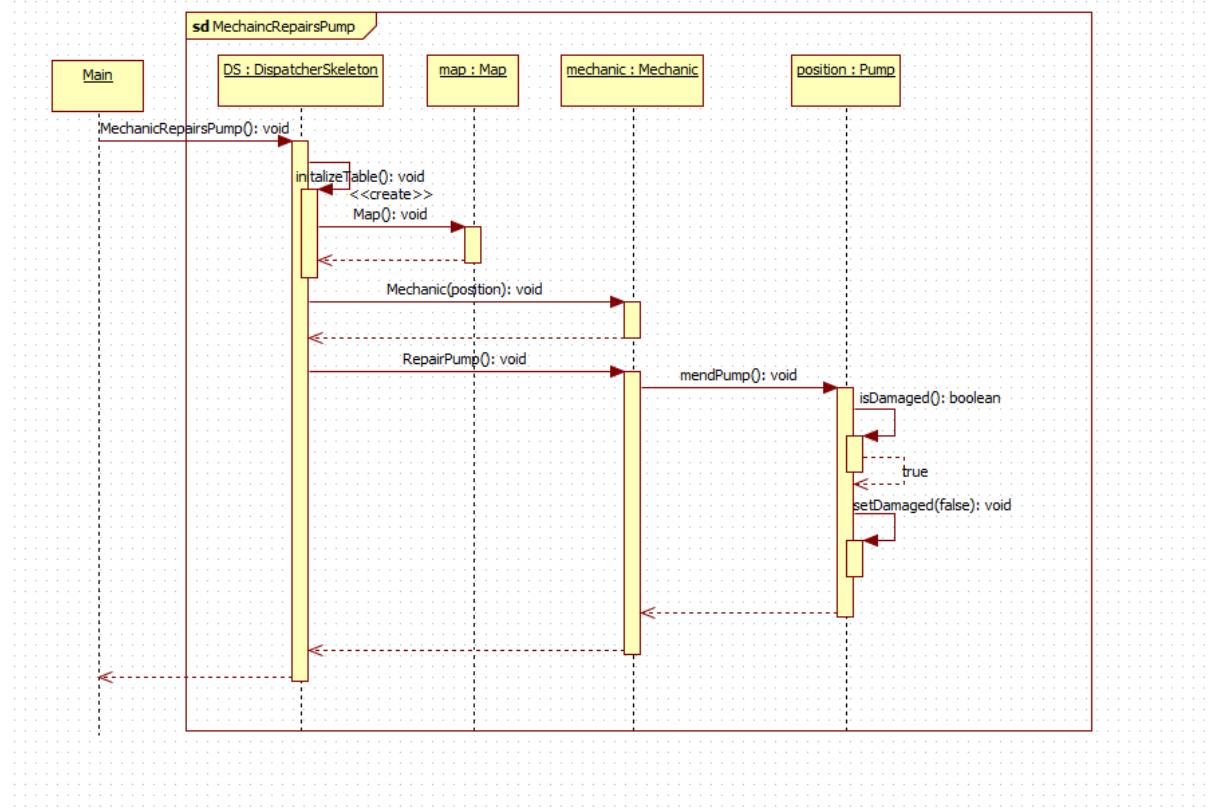
**7.0.3.5 Player Leaks Pipe** ebben a diagramban most Player fő objektum ez Player, mert ez módosítás volt “Szerelő is tud lyukasztani.”;

**7.0.3.12 Player Moves to Slippery Pipe** itt lesz eset, amikor Player próbál menni csúszóssá csőre, mert ez módosítás volt “A szabotőr azt a csövet, amin áll, rövid időre csúszóssá tudja tenni. Ilyenkor aki rálép, véletlenszerűen a cső valamelyik végéhez kapcsolódó elemre kerül. ”;

**7.0.3.13 Player Tries to Move From Sticky Pipe** itt lesz eset, amikor Player próbál menni ragadóssá csőre, mert ez módosítás volt “Mind a szabotőrök, mind a szerelők azt a csövet, amin állnak, rövid időre ragadóssá tudják tenni. Aki legközelebb rálép, egy ideig nem tud továbblépni.”;

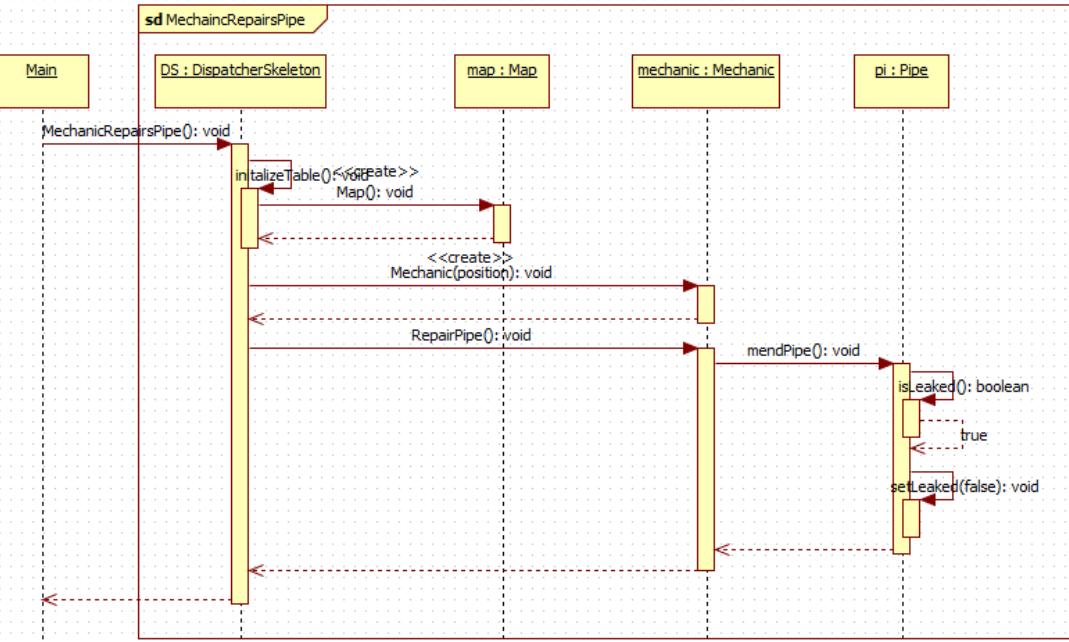
**7.0.3.17 Player Leaks Pipe Fail due to Cooldown** ez a diagram leírja eset, amikor Player próbál lyukasztani Pipe, de nem tud, mert ez módosítás volt “Foltozott cső véletlen hosszúságú ideig nem lyukadhat lyukasztható ki.”

### 7.0.3.1 Mechanic Repairs Pump (Amikor Pump - Damaged)



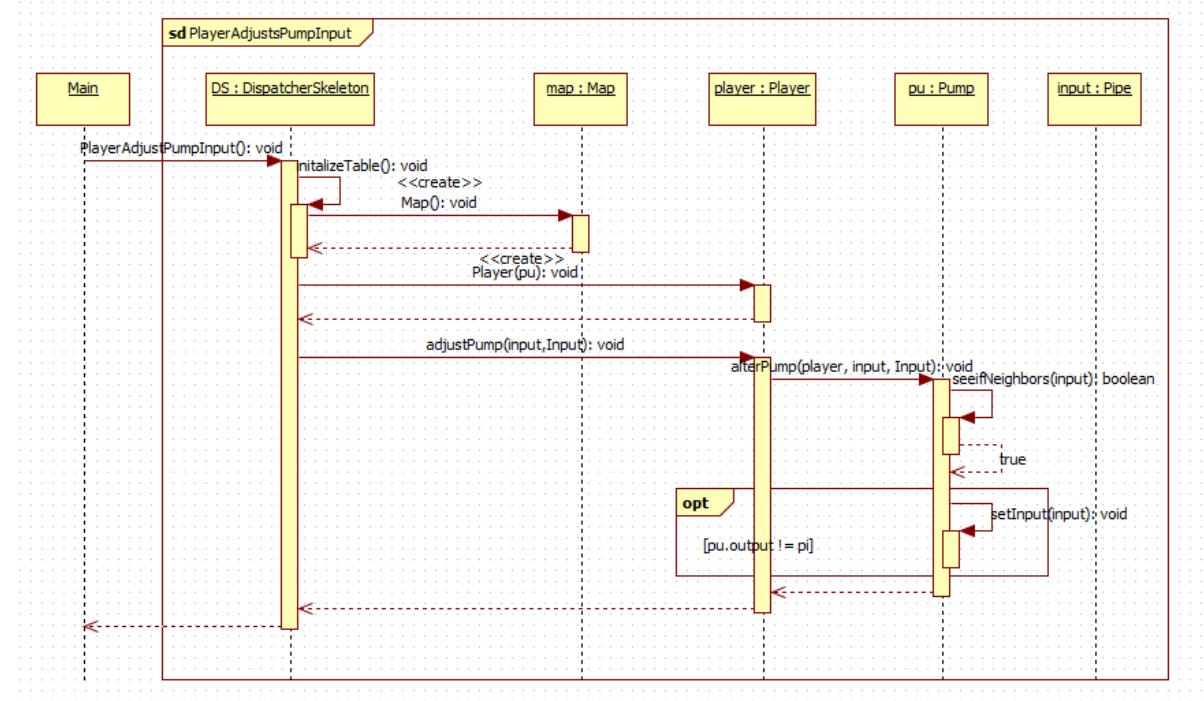
**Komment:** A DispatcherSkeleton osztályunk meghívja az (m) Mechanic objektumra a RepairPump(pu) függvényt amely elsősorban lekérdezi a (pu) Pump objektum *isDamaged* attribútumát (annak érdekében, hogy olyan pumpát ne tudjon megjavítani a szerelő amely pumpa alapjáraton nem sérült). Erre a lekérdezésre megkapjuk a pumpa állapotát és ha *true* értéket kapunk egyszerűen a pumpa *isDamaged* attribútumát egy setter fügvénnyel átállítjuk.

### 7.0.3.2 Mechanic Repairs Pipe (Amikor Pipe - Leaked)



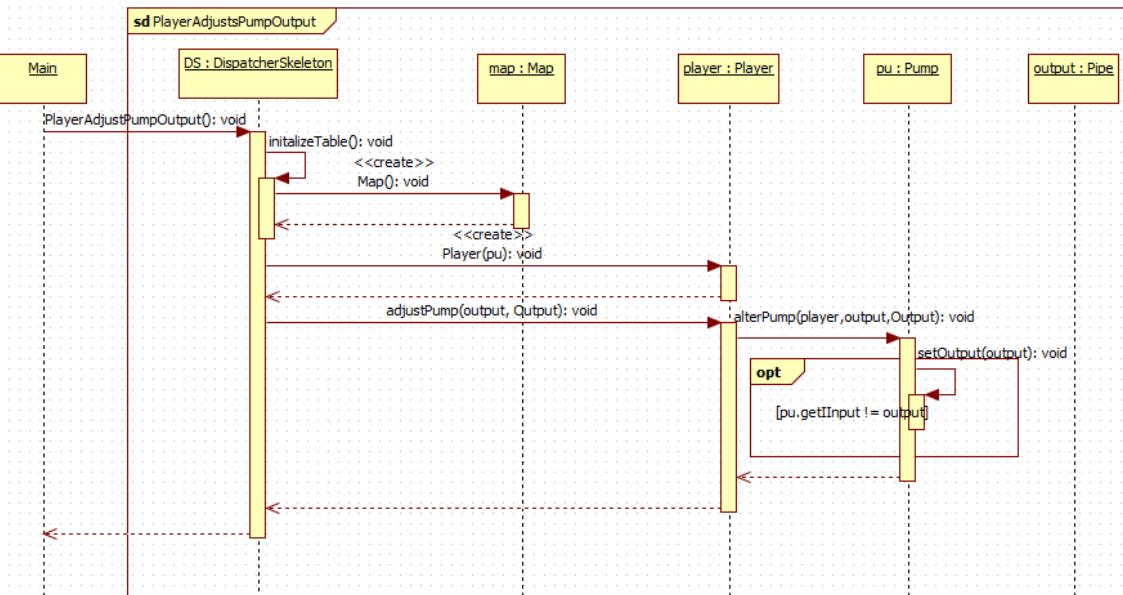
**Komment:** Hasonlóan az előző szekvencia diagramhoz a DispatcherSkeleton osztályunk meghívja az (m) Mechanic objektumra a RepairPipe(pi) függvényt amely elsősorban lekérdezi a (pi) Pipe objektum isLeaked attribútumát (annak érdekében, hogy olyan csövet ne tudjon megjavítani a szerelő amely cső alapjáraton nem sérült). Erre a lekérdezésre megkapjuk a cső állapotát és ha *true* értéket kapunk egyszerűen a cső isLeaked attribútumát egy setter fügvénnyel átállítjuk.

### 7.0.3.3 Player Adjust Pump Input



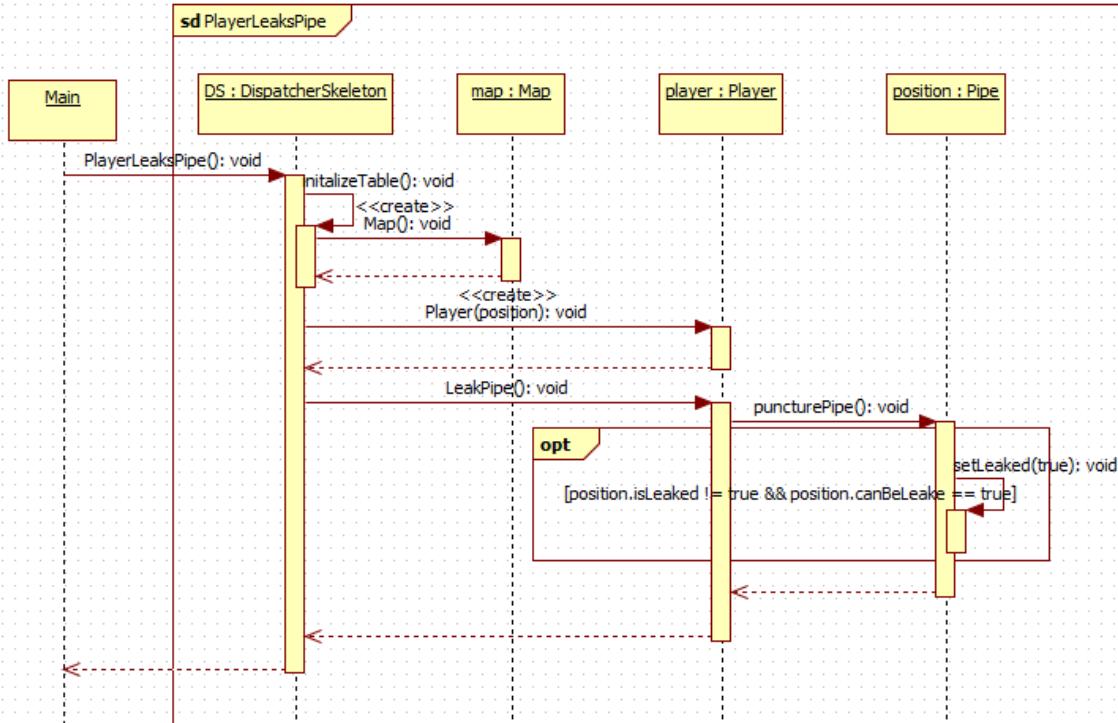
**Komment:** Ebben a szekvencia diagramban egyszerűen azt követjük végig, hogy a játékos a pumpa Input csövét hogyan is állítja át egy másik csőre. A DS meghívja (p) Player objektumra az AdjustPump(pu,pi, Input) függvényt. Ezt követően a függvényen belül lekérdezzük (pu) pumpa szomszédjainak listáját (Ez csak abból a szempontból fontos, hogy az AdjustPump argumentumában megadott (pi) Pipe szomszédja-e (pu) Pump-nak más különben nem foglalkozunk tovább ezzel és visszatérünk). Amennyiben az argumentumban megadott cső ténylegesen szomszédja (pu) Pump-nak lekérdezzük, hogy a (pu) Pump mostani Inputjában folyik-e éppen víz (amennyiben folyik víz benne azaz a getWaterFlowing() true értékkel tér vissza nem tudjuk megváltoztatni az inputot, mivel ez egy fontos feltevés volt a use-case leírásoknál is). Legvégül az argumentumban kapott (pi) csővet egy setter segítségével megtesszük (pu) Pump inputjává.

### 7.0.3.4 Player Adjust Pump Output



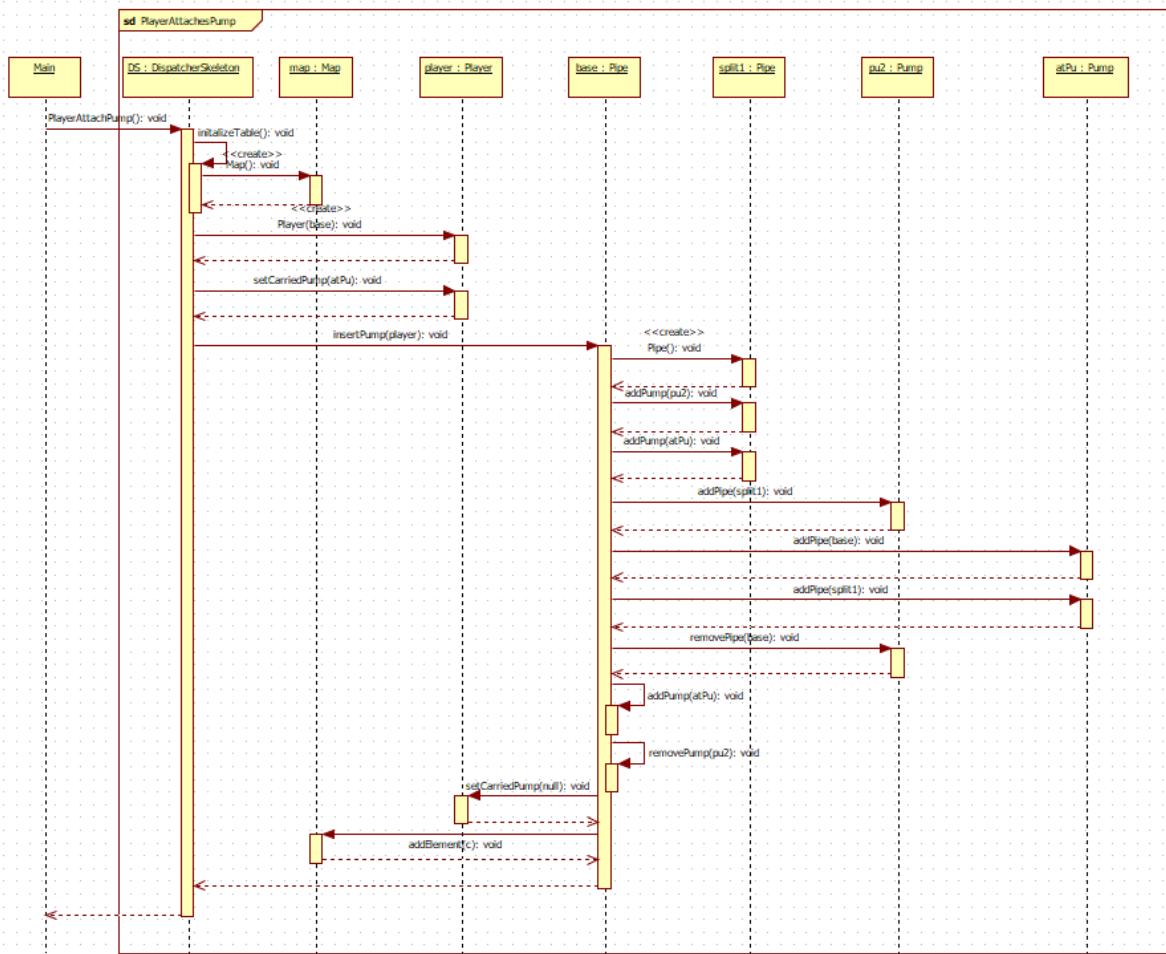
**Komment:** Ennek a szekvencia diagramnak a leírása egy ponttól elvéve teljes mértékben megegyezik az előző leírásával. Itt az Output csövet változtatjuk meg de mivel nincsenek számunkra nem kedvező feltételek az Output cső változtatására ezér egyszerűen csak megvizsgáljuk a két elem szomszédosságát majd amennyiben szomszédosak egymással a megadott csövet a pumpa Outputjának tesszük.

### 7.0.3.5 Player Leaks Pipe



**Komment:** A csövet szerelő és szabotőr is lyukaszthatja a korábbi, csak szabotőr által lyukasztható verzióban. További változás, hogy vannak csövek, amiket nem lehet kilyukasztani(egy darabig), hiszen követelmény, hogy egy foltozott cső véletlenszerű ideig nem lyukasztaható ki, tehát arre is rá kell vizsgálnunk, hogy a cső lyukasztható állapotban van-e.

### 7.0.3.6 Player Attaches Pump

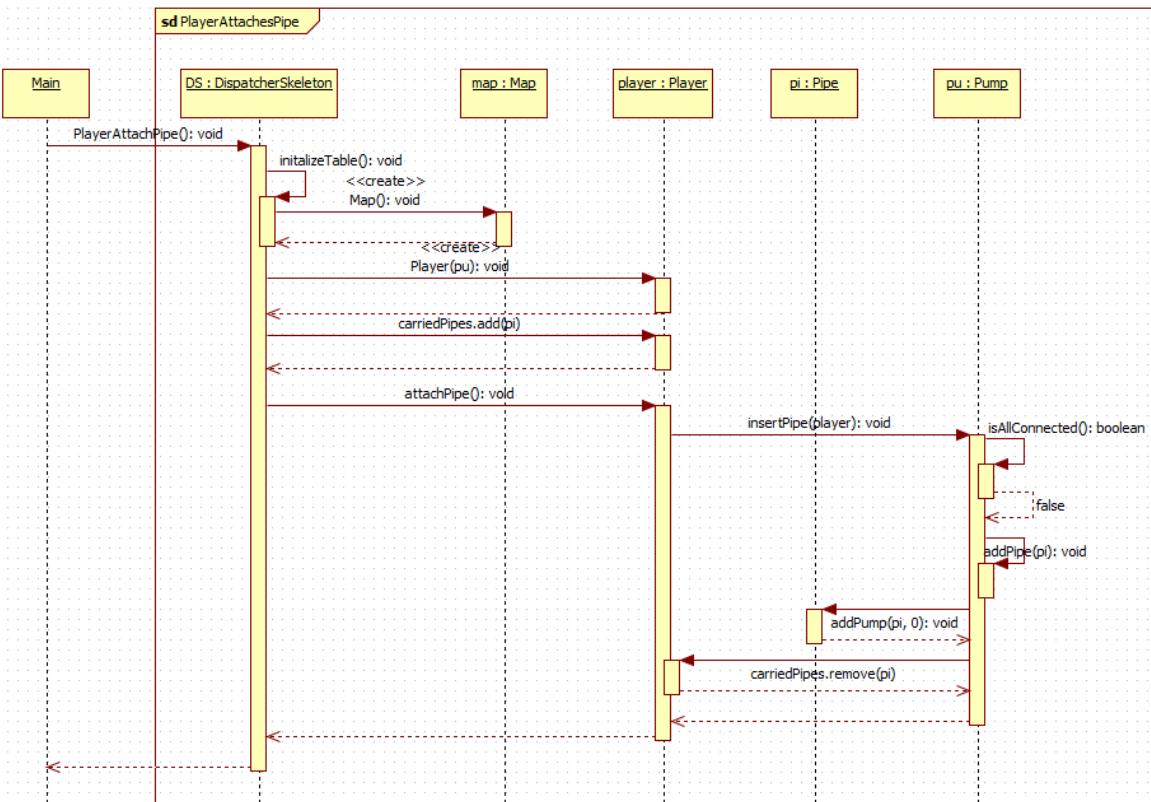


**Komment:** Ezen szekvencia diagram mutatja meg az osztályok egymással való kommunikációt egy pumpa illesztés akciónál. A pumpa illesztésének folyamatát (most itt éppen) a DS kezdeményezi egy attachPump(pu) függvénytel. A pumpát azon csőhöz fogjuk illeszteni amelyen éppen a játékos áll, jelen esetben (base) Pipe objetumunkhoz. Az első fázisa az illesztésnek a (base) Pipe felosztása illetve a környező pumpák lekérdezése (ezt egy getNeighbor() függvény hívásánál kapjuk meg azonban az egyszerűség kedvéért most itt csak lekérdeztünk kettő adott pumpához hogy szomszédja-e (base) Pipe-nak). Mivel az illesztés úgy megvégbe, hogy a játékos félbe vág egy csövet ezért itt létre kell hoznunk két másik cső objektumot jelen esetben (split1) Pipe és (split2) Pipe objektumokat. Ezek létrehozása után egyenként meg kell tennünk az illesztéseket (példaként kövessük végig (split1) Pipe objektumunkat):

-A (split1) Pipe objektum pumpa listájába hozzáadjuk az újonnan beillesztett pumpát (atPu) illetve az eredeti csőhöz (base) tartozó valamely szomszédját (jelen esetben (pu1) csövet)

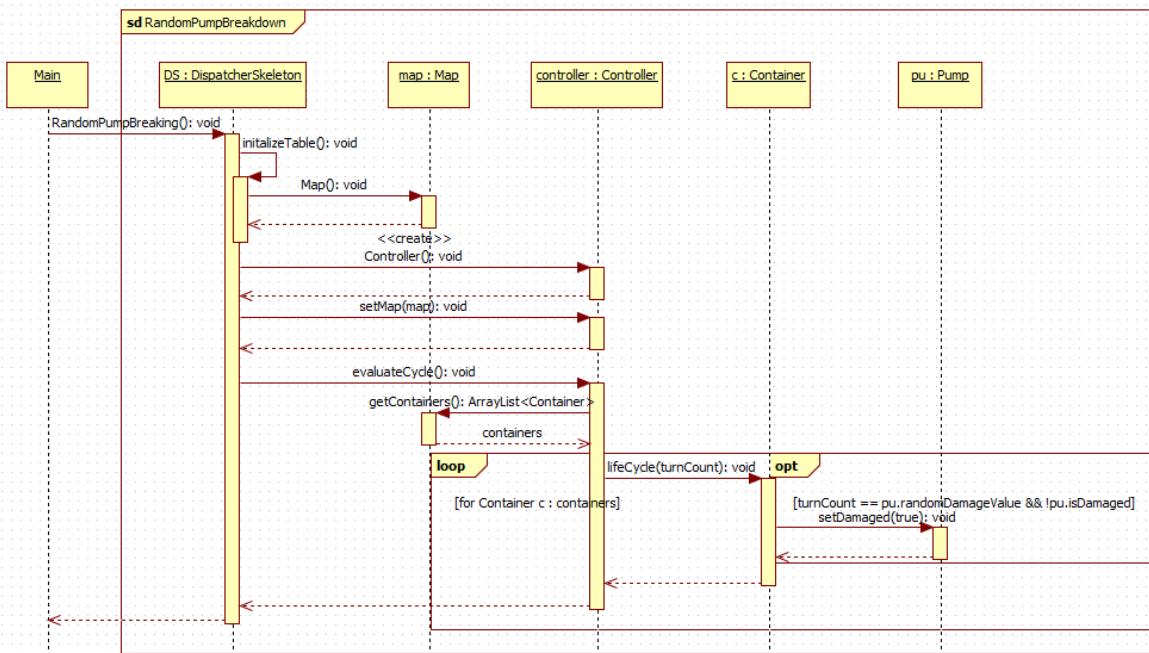
-Ezután értelemszerűen hozzáadjuk (pu1) és (atPu) pumpák cső listájához (split1) csövet)

### 7.0.3.7 Player Attaches Pipe



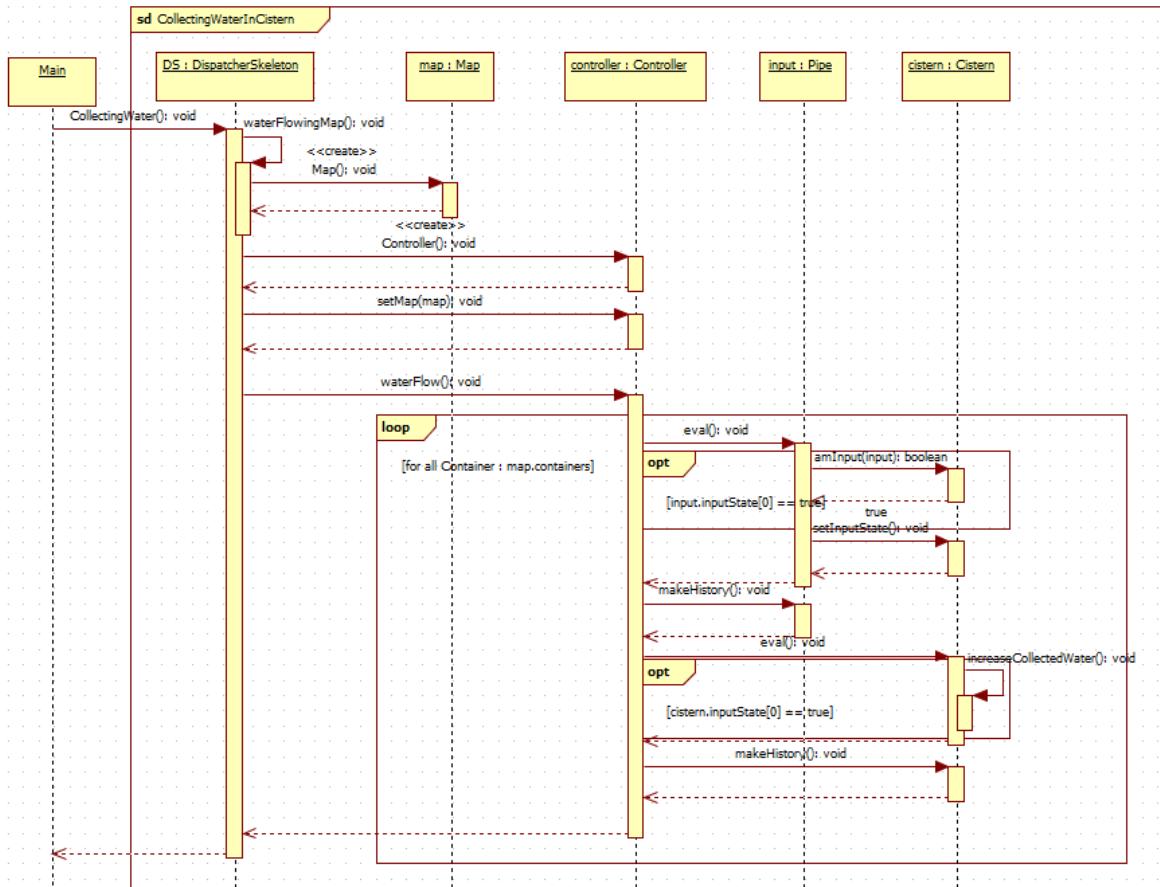
**Komment:** Ebben a szekvencia diagramban tüntejük fel a játékosok/játékos csőillesztési akcióját. A DS meghívja a (p) Player-re az `attachPipe(pi)` függvényt, ezután ezen függvényen belül lekérdezzük egy `isAllConnected()` metódus segítségével a (position) Pump objektumtól, hogy az összes csatlakoztatási helyre csatlakoztatva van-e cső. Amennyiben nem (azaz false értéket ad vissza) a pumpa cső listájához egy `addPipe(pi: Pipe)` függvény segítségével hozzáadjuk a csövet a pumpához, majd ez egy `addPump(pu: Pump)` függvénytel megtesszük a csónál is (tehát mind a csőhöz, mind a pumpához hozzáadtuk az egyiket). Legvégül a játékos `carriedPipes` listájából kiveszük az éppen lerakott cső elemét.

### 7.0.3.8 Random Pump Breakdown



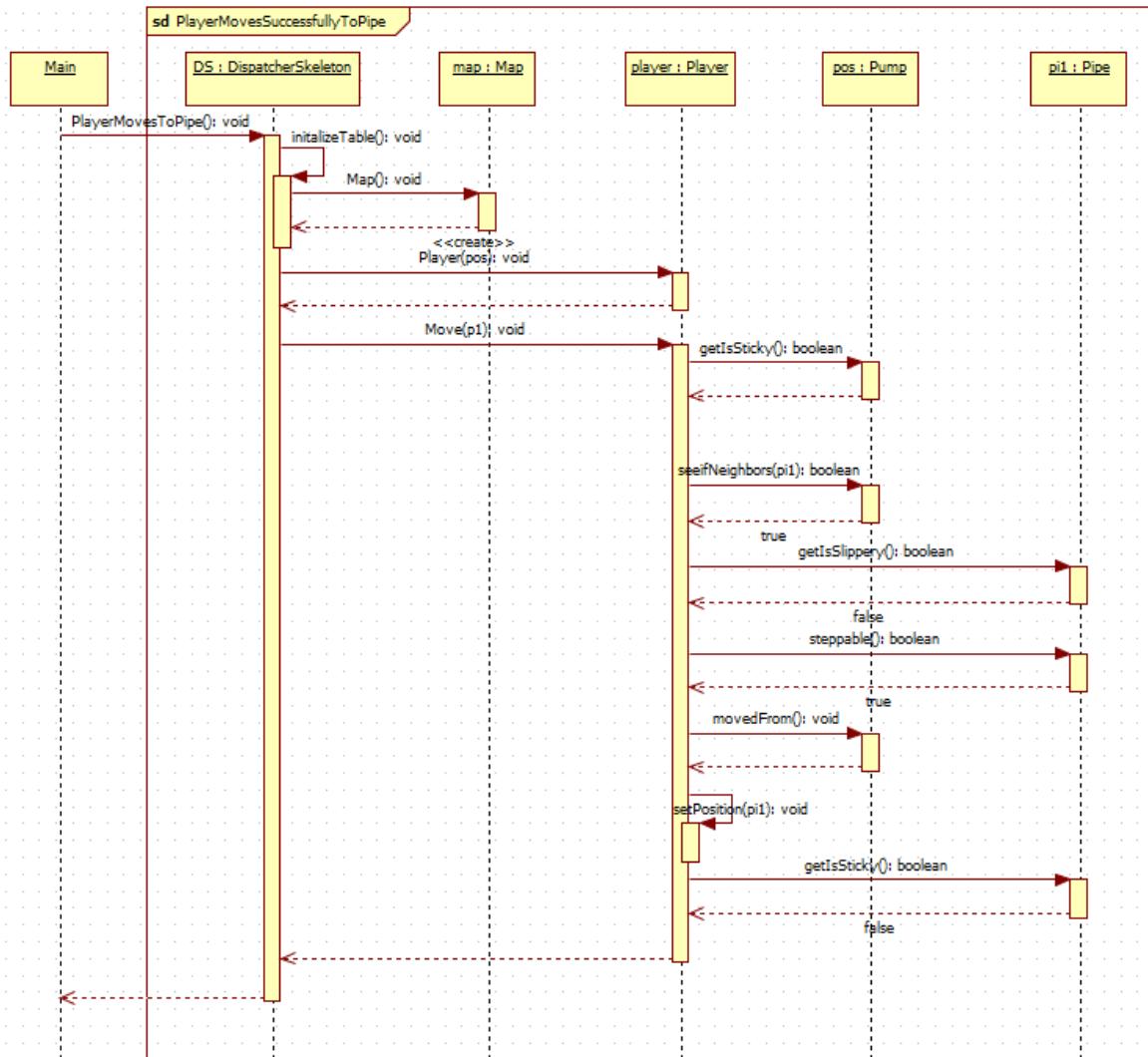
**Komment:** Itt a véletlenszerű elomlását írjuk le a pumpának. Végigmegyünk a Map osztály konténer listájában lévő összes Pump elemen és amely elemnek a randomDamageValue-ja megegyezik az eltelt körök számával akkor az adott pumpa elromlik

### 7.0.3.9 Collecting Water in Cistern



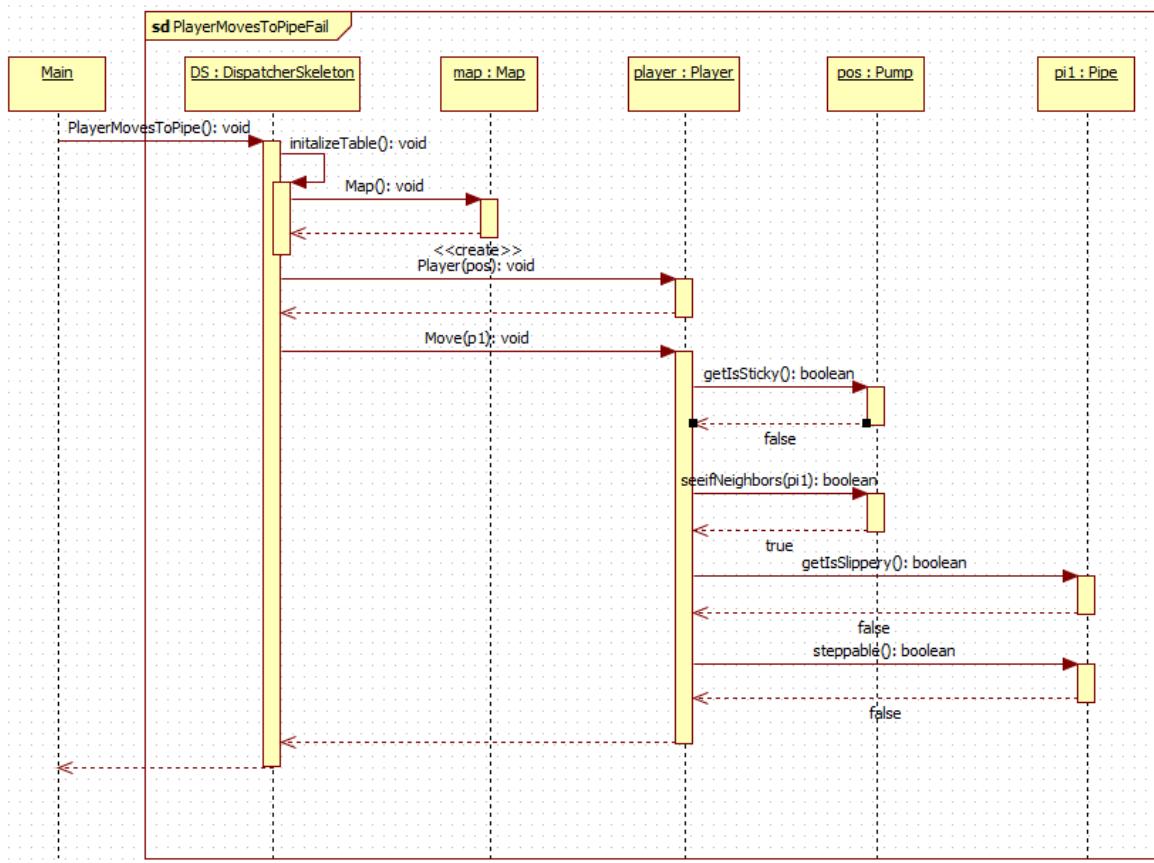
**Komment:** Ebben a szekvencia diagramban a Ciszternába beérkező víz akcióját követjük végig. Maga a folyamat teljesen megegyezik az 5.3.9-es szekvencia diagrammal egy eltéréssel, hogy amikor a következő aktív elem amire meghívjuk az eval() metódust a ciszterna, akkor ez a kiértékelés a ciszternába bemenő víz mennyiségek növeléséért felelős csak..

### 7.0.3.10 Player Moves Successfully to Pipe



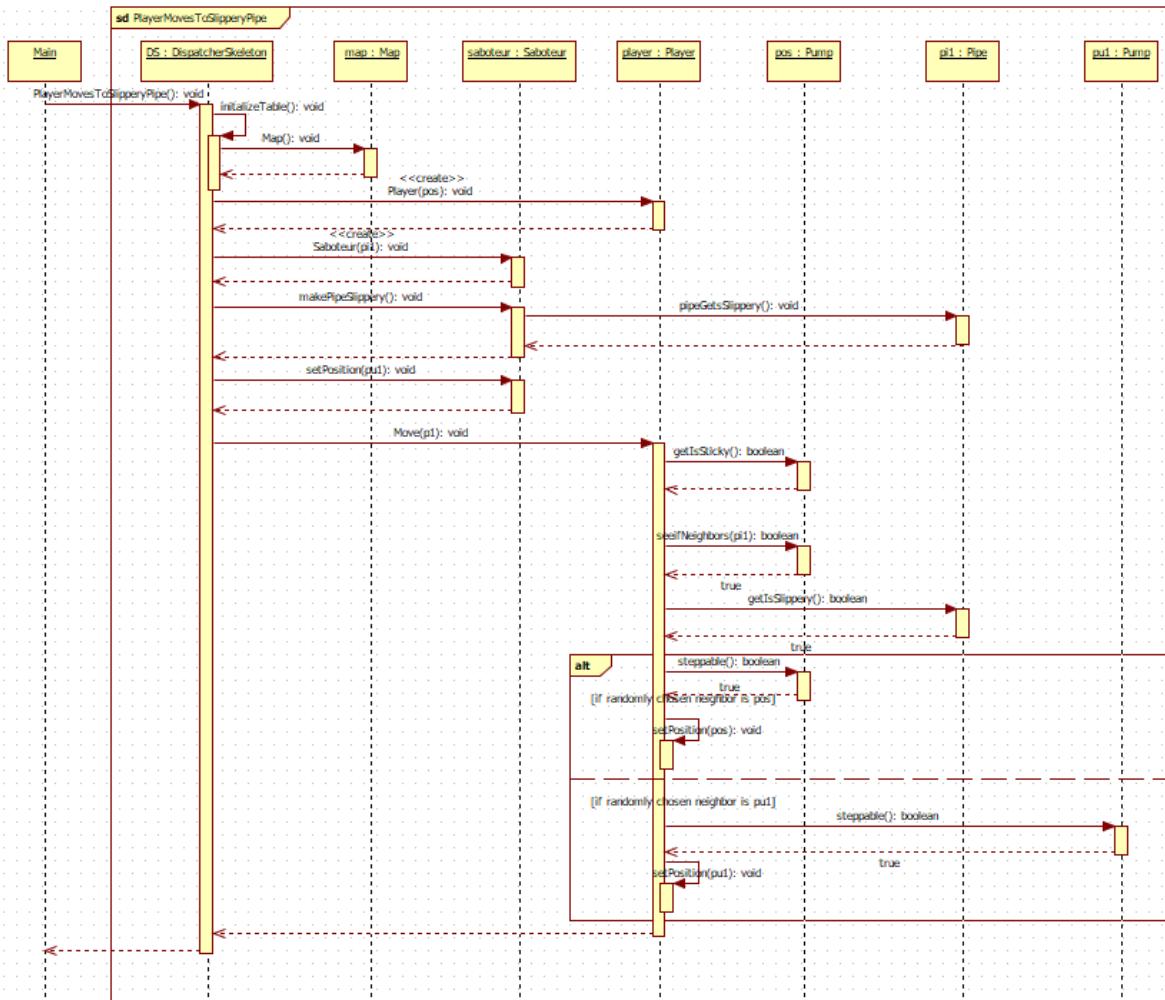
**Komment:** Először a DispatcherSkeleton `Move(pi)` kezdeni, utána a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben minden igaz, akkor Player sikerül lépni, ezzel új `setPosition(pi)` hívja.

### 7.0.3.11 Player Moves to Pipe Fail



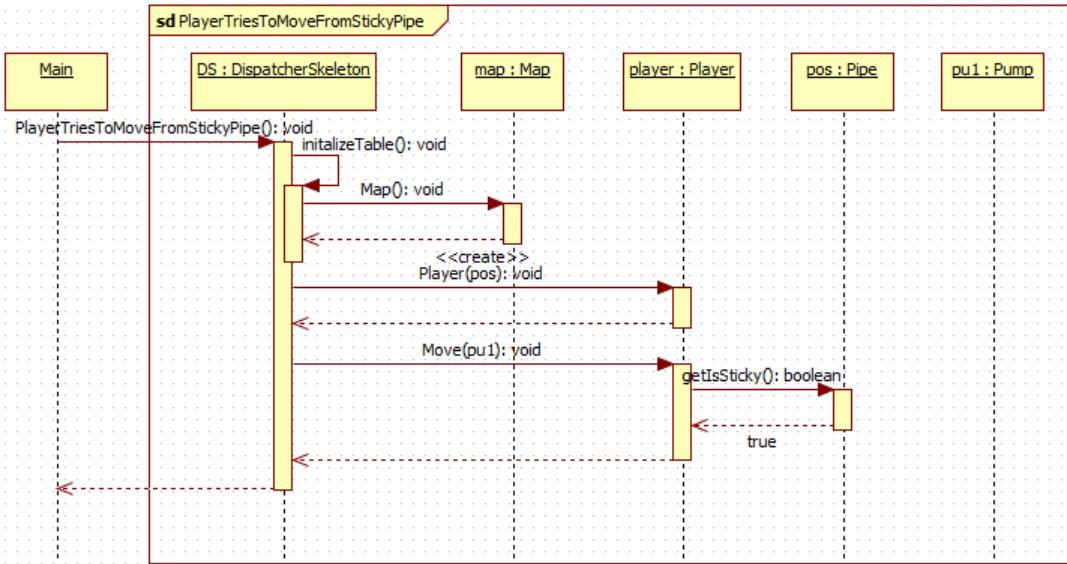
**Komment:** Először a DispatcherSkeleton `Move(pi)` kezdeni, utána a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben `steppable` igaz visszaadni, de Pipe nem szomszédja, ezért Player nem tud oda menni.

### 7.0.3.12 Player Moves to Slippery Pipe



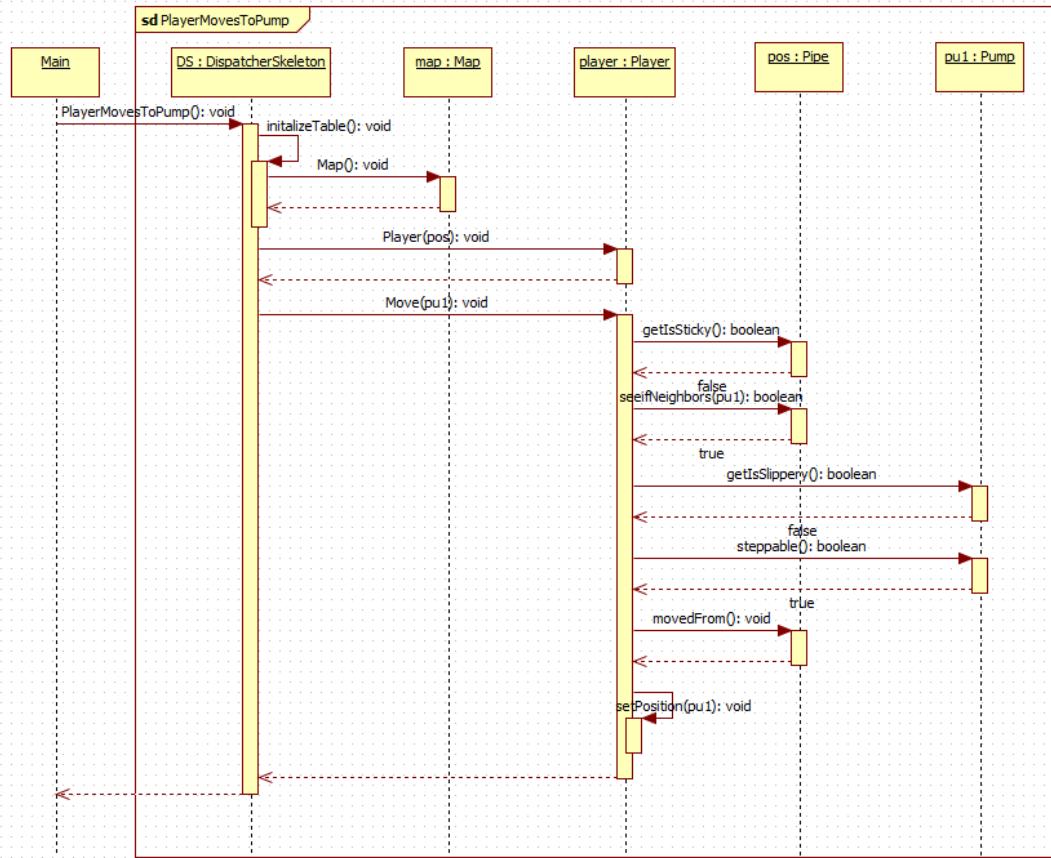
**Kommet:** A szabotőr csúszóssá tette a csövet, amire a játékos rá akar lépni. A játékos ha rálép a csőre, akkor a csövön álló játékos átkerül a cső két végét alkotó pumpa valamelyikére.

### 7.0.3.13 Player Tries to Move From Sticky Pipe



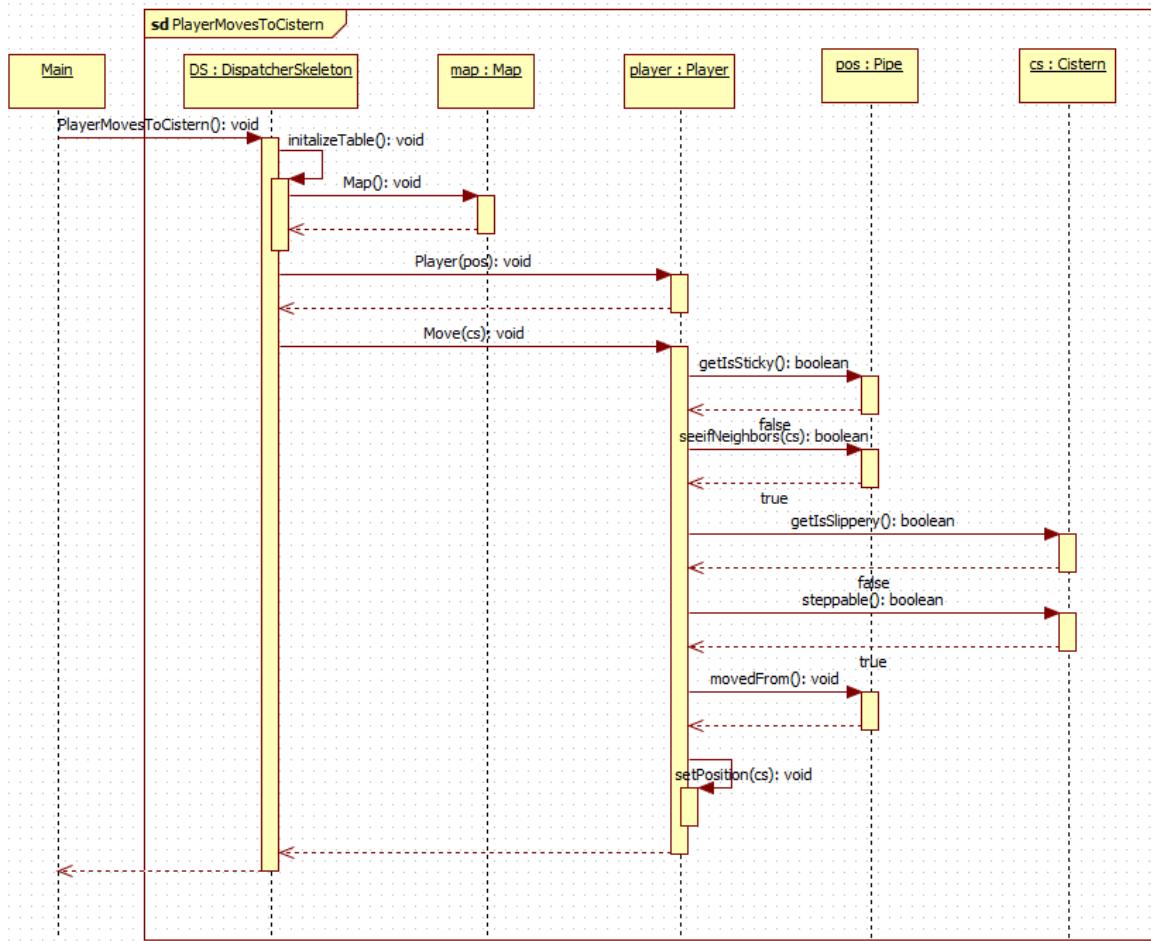
**Komment:** A játékos elmozdul a ragadós csőről. A ragadósság vizsgálata után(getIsSticky) átlép egy másik objektumra.

### 7.0.3.14 Player Moves to Pump



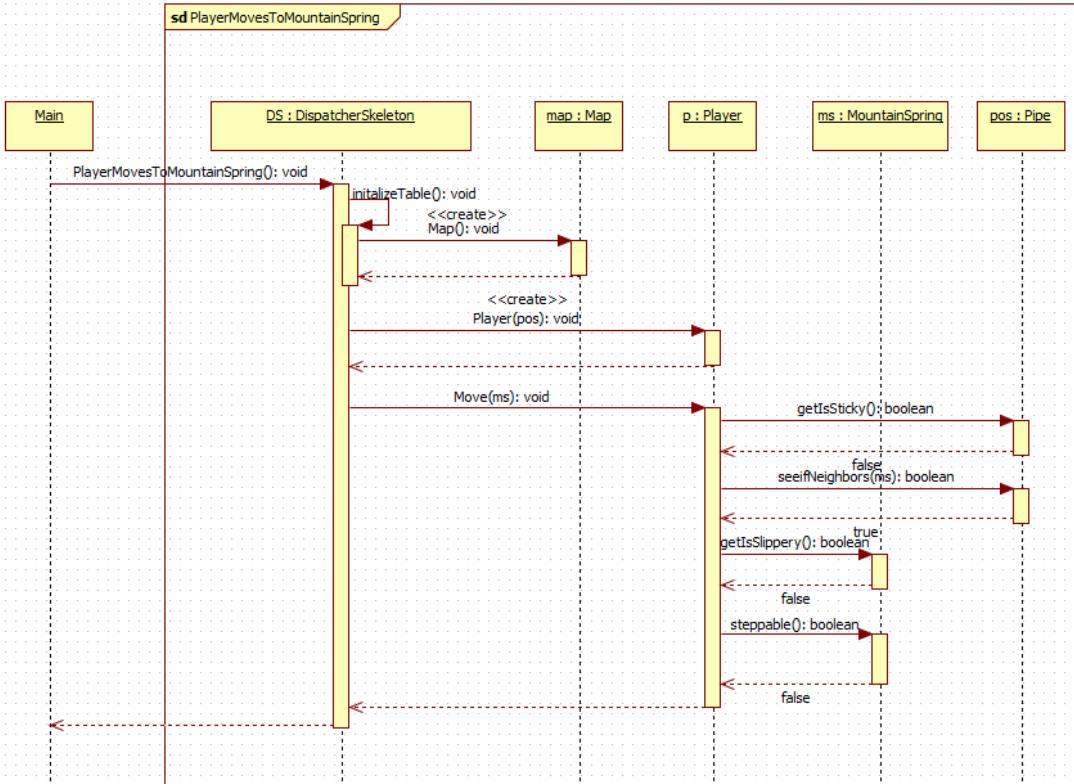
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben alt kezde, ha Pump szomszédja, akkor Player lehet oda menni és `setPosition` csak berakni új pozíciót , ha nem, akkor nem tud lépni, mert ez az aktív eleme, nem szomszédja

### 7.0.3.15 Player Moves to Cistern



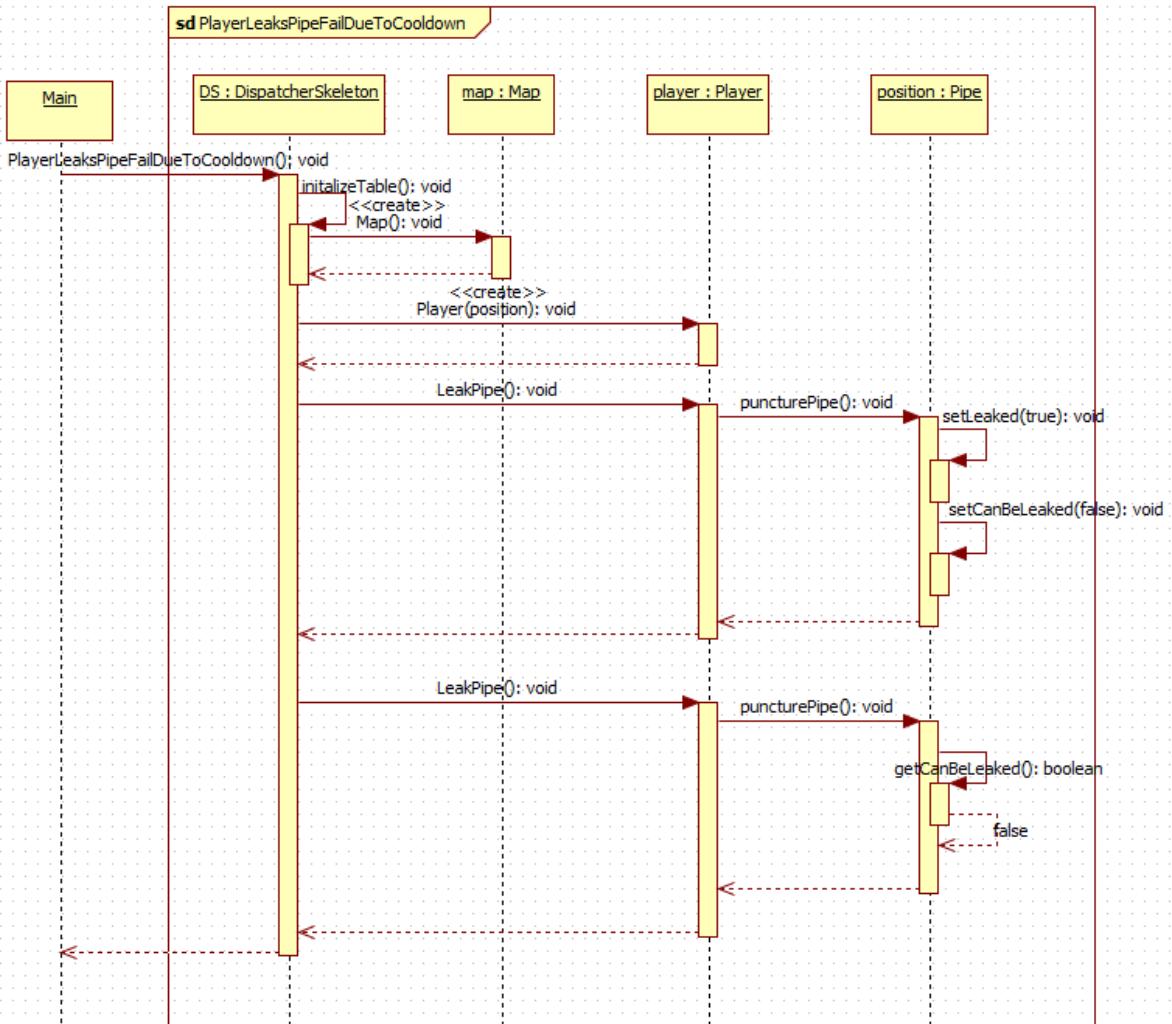
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja , majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben alt kezde, ha Cistern szomszédja, akkor Player lehet oda menni és `setPosition` csak berakni új pozíciót , ha nem, akkor nem tud lépni, mert ez az aktív eleme, nem szomszédja

### 7.0.3.16 Player Moves to Mountain Spring



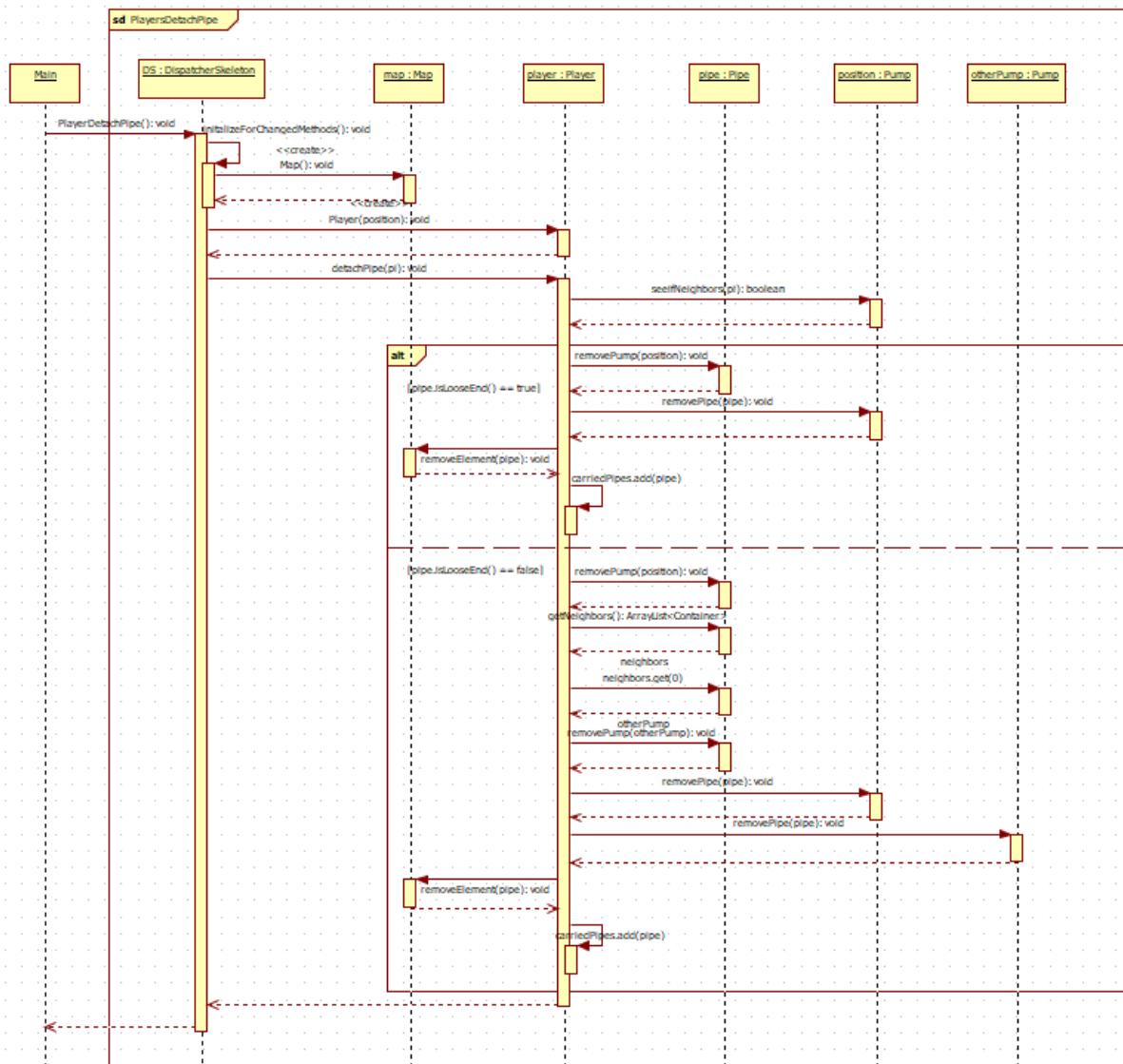
**Komment:** Először a Player megkérdezi, hogy ráléphet-e az aktív elemre `steppable()` hívja, majd fontos, hogy a játékos ellenőrizze, hogy az elem a szomszédja-e `seeifNeighbours()` hívja. Ebben az esetben `steppable` hamis (Mert ez a `MountainSpring`) visszaadni, ezért Player nem tud menni erre az elemre

### 7.0.3.17 Player Leaks Pipe Fail due to Cooldown



**Komment:** Ebben a diagramban tudunk látni, hogy Player akar kétszer `LeakPipe()` hívni, de ez most nem lehetséges egy sorban csinálni, mert, amikor hívta `LeakPipe()` utána `puncturePipe()`, amelyik `setLeaked(true)` (igazra csinálni) és `setCanBeLeaked(false)` ("Foltozott cső véletlen hosszúságú ideig nem lyukasztható ki.") ez jelent, hogy, amikor Pipe lyukasztani utána nem lehet újra lykazni ideig

### 7.0.3.18 Player Detach Pipe



**Komment:** A diagramon csak annyi módosult, hogy most már nem csak a szabad végű csöveket lehet levenni, hanem bármelyiket. Azt, hogy egy csőnek minden vége leszerelhető, mi úgy értelmeztük, hogy bárhonnan elvehető. Az nem volt megfogalmazva, hogy mi történik ha két köztes elem közé csatlakoztatott csőnek minden véget leszereljük, tehát mi úgy döntöttünk, hogy ilyenkor a játékos inventoryjába kerül a cső. Ez azért van, hogy elkerüljük, hogy a semmiben, csatlakozatlan csövek lebegjenek a pályán.

## 7.1 Prototípus interface-definíciója

### 7.1.1 Az interfész általános leírása

A prototípus rendszer input és output felületei karakteres formában érhetők el. Az input fájlból is beolvasható, az output pedig fájlba menthető. A tesztek leírásához egyszerű parancsokat hozunk létre, mint bemeneti nyelv (7.1.2), mely különös figyelmet fordít arra, hogy a rendszer véletlen elemeket is tartalmazhat, így a tesztelés során a véletlenszerűség ki-bekapcsolható, és a program determinisztikusan is tesztelhető.

### 7.1.2 Bemeneti nyelv

Itt félreértes elkerülésére a ***parancs <arg>*** formátum a ***parancs*** nevét, majd a **<...>**, az egyes argumentumokat jelöli. A következő parancsokat majd a konzolba egybeírva kell beírni, és az egyes argumentumok között a \_ szimbólumot használni. Egy példa pálya létrehozására:

***operationCreateMap10***

***operationCreateContainerPumpAt2\_2***

***operationCreatePlayerMechanicAt2\_2***

#### 7.1.2.1 Pályakezelő parancsok

***operationLoadMap <Filename>.txt***

**Leírás:** Betölt egy előre elkészített pálya állapotot egy megfelelő formátumú (.txt) fájlból.

**Argumentum:** Egy tetszőleges fájlnak a neve, ahonnan betölti a pályát.

***operationSaveMap <Filename>.txt***

**Leírás:** A jelenlegi játékpálya állapotát egy .txt formátumú fájlba menti.

**Argumentum:** Egy tetszőleges fájlnak a neve, ahol menti a pálya állapotot.

***operationCreateMap <Mapsize>***

**Leírás:** Egy üres, tetszőleges méretű négyzetrácsos pályát hoz létre.

**Argumentum:** Egy tetszőleges egész szám, ami a pálya méretéért felel.

***operationCreateContainer <Containertype> At <PosX>\_<PosY>***

**Leírás:** Egy pályaelem létrehozása egy tetszőleges koordinátán lévő pontnál.

**Argumentumok:**

1. **Containertype:** Valamilyen típusú tároló elem létrehozása.

**Opciók:**

1. **Cistern:** Egy ciszternát hoz létre. (Max 1/Map)

2. **MountainSpring:** Egy hegyi forrást hoz létre. (Max 1/Map)

3. **Pipe:** Egy csövet hoz létre.

4. **Pump:** Egy pumpát hoz létre.

2. **PosX:** A létrehozandó elem X koordinátája

3. **PosY:** A létrehozandó elem Y koordinátája

***operationDeleteContainerAt <PosX>\_<PosY>***

**Leírás:** Egy pályaelem törlése egy tetszőleges koordinátán lévő pontnál.

**Argumentumok:**

1. **PosX:** A törlendő elem X koordinátája

2. **PosY:** A törlendő elem Y koordinátája

***operationDeletePlayer <Player>***

**Leírás:** Egy konkrét játékos törlése a pályáról.

**Argumentumok:**

1. **Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***operationCreatePlayer <Playertype> At <PosX>\_<PosY>***

**Leírás:** Egy játékost helyez el egy tetszőleges koordinátánál lévő pályaelemen.

**Argumentumok:**

1. **Playertype:** Valamelyen típusú játékos létrehozása.

**Opciók:**

1. **Saboteur:** Egy szabotőr típusú játékos létrehozása.
2. **Mechanic:** Egy szerelő típusú játékos létrehozása.
2. **PosX:** A létrehozandó játékos X koordinátája.
3. **PosY:** A létrehozandó játékos Y koordinátája.

***operationConnectContainerAt <Pos1X>\_<Pos1Y> ToContainerAt <Pos2X>\_<Pos2Y>***

**Leírás:** Összeköt egy bizonyos pozícióban lévő elemet, egy másik pozícióban lévő elemmel, ha ez lehetséges.

**Argumentumok:**

1. **Pos1X:** Első tároló X koordinátája.
2. **Pos1Y:** Első tároló Y koordinátája.
3. **Pos2X:** Második tároló X koordinátája.
4. **Pos2Y:** Második tároló Y koordinátája.

**7.1.2.2 Játékos action parancsok*****player <Player> moveTo <Direction>***

**Leírás:** Egy konkrét játékost mozgat egy megadott irányba, ha ez lehetséges..

**Argumentum:**

1. **Player:** Egy szám, ami a konkrét játékosnak az azonosítója.
2. **Direction:** Az irány ami felé a játékost mozgatja.

**Opciók:**

1. **Left:** Balra mozgatja a játékost
2. **Right:** Jobbra mozgatja a játékost.
3. **Up:** Felfelé mozgatja a játékost.
4. **Down:** Lefelé mozgatja a játékost.

***player <Player> AdjustPumpTo <Direction>***

**Leírás:** Egy konkrét játékos átállítja, hogy a melyik irányba pumpáljon a pumpa, ha ez lehetséges.

**Argumentum:**

1. **Player:** Egy szám, ami a konkrét játékosnak az azonosítója.
2. **Direction:** Az irány ami felé a játékost átállítja a pumpát.

**Opciók:**

1. **Left:** Balra állítja át a pumpa vízpumpálásának irányát a játékos.
2. **Right:** Jobbra állítja át a pumpa vízpumpálásának irányát a játékos.
3. **Up:** Felfelé állítja át a pumpa vízpumpálásának irányát a játékos.
4. **Down:** Lefelé állítja át a pumpa vízpumpálásának irányát a játékos.

***player <Player> LeakPipe***

**Leírás:** Egy konkrét játékos elsüti a cső lékelés akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> AttachPipe***

**Leírás:** Egy konkrét játékos elsüti a cső elhelyezés akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> AttachPump***

**Leírás:** Egy konkrét játékos elsüti a pumpa elhelyezés akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> DetachPipe***

**Leírás:** Egy konkrét játékos elsüti a cső leszerelés akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> RepairPipe***

**Leírás:** Egy konkrét játékos elsüti a cső megjavítás akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> RepairPump***

**Leírás:** Egy konkrét játékos elsüti a pumpa javítás akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

***player <Player> MakePipeSlippery***

**Leírás:** Egy konkrét játékos elsüti a cső csúszósítás akcióját az adott pozíójában, ha ez lehetséges.

**Argumentum:**

**1. Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

### 7.1.2.3 Manuális parancsok

***manualCreateContainer <Containertype> AtCistern***

**Leírás:** Manuálisan létrehoz egy pályaelemet a ciszterna tárolójában.

**Argumentumok:**

**1. Containertype:** Valamilyen típusú tároló elem létrehozása a ciszterna tárolójában.

**Opciók:**

**1. Pipe:** Cső létrehozása a ciszterna tárolójában.

**2. Pump:** Pumpa létrehozása a ciszterna tárolójában. (Max 1/Cistern)

#### *manualCreateContainer <Containertype> AtPlayer <Player>*

**Leírás:** Manuálisan létrehoz egy pályaelemet egy játékos tárolójában.

**Argumentumok:**

**1. Containertype:** Valamilyen típusú tároló elem létrehozása egy játékos tárolójában.

**Opciók:**

**1. Pipe:** Cső létrehozása egy játékos tárolójában.

**2. Pump:** Pumpa létrehozása egy játékos tárolójában. (Max 1/Player)

**2. Player:** Egy szám, ami a konkrét játékosnak az azonosítója

#### *manualFlowWater*

**Leírás:** Manuálisan végig folyatja a vizet a jelenlegi pályán.

**Argumentumok:** -

#### *manualIncreaseTurnCountBy <Turns>*

**Leírás:** Manuálisan növeli a jelenlegi kör számát egy megadott értékkel (int).

**Argumentumok:**

**1. Turns:** Ennyivel növeli a jelenlegi kör számát.

#### *manualSetTurnCount <Turns>*

**Leírás:** Manuálisan beállítja a jelenlegi kör számát egy megadott értékre (int).

**Argumentumok:**

**1. Turns:** Ennyire állítja be a jelenlegi kör számát.

#### *manualDamageContainerAt <PosX> <PosY>*

**Leírás:** Manuálisan elrontja egy tárolóelem állapotát egy adott koordinátánál, ha ez lehetséges.

**Argumentumok:**

**1. PosX:** Az elrontandó tárolóelem (Cső/Pumpa) X koordinátája.

**2. PosY:** Az elrontandó tárolóelem (Cső/Pumpa) Y koordinátája.

#### *manualMakePipeSlipperyAt <PosX> <PosY>*

**Leírás:** Manuálisan csúszóssé teszi a cső állapotát egy adott koordinátánál, ha ez lehetséges.

**Argumentumok:**

**1. PosX:** Az csúszós állapotba tenni kívánt cső X koordinátája.

**2. PosY:** Az csúszós állapotba tenni kívánt cső Y koordinátája.

#### *manualTeleportPlayer<Player>To <PosX> <PosY>*

**Leírás:** Manuálisan áthelyez egy konkrét játékos egy konkrét koordinátára, ha ez lehetséges.

**Argumentumok:**

**1. PosX:** Egy tárolónak az X koordinátája, ahová a játékost elhelyezi.

**2. PosY:** Egy tárolónak az Y koordinátája, ahová a játékost elhelyezi.

### 7.1.2.4 Listázó parancsok

#### *listContainers*

**Leírás:** Kilistázza az összes pályaelem pozícióját, és típusát.

**Argumentumok:** -

#### *listDamagedContainers*

**Leírás:** Kilistázza az összes sérült pálya elem pozícióját, típusát és sérülés fajtáját.

**Argumentumok:** -

#### *listConnectedContainers*

**Leírás:** Kilistázza az összes pályaelem pozícióját, és az ezekhez csatlakozó összes pályaelem pozícióját.

**Argumentumok:** -

#### *listConnectedContainersAt <PosX>\_<PosY>*

**Leírás:** Kilistázza az összes pályaelem pozícióját, ami a keresett pálya elemhez csatlakozik.

**Argumentumok:**

1. **PosX:** Keresett pályaelem X koordinátája.

2. **PosY:** Keresett pályaelem Y koordinátája.

#### *listSlipperyPipes*

**Leírás:** Kilistázza az összes csúszós cső pozícióját.

**Argumentumok:** -

#### *listPlayersPos*

**Leírás:** Kilistázza az összes játékos jelenlegi pozícióját.

**Argumentumok:** -

#### *listPlayer<Player>Pos*

**Leírás:** Kilistázza egy konkrét játékos jelenlegi pozícióját.

**Argumentumok:**

1. **Player:** Egy szám, ami a konkrét játékosnak az azonosítója.

#### *listCurrentTurn*

**Leírás:** Kiírja a jelenlegi kör számát.

**Argumentumok:** -

#### *listPumpsDamageTurn*

**Leírás:** Kiírja az összes pumpának pozícióját, és azt hogy melyik körben fognak elromlani.

**Argumentumok:** -

#### *listPumpsDirection*

**Leírás:** Kilistázza az összes pumpának pozícióját, és azt hogy merre folyik bennük a víz.

**Argumentumok:** -

***listPumpAt <PosX>\_<PosY> DamageTurn***

**Leírás:** Kiírja, hogy melyik körben fog elromlani egy konkrét pumpa.

**Argumentumok:**

1. **PosX:** Keresett pumpa X koordinátája.
2. **PosY:** Keresett pumpa Y koordinátája.

***listPumpAt <PosX>\_<PosY> Direction***

**Leírás:** Kiírja egy konkrét pumpának a víz folyási irányát.

**Argumentumok:**

1. **PosX:** Keresett pumpa X koordinátája.
2. **PosY:** Keresett pumpa Y koordinátája.

**7.1.2.5 Véletlenparancsok*****randomPumpBreakdownTurnOff***

**Leírás:** Kikapcsolja azt, hogy a pumpák egy véletlen körben elromoljanak

**Argumentumok:** -

***randomPumpBreakdownTurnOn***

**Leírás:** Bekapcsolja azt, hogy a pumpák egy véletlen körben elromoljanak

**Argumentumok:** -

**7.1.3 Kimeneti nyelv****7.1.3.1 Pályakezelő parancsok*****operationLoadMap <Filename>.txt***

**Sikerességi kimenet:** Map <Filename>.txt successfully loaded

**Sikertelen futtatás kimenet:** Map <Filename>.txt not found

**Mi okozhat hibát:** Ha a megnyitandó fájl nem létezik.

***operationSaveMap <Filename>.txt***

**Sikerességi kimenet:** Map successfully save into <Filename>.txt

**Sikertelen futtatás kimenet:** Map cannot be saved into <Filename>.txt, because a file with the name <Filename>.txt already exists.

**Mi okozhat hibát:** Ha a fájl ahová menteni szeretnénk, már létezik a megadott névvel.

***operationCreateMap <Mapsiz>***

**Sikerességi kimenet:** Map with size <Mapsiz> was successfully created.

**Sikertelen futtatás kimenet:** <Mapsiz> is not an integer

**Mi okozhat hibát:** Ha nem egész számot adunk meg

***operationCreateContainer <Containertype> At <PosX>\_<PosY>***

**Sikeres futtatás kimenet:** <Containertype> was successfully created at X: <PosX>, Y: <PosY>

**Sikertelen futtatás kimenet:**

1. <Containertype> cannot be created at X: <PosX>, Y: <PosY>, because at this position a container already exists.
2. Cistern cannot be created at X: <PosX>, Y: <PosY>, because the maximum number of Cisterns is 1.
3. MountainSpring cannot be created at X: <PosX>, Y: <PosY>, because the maximum number of MountainSprings is 1.
4. <Containertype> is not a valid value
5. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
6. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha a felhasználó egy olyan pozícióban akar létrehozni egy pályaelemet, ahol már van pályaelem.
2. Egynél több ciszterna nem lehet a pályán
3. Egynél több hegyi forrás nem lehet a pályán
4. Rossz érték megadása container típusnak
5. Hibás érték megadása koordinátáknak

***operationDeleteContainerAt <PosX>\_<PosY>***

**Sikeres futtatás kimenet:** Container was successfully deleted at X: <PosX>, Y: <PosY>

**Sikertelen futtatás kimenet:**

1. Container cannot be deleted at X: <PosX>, Y: <PosY>, because there is nothing there
2. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
3. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. A pozíció ahol a pálya elemet törölni szeretnénk, már üres (nincs ott pályaelem)
2. Hibás érték megadása koordinátáknak

***operationDeletePlayer <Player>***

**Sikeres futtatás kimenet:** Player <Player> was successfully deleted

**Sikertelen futtatás kimenet:** Player <Player> cannot be deleted, because there is no such Player, with the ID: <Player>

**Mi okozhat hibát:** Nincs olyan számú játékos, akit törölni szeretnénk.

***operationCreatePlayer <Playertype> At <PosX>\_<PosY>***

**Sikeres futtatás kimenet:** <Playertype> was successfully created at X: <PosX>, Y: <PosY>

**Sikertelen futtatás kimenet:**

1. <Playertype> cannot be created at X: <PosX>, Y: <PosY>, because there is no container at X: <PosX>, Y: <PosY> to stand on.
2. <Playertype> is not a valid value
3. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
4. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Nincs pályaelem a megadott pozícióban, ahol a játékos létre akarjuk hozni
2. Rossz érték megadása játékos típusnak
3. Hibás érték megadása koordinátáknak

***operationConnectContainerAt <Pos1X>\_<Pos1Y>ToContainerAt <Pos2X>\_<Pos2Y>***

**Sikeres futtatás kimenet:** Container at X: <Pos1X>, Y: <Pos1Y> was successfully connected to container at X: <Pos2X>, Y: <Pos2Y>

**Sikertelen futtatás kimenet:**

1. Container at X: <Pos1X>, Y: <Pos1Y> cannot be connected to Container at X: <Pos2X>, Y: <Pos2Y>, because they not neighbors.
2. Container at X: <Pos1X>, Y: <Pos1Y> cannot be connected to Container at X: <Pos2X>, Y: <Pos2Y>, because they are the same type of containers.
3. The value of <Pos1X> is too big/small or not an integer, the maximum value is Mapsize-1
4. The value of <Pos1Y> is too big/small or not an integer, the maximum value is Mapsize-1
5. The value of <Pos2X> is too big/small or not an integer, the maximum value is Mapsize-1
6. The value of <Pos2Y> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Olyan elemeket próbálunk csatlakoztatni egymáshoz, amik nem szomszédos cellában vannak egymáshoz képest.
2. Ugyanolyan típusú elemet próbálunk csatlakoztatni egymáshoz
3. Hibás érték megadása koordinátáknak

### 7.1.3.2 Játékos action parancsok

#### *player <Player> moveTo <Direction>*

**Sikeres futtatás kimenet:** Player <Player> successfully moved <Direction>

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot move <Direction>, because a container doesn't exist in this direction
2. Player <Player> cannot move <Direction>, because the container this direction is occupied
3. There is no such player as Player <Player>
4. <Direction> is not a valid value

**Mi okozhat hibát:**

1. A játékos amerre szeretne lépni, abban az irányban nincsen pálya elem.
2. A játékos amerre szeretne lépni, az abban az irányban lévő pálya elem már foglalt.
3. Nincs ilyen számú játékos
4. Hibás érték megadása iránynak

#### *player <Player> AdjustPumpTo <Direction>*

**Sikeres futtatás kimenet:** Player <Player> successfully adjust pump output to <Direction>

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot adjust pump output to <Direction>, because the player is not at a pump
2. There is no such player as Player <Player>
3. <Direction> is not a valid value

**Mi okozhat hibát:**

1. Ha a játékos az akcióját nem egy pumpánál süti el
2. Nincs ilyen számú játékos
3. Hibás érték megadása iránynak

#### *player <Player> LeakPipe*

**Sikeres futtatás kimenet:** Player <Player> successfully leaked the pipe at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot leak a pipe, because they are not standing on a pipe
2. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ha a játékos nem csövön süti el az akcióját.
2. Nincs ilyen számú játékos

***player <Player> AttachPipe***

**Sikeres futtatás kimenet:** Player <Player> successfully attached a pipe at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot attach a pipe at their position, because the container they are standing at is the same type
2. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ugyanolyan típusú elemet próbálunk csatlakoztatni egymáshoz
2. Nincs ilyen számú játékos

***player <Player> AttachPump***

**Sikeres futtatás kimenet:** Player <Player> successfully attached a pump at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot attach a pump at their position, because the container they are standing at is the same type
2. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ugyanolyan típusú elemet próbálunk csatlakoztatni egymáshoz
2. Nincs ilyen számú játékos

***player <Player> DetachPipe***

**Sikeres futtatás kimenet:** Player <Player> successfully detached a pipe at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot detach a pipe at their position, because they are not standing on a pipe
2. There is no such player as Player <Player>

**Mi okozhat hibát:** Ha a játékos nem csövön áll***player <Player> RepairPipe***

**Sikeres futtatás kimenet:** Player <Player> successfully repaired a pipe at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot repair a pipe at their position, because they are not standing on a pipe
2. Player <Player> cannot repair a pipe at their position, because the pipe they are standing on is not leaked
3. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ha a játékos nem csövön áll
2. Ha a játékos nem elromlott csövet próbál megjavítani
3. Nincs ilyen számú játékos

***player <Player>RepairPump***

**Sikeres futtatás kimenet:** Player <Player> successfully repaired a pump at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot repair a pump at their position, because they are not standing on a pump
2. Player <Player> cannot repair a pump at their position, because the pump they are standing on is not damaged
3. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ha a játékos nem pumpánál áll
2. Ha a játékos nem elromlott pumpát próbál megjavítani
3. Nincs ilyen számú játékos

***player <Player> MakePipeSlippery***

**Sikeres futtatás kimenet:** Player <Player> successfully made a pipe slippery at their position

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot make a pipe slippery at their position, because they are not standing on a pipe
2. Player <Player> cannot make a pipe slippery at their position, because the action is on cooldown
3. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ha a játékos nem csövön áll
2. A játékos akciója nem áll készen
3. Nincs ilyen számú játékos

**7.1.3.3 Manuális parancsok*****manualCreateContainer <Containertype> AtCistern***

**Sikeres futtatás kimenet:** <Containertype> was successfully created in cistern's inventory

**Sikertelen futtatás kimenet:**

1. Pump cannot be created in cistern's inventory, because the inventory is full
2. <Containertype> cannot be created in cistern's inventory, because there is no cistern on the map
3. <Containertype> is not a valid value

**Mi okozhat hibát:**

1. Nem lehet 1-nél több pumpa egyszerre a ciszternában
2. Ha nincs ciszterna a pályán, akkor nem lehet ott létrehozni elemeket
3. Hibás érték megadása

***manualCreateContainer <Containertype> AtPlayer <Player>***

**Sikeres futtatás kimenet:** <Containertype> was successfully created in player <Player>'s inventory

**Sikertelen futtatás kimenet:**

1. Pump cannot be created in player <Player>'s inventory, because the inventory is full
2. <Containertype> is not a valid value
3. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Egy játékosnál csak egy pumpa lehet egyszerre
2. Hibás érték megadása
3. Nincs ilyen számú játékos

***manualFlowWater***

**Sikeres futtatás kimenet:** The water is flowing in the containers

**Sikertelen futtatás kimenet:** The water cannot flow, because there is no MountainSpring on the map

**Mi okozhat hibát:** Ha nincs hegyi forrás a pályán, akkor a víz nem tud elkezdeni folyni

***manualIncreaseTurnCountBy <Turns>***

**Sikeres futtatás kimenet:** Current turn count successfully increased by <Turns>

**Sikertelen futtatás kimenet:**

1. Current turn count cannot be increased by <Turns>, because there is no loaded/created map
2. <Turns> is not an integer

**Mi okozhat hibát:**

1. Ha nincs betöltött/létrehozott pálya, aminek növelhetnénk a körének számát
2. Nem egész szám megadása

***manualSetTurnCount<Turns>***

**Sikeres futtatás kimenet:** Current turn count successfully set to <Turns>

**Sikertelen futtatás kimenet:**

1. Current turn count cannot be set to <Turns>, because there is no loaded/created map
2. <Turns> is not an integer

**Mi okozhat hibát:**

1. Ha nincs betöltött/létrehozott pálya, aminek beállíthatnánk a körének számát
2. Nem egész szám megadása

***manualDamageContainerAt <PosX> <PosY>***

**Sikeres futtatás kimenet:** The container at X:<PosX>, Y:<PosY> was damaged successfully

**Sikertelen futtatás kimenet:**

1. The container at X:<PosX>, Y:<PosY> cannot be damaged, because there is no container at X:<PosX>, Y:<PosY>
2. The container at X:<PosX>, Y:<PosY> cannot be damaged, because the container is already damaged
3. The container at X:<PosX>, Y:<PosY> cannot be damaged, because the container is not a Pipe/Pump
4. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
5. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha nincs az adott koordinátál nincs létező pályaelem
2. Ha nincs az adott koordinátál már sérült a pályaelem
3. Ha nincs az adott koordinátál nem cső/pumpa van
4. Ha hibás a koordinátáknak a megadott értéke

***manualMakePipeSlipperyAt <PosX> <PosY>***

**Sikeres futtatás kimenet:** The container at X:<PosX>, Y:<PosY> was successfully made slippery

**Sikertelen futtatás kimenet:**

1. The container at X:<PosX>, Y:<PosY> cannot be made slippery, because there is no container at X:<PosX>, Y:<PosY>
2. The container at X:<PosX>, Y:<PosY> cannot be made slippery, because the container is already slippery
3. The container at X:<PosX>, Y:<PosY> cannot be made slippery, because the container is not a Pipe
4. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
5. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha nincs az adott koordinátál nincs létező pályaelem
2. Ha nincs az adott koordinátál már csúszós a pályaelem
3. Ha nincs az adott koordinátál nem cső van
4. Ha hibás a koordinátáknak a megadott értéke

### *manualTeleportPlayer<Player>To <PosX> <PosY>*

**Sikeres futtatás kimenet:** Player <Player> was successfully teleported to X:<PosX>, Y:<PosY>

**Sikertelen futtatás kimenet:**

1. Player <Player> cannot be teleported to X:<PosX>, Y:<PosY>, because the target container doesn't exist at X:<PosX>, Y:<PosY>
2. Player <Player> cannot be teleported to X:<PosX>, Y:<PosY>, because the target container is occupied
3. Player <Player> cannot be teleported to X:<PosX>, Y:<PosY>, because the target container is not steppable
4. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
5. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha a pályaelem nem létezik
2. Ha a pályaelem foglalt
3. Ha a pályaelemre nem lehet rálépni
4. Ha hibás a koordinátáknak a megadott értéke

### 7.1.3.4 Listázó parancsok

#### *listContainers*

**Sikeres futtatás kimenet:** pl.

There is cistern at X:1, Y:0

There is a pipe at X:1, Y:1

There is a pump at X:1, Y:2

**Sikertelen futtatás kimenet:** There are no containers on the map

**Mi okozhat hibát:** Ha nincs pálya elem a pályán, akkor nincs mit kilistázni

#### *listDamagedContainers*

**Sikeres futtatás kimenet:** pl.

There is a leaked pipe at X:1, Y:1

There is a damaged pump at X:1, Y:2

**Sikertelen futtatás kimenet:**

1. There are no containers on the map
2. There are no damaged containers on the map

**Mi okozhat hibát:**

1. Ha nincs pálya elem a pályán, akkor nincs mit kilistázni
2. Ha nincs sérült pálya elem a pályán, akkor nincs mit kilistázni

***listConnectedContainers*****Sikeres futtatás kimenet:** pl

The container at X:1, Y:1 is connected to the container at X:1,Y:2

The container at X:1, Y:2 is connected to the container at X:1,Y:3

**Sikertelen futtatás kimenet:**

1. There are no containers on the map
2. There are no connected containers on the map

**Mi okozhat hibát:**

1. Ha nincs pálya elem a pályán, akkor nincs mit kilistázni
2. Ha nincs összekötött pálya elem a pályán, akkor nincs mit kilistázni

***listConnectedContainersAt <PosX>\_<PosY>*****Sikeres futtatás kimenet:** pl

The container at X:0, Y:1 is connected to the container at X:1, Y:1

The container at X:1, Y:0 is connected to the container at X:1, Y:1

The container at X:1, Y:2 is connected to the container at X:1, Y:1

The container at X:2, Y:2 is connected to the container at X:1, Y:1

**Sikertelen futtatás kimenet:**

1. There is no container at X:<PosX>, Y:<PosY>
2. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
3. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha nincs pálya elem a megadott koordinátáknál
2. Ha hibás a koordinátáknak a megadott értéke

***listSlipperyPipes*****Sikeres futtatás kimenet:** pl

There is a slippery pipe at X:1, Y:1

There is a slippery pipe at X:1, Y:3

**Sikertelen futtatás kimenet:**

1. There are no containers on the map
2. There are no slippery pipes on the map

**Mi okozhat hibát:**

1. Ha nincs pálya elem a pályán, akkor nincs mit kilistázni
2. Ha nincs csúszós cső a pályán, akkor nincs mit kilistázni

***listPlayersPos*****Sikeres futtatás kimenet:** pl.

Player1 is at X:1, Y:1

Player2 is at X:2, Y:1

Player3 is at X:1, Y:3

**Sikertelen futtatás kimenet:** There are no players on the map**Mi okozhat hibát:** Ha nincs játékos a pályán, akkor nincs kit kilistázni

***listPlayer<Player>Pos*****Sikeres futtatás kimenet:** pl.

Player&lt;Player&gt; is at X:1, Y:1

**Sikertelen futtatás kimenet:**

1. There are no players on the map
2. There is no such player as Player <Player>

**Mi okozhat hibát:**

1. Ha nincs játékos a pályán, akkor nincs mit kilistázni
2. Ha nincs ilyen számú játékos

***listCurrentTurn*****Sikeres futtatás kimenet:** pl.

The current turn is: 10.

**Sikertelen futtatás kimenet:** There is no loaded map to list the current turn on.**Mi okozhat hibát:** Ha nincs pálya, akkor nincs mit kilistázni***listPumpsDamageTurn*****Sikeres futtatás kimenet:** pl.

The pump at X:1, Y:1 will be damaged in 3 turns

The pump at X:1, Y:3 will be damaged in 6 turns

**Sikertelen futtatás kimenet:**

1. There are no pumps on the map
2. There are no undamaged pumps on the map

**Mi okozhat hibát:**

1. Ha nincs pumpa a pályán, akkor nincs mit kilistázni
2. Ha nincs ép pumpa a pályán, akkor nincs mit kilistázni

***listPumpsDirection*****Sikeres futtatás kimenet:** pl.

The pump at X:1, Y:1 is pumping to left

The pump at X:1, Y:3 is pumping to right

**Sikertelen futtatás kimenet:** There are no pumps on the map**Mi okozhat hibát:** Ha nincs pumpa a pályán, akkor nincs mit kilistázni

***listPumpAt <PosX>\_<PosY> DamageTurn*****Sikeres futtatás kimenet:** pl.

The pump at X:&lt;PosX&gt;, Y:&lt;PosY&gt; will be damaged in 3 turns

**Sikertelen futtatás kimenet:**

1. There are no pumps on the map
2. There is no pump at the target location
3. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
4. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha nincs pumpa a pályán, akkor nincs mit kilistázni
2. Ha nincs pumpa a megadott koordinátknál, akkor nincs mit kilistázni
3. Hibásak a megadott koordináták

***listPumpAt <PosX>\_<PosY> Direction*****Sikeres futtatás kimenet:** pl.

The pump at X:&lt;PosX&gt;, Y:&lt;PosY&gt; is pumping to left

**Sikertelen futtatás kimenet:**

1. There are no pumps on the map
2. There is no pump at the target location
3. The value of <PosX> is too big/small or not an integer, the maximum value is Mapsize-1
4. The value of <PosY> is too big/small or not an integer, the maximum value is Mapsize-1

**Mi okozhat hibát:**

1. Ha nincs pumpa a pályán, akkor nincs mit kilistázni
2. Ha nincs pumpa a megadott koordinátknál, akkor nincs mit kilistázni
3. Hibásak a megadott koordináták

**7.1.3.5 Véletlenparancsok*****randomPumpBreakdownTurnOff*****Sikeres futtatás kimenet:** pl.

Pumps won't be damaged in a random turn

**Sikertelen futtatás kimenet:** -**Mi okozhat hibát:** -

***randomPumpBreakdownTurnOn*****Sikeres futtatás kimenet:** pl.

Pumps will be damaged in a random turn

**Sikertelen futtatás kimenet:** -**Mi okozhat hibát:** -

## 7.2 Összes részletes use-case

<b>Use-case neve</b>	<i>operationLoadMap &lt;Filename&gt;.txt</i>
<b>Rövid leírás</b>	Betölthetünk egy tetszőleges, előre elkészített pályaállapotot egy .txt formátumú fájlból.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program betölti a pályaállapotot a parancs argumentumában megadott fájlból.

<b>Use-case neve</b>	<i>operationSaveMap &lt;Filename&gt;.txt</i>
<b>Rövid leírás</b>	A jelenlegi pályaállapotot tudjuk elmenteni egy .txt formátumú fájlba
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program elmenti a jelenlegi pályaállapotot a parancs argumentumában megadott .txt formátumú fájlba.

<b>Use-case neve</b>	<i>operationCreateMap &lt;Mapsizer&gt;</i>
<b>Rövid leírás</b>	Létrehozhatunk egy üres, tetszőleges méretű négyzetrácsos pályát.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program létrehoz egy, a parancs argumentumában megadott méretű, négyzetrácsos pályát.

<b>Use-case neve</b>	<i>operationCreateContainer &lt;Containertype&gt; At &lt;PosX&gt;_&lt;PosY&gt;</i>
<b>Rövid leírás</b>	Segítségével létrehozhatunk egy új, tetszőleges típusú pályelemet egy tetszőleges pozícióban.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program létrehoz egy új, argumentumban megadott típusú pályaelemet az argumentumban megadott pozícióban, amennyiben a jelenlegi pályára érvényes pozíciót adtunk meg.

<b>Use-case neve</b>	<i>operationDeleteContainerAt &lt;PosX&gt;_&lt;PosY&gt;</i>
<b>Rövid leírás</b>	Ezzel a prancsal kitörölhetünk egy tetszőleges pozícióból egy pályaelemet.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program törli az argumentumban megadott pozícióban lévő pályaelemet.

<b>Use-case neve</b>	<i>operationDeletePlayer &lt;Player&gt;</i>
<b>Rövid leírás</b>	Segítségével egy játékost törölhetünk a pályáról.
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program törli az argumentumban megadott azonosítójú játékost a pályáról.

<b>Use-case neve</b>	<i>operationCreatePlayer &lt;Playertype&gt; At &lt;PosX&gt;_&lt;PosY&gt;</i>
<b>Rövid leírás</b>	Hozzáadhatunk egy új, tetszőleges típusú játékost a pályához egy tetszőleges, jelenlegi pályán érvényes pozícióba.
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program hozzáad egy argumentumban megadott típusú játékost az argumentumban megadott pozícióba, amennyiben érvényes pozíciót adtunk meg.

<b>Use-case neve</b>	<i>operationConnectContainerAt &lt;Pos1X&gt;_&lt;Pos1Y&gt; ToContainerAt &lt;Pos2X&gt;_&lt;Pos2Y&gt;</i>
<b>Rövid leírás</b>	Összeköt két, különböző pozícióban lévő pályaelemet, amennyiben ez lehetséges.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program összeköti a két, argumentumban megadott pozícióban lévő pályaelemet, amennyiben érvényes pozíciókat adtunk meg, és ténylegesen találhatók ott elemek.

<b>Use-case neve</b>	<i>player &lt;Player&gt; moveTo &lt;Direction&gt;</i>
<b>Rövid leírás</b>	Ha lehetséges, elmozgat egy tetszőleges játékost egy irányba.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Ha létező játékost adtunk meg az argumentumban, és lehetséges az adott irányban történő mozgás, a program mozgatja a játékost a megadott irányba.

<b>Use-case neve</b>	<i>player &lt;Player&gt; AdjustPumpTo &lt;Direction&gt;</i>
<b>Rövid leírás</b>	Egy játékos átállítja egy pumpánál, hogy melyik irányba pumpáljon.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos átállítja a pumpát a megadott irányba, ha jelenleg pumpán áll és az átállítás az adott irányba lehetséges.

<b>Use-case neve</b>	<i>player &lt;Player&gt; LeakPipe</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos kielsüti a cső lyukasztás akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott játékos, ha csövön áll, és az a cső nincsen kilyukasztva, akkor kilyukasztja azt.

<b>Use-case neve</b>	<i>player &lt;Player&gt; AttachPipe</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos elsüti a cső elhelyezés akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos elhelyezi az általa hurcolt csövet a jelenlegi pozíójában, ha erre van lehetőség.

<b>Use-case neve</b>	<i>player &lt;Player&gt; AttachPump</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos elsüti a pumpa elhelyezés akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos elhelyezi az általa cipelt pumpát a jelenlegi pozíójában, ha erre van lehetőség.

<b>Use-case neve</b>	<i>player &lt;Player&gt; DetachPipe</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos elsüti a cső leszerelés akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos leszereli a jelenlegi pozíójában lévő csövet, ha csövön áll éppen.

<b>Use-case neve</b>	<i>player &lt;Player&gt; RepairPipe</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos elszüti a cső megjavítás akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos megjavítja a jelenlegi pozíójában lévő csövet, ha csövön áll és az ki van lyukasztva..

<b>Use-case neve</b>	<i>player &lt;Player&gt; RepairPump</i>
<b>Rövid leírás</b>	Egy tetszőleges játékos elszüti a pumpa megjavítás akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos megjavítja a jelenlegi pozíójában lévő pumpát, ha pumpán áll és az el van romolva.

<b>Use-case neve</b>	<i>player &lt;Player&gt; MakePipeSlippery</i>
<b>Rövid leírás</b>	Egy játékos elszüti a cső csúszósítás akcióját a jelenlegi pozíójában.
<b>Aktorok</b>	Player
<b>Forgatókönyv</b>	Az argumentumban megadott azonosítójú játékos csúszóssá teszi a pályaelemet, amelyen áll, ha az még nem csúszik.

<b>Use-case neve</b>	<i>manualCreateContainer &lt;Containertype&gt; AtCistern</i>
<b>Rövid leírás</b>	Manuálisan létrehoz egy tetszőleges típusú pályaelemet a ciszterna tárolójában.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program létrehoz egy az argumentumban megadott típusú pályaelemet a ciszterna tárolójában.

<b>Use-case neve</b>	<i>manualCreateContainer &lt;Containertype&gt; AtPlayer &lt;Player&gt;</i>
<b>Rövid leírás</b>	Manuálisan létrehoz egy tetszőleges típusú pályaelemet a kiválasztott játékos tárolójában,
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program létrehoz egy az argumentumban megadott típusú pályaelemet a szintén argumentumban megadott azonosítójú játékos tárolójában.

<b>Use-case neve</b>	<i>manualFlowWater</i>
<b>Rövid leírás</b>	Manuálisan végigfolyatja a vizet a pályán.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program végig folyatja a vizet a pályán, figyelembe véve a pmpák állását és a csövek állapotát.

<b>Use-case neve</b>	<i>manualIncreaseTurnCountBy &lt;Turns&gt;</i>
<b>Rövid leírás</b>	Megnöveli a kör számát egy tetszőleges értékkel.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program megnöveli a jelenlegi kör számát az argumentumban megadott értékkel.

<b>Use-case neve</b>	<i>manualSetTurnCount &lt;Turns&gt;</i>
<b>Rövid leírás</b>	Beállítja a kör számát a megadott értékre.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program beállítja a jelenlegi kör számát az argumentumban megadott értékre.

<b>Use-case neve</b>	<i>manualDamageContainerAt &lt;PosX&gt; &lt;PosY&gt;</i>
<b>Rövid leírás</b>	Elrontja a megadott pozícióban lévő pályaelem állapotát.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program elrontja az argumentumban megadott pozícióban lévő pályaelem állapotát, ha érvényes pozíciót adtunk meg és létezik ott működőképes pályaelem.

<b>Use-case neve</b>	<i>manualMakePipeSlipperyAt &lt;PosX&gt; &lt;PosY&gt;</i>
<b>Rövid leírás</b>	Csúszóssé teszi a megadott pozícióban lévő pályaelemet.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program csúzóssá teszi az argumentumban megadott pozícióban lévő pályaelemet, ha érvényes pozíciót adtunk meg és létezik ott nem csúszós pályaelem.

<b>Use-case neve</b>	<i>manualTeleportPlayer&lt;Player&gt;To &lt;PosX&gt; &lt;PosY&gt;</i>
<b>Rövid leírás</b>	Áthelyez egy tetszőleges játékost egy tetszőleges, pályán érvényes pozícióba.
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program áthelyezi az argumentumban megadott azonosítójú játékost a szintén argumentumban megadott pozícióba, ha érvényes koordinátákat adtunk meg.

<b>Use-case neve</b>	<i>listContainers</i>
<b>Rövid leírás</b>	Kilistázza a pályán lévő összes pályaelem típusát és pozícióját.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra a pályán jelenleg megtalálható pályaelemek típusát és pozícióját.

<b>Use-case neve</b>	<i>listDamagedContainers</i>
<b>Rövid leírás</b>	Kilistázza a pályán lévő összes sérült pályaelem típusát és pozícióját, illetve s sérülés fajtáját.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra a pályán jelenleg megtalálható sérült pályaelemek típusát és pozícióját, illetve, hogy milyen típusú a sérülésük.

<b>Use-case neve</b>	<i>listConnectedContainers</i>
<b>Rövid leírás</b>	Kilistázza az összes pályaelem pozícióját, és minden pályaelemhez kilistázza a hozzá kapcsolódó összes pályaelem pozícióját.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra a pályán jelenleg megtalálható pályaelemek pozícióját, és minden pályaelemhez kilistázza a hozzá kapcsolódó összes pályaelem pozícióját.

<b>Use-case neve</b>	<i>listConnectedContainersAt &lt;PosX&gt;_&lt;PosY&gt;</i>
<b>Rövid leírás</b>	Kilistázza a megadott pozícióban lévő pályaelemhez csatlakozó pályaelemek pozícióját.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra az argumentumban megadott pozíciójú pályaelemhez kapcsolódó pályaelemek pozícióját, ha érvényes pozíciót adtunk meg.

<b>Use-case neve</b>	<i>listSlipperyPipes</i>
<b>Rövid leírás</b>	Kilistázza a csúzós csövek pozícióját.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra a pályán lévő összes csúzós állapotú cső pozícióját.

<b>Use-case neve</b>	<i>listPlayersPos</i>
<b>Rövid leírás</b>	Kilistázza az összes játékos jelenlegi pozíóját.
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program kilistázza a standard outputra a pályán lévő összes játékos jelenlegi pozíóját.

<b>Use-case neve</b>	<i>listPlayer&lt;Player&gt;Pos</i>
<b>Rövid leírás</b>	Kiírja a megadott játékos jelenlegi pozíóját.
<b>Aktorok</b>	Controller, Player
<b>Forgatókönyv</b>	A program kiírja a standard outputra az argumentumban megadott azonosítójú játékos pozíóját.

<b>Use-case neve</b>	<i>listCurrentTurn</i>
<b>Rövid leírás</b>	Kiírja a kör számát.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kiírja a standard outputra a jelenlegi kör számát.

<b>Use-case neve</b>	<i>listPumpsDamageTurn</i>
<b>Rövid leírás</b>	Kilistázza az összes pumpa pozíóját, és hogy hányadik körben fognak elromlani.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra az összes pumpa pozíóját, és hogy az egyes pumpák hányadik körben fognak elromlani.

<b>Use-case neve</b>	<i>listPumpsDirection</i>
<b>Rövid leírás</b>	Kilistázza az összes pumpának pozícióját, és azt hogy merre folyik bennük a víz.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kilistázza a standard outputra az összes pumpa pozíóját, és hogy az egyes pumpáknál melyik irányba folyik a víz.

<b>Use-case neve</b>	<i>listPumpAt &lt;PosX&gt;_&lt;PosY&gt; DamageTurn</i>
<b>Rövid leírás</b>	Kiírja, hogy a megadott pozícióban lévő pumpa hányadik körben fog elromlani.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kiírja standard outputra, hogy az argumentumban megadott pozíciójú pumpa hányadik körben fog elromlani.

<b>Use-case neve</b>	<i>listPumpAt &lt;PosX&gt;_&lt;PosY&gt; Direction</i>
<b>Rövid leírás</b>	Kiírja, hogy a megadott pozícióban lévő pumpánál melyik irányba folyik a víz.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A program kiírja standard outputra, hogy az argumentumban megadott pozíciójú pumpánál melyik irányba folyik a víz.

<b>Use-case neve</b>	<i>randomPumpBreakdownTurnOff</i>
<b>Rövid leírás</b>	Kikapcsolja a pumpák véletlenszerű elromlását.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A programban kikapcsolásra kerül a pumpák véletlen körben történő elromlása.

<b>Use-case neve</b>	<i>randomPumpBreakdownTurnOn</i>
<b>Rövid leírás</b>	Bekapcsolja a pumpák véletlenszerű elromlását.
<b>Aktorok</b>	Controller
<b>Forgatókönyv</b>	A programban bekapcsolásra kerül a pumpák véletlen körben történő elromlása.

### 7.3 Tesztelési terv

<b>Teszt-eset neve</b>	Random Pump Break
<b>Rövid leírás</b>	A pumpa eltörésének tesztelése, ebben az esetben 100% valószínűséggel eltörik a pumpa.
<b>Teszt célja</b>	Vizsgáljuk, hogy helyesen működsik-e a RandomPumpBreak() metódus

<b>Teszt-eset neve</b>	Pump repair
<b>Rövid leírás</b>	A pumpa megjavításának tesztelése.
<b>Teszt célja</b>	Azt vizsgáljuk, hogy a Player meg tudja-e javítani az elromlott pumpákat. Itt elsősorban a Player és a Pump osztályt teszteljük

<b>Teszt-eset neve</b>	SetInputTest
<b>Rövid leírás</b>	Pumpa bemeneti csövének beállítását teszteljük.
<b>Teszt célja</b>	Arra vagyunk kíváncsiak, hogy helyesen működik-e a SetInput()

<b>Teszt-eset neve</b>	Pipe repair
<b>Rövid leírás</b>	A csövek megjavításának tesztelése.
<b>Teszt célja</b>	Azt vizsgáljuk, hogy a Player meg tudja-e javítani a kilyukadt csöveget. Tesztelt osztályok: Player, Pipe

<b>Teszt-eset neve</b>	Leak pipe
<b>Rövid leírás</b>	A csövek kilyukasztásának tesztelése
<b>Teszt célja</b>	A teszt célja a Player csőlyukasztásának tesztelése. Tesztelt osztályok: Player, Pipe

<b>Teszt-eset neve</b>	SetOutputTest
<b>Rövid leírás</b>	Pumpa kimeneti csövének beállítását teszteljük.
<b>Teszt célja</b>	Arra vagyunk kíváncsiak, hogy helyesen működik-e a SetOutput()

<b>Teszt-eset neve</b>	Attach Pipe Success
<b>Rövid leírás</b>	A pumpához történő sikeres csőcsatlakoztatást teszteli.
<b>Teszt célja</b>	Cél, hogy a sikeres csőcsatlakoztatást vizsgáljuk. Ehhez létrehozunk egy pumpát, adunk a Playernek egy csövet és rácsatlakoztatjuk a pumpára. Tesztelt osztályok: Player, Pump, Pipe, Container

<b>Teszt-eset neve</b>	PipeStickynessTest
<b>Rövid leírás</b>	Egy cső ragadósságát vizsgáljuk, ami azt jelenti, hogy megnézzük, hogy aki rálép ragadós állapotban, az valóban nem tud-e tovább mozogni egy darabig
<b>Teszt célja</b>	PipeSticky() metódus tesztje

<b>Teszt-eset neve</b>	Attach Pipe Fail
<b>Rövid leírás</b>	A pumpához történő sikertelen csőcsatlakoztatást teszteli.
<b>Teszt célja</b>	A teszteset célja a sikertelen csatlakoztatás vizsgálata. Ehhez létrehozunk egy pumpát, feltöljük csövekkel amíg eléri a kapacitását, majd adunk a Playernek egy csövet, és megpróbáljuk hozzácsatolni a pumpához. Tesztelt osztályok: Player, Pump, Pipe, Container

<b>Teszt-eset neve</b>	Attach Pump Success
<b>Rövid leírás</b>	A pumpa sikeres lerakását tesztelő teszteset.
<b>Teszt célja</b>	A teszteset célja a sikeres pumpalerakás vizsgálata. Ehhez adunk egy Playernek egy pumpát, majd megpróbáljuk lerakni a jelenlegi pozíójába. Tesztelt osztályok: Player, Pump, Container

<b>Teszt-eset neve</b>	Attach Pump Fail
<b>Rövid leírás</b>	A pumpa sikertelen lerakását tesztelő teszteset.
<b>Teszt célja</b>	A teszteset célja a sikertelen pumpalerakás vizsgálata. Ehhez adunk egy Playernek egy pumpát, létrehozunk egy másikat a pályán, majd megpróbáljuk lerakni a oda, ahol a másik pumpa van. Tesztelt osztályok: Player, Pump, Container

<b>Teszt-eset neve</b>	Detach Pipe Success
<b>Rövid leírás</b>	Egy pumpától való cső sikeres lecsatlakoztatását vizsgá teszteset.
<b>Teszt célja</b>	Cél, hogy a sikeres lecsatlakoztatást teszteljük. Ehhez létrehozunk egy pumpát, rakunk rá egy csövet, majd megpróbáljuk leszedni róla. Tesztelt osztályok: Player, Pump, Pipe, Container

<b>Teszt-eset neve</b>	Remove Pump Test
<b>Rövid leírás</b>	A csövek végéhez kapcsolódó pumpa eltávolításának sikerességét teszteljük
<b>Teszt célja</b>	Vizsgáljuk, hogy jól működik-e a pumpa-leválasztás a csövekről

<b>Teszt-eset neve</b>	Pipe puncturing Test
<b>Rövid leírás</b>	Cső lyukasztás sikerességének tesztelése
<b>Teszt célja</b>	Meg kell vizsgálnunk, hogy egy csőlyukasztás eseménykor valóban megtörténnek e az elvárt változások a programban.

<b>Teszt-eset neve</b>	Detach Pipe Fail
<b>Rövid leírás</b>	Egy pumpától való cső sikertelen lecsatlakoztatását vizsgá teszeset.
<b>Teszt célja</b>	Cél, hogy a sikertelen lecsatlakoztatást teszteljük. Ehhez létrehozunk egy pumpát, nem rakunk rá csövet, majd megpróbáljuk leszedni róla a nem létező csövet. Tesztelt osztályok: Player, Pump, Pipe, Container

<b>Teszt-eset neve</b>	Pipe punctuality Test
<b>Rövid leírás</b>	Azt vizsgáljuk, hogy valóban blokkolva vannak e a lyukasztás műveletek foltozás után.
<b>Teszt célja</b>	Cső lyukaszthatóságának vizsgálata.

<b>Teszt-eset neve</b>	Player Moves To Pipe Success
<b>Rövid leírás</b>	A játékos csőre történő sikeres mozgásának tesztje.
<b>Teszt célja</b>	Cél, hogy a sikeres csőre való mozgást teszteljük. Ehhez létrehozunk egy csövet, amin nem áll másik játékos, és megpróbájuk odaléptetni a Playert. Tesztelt osztályok: Player, Pipe

<b>Teszt-eset neve</b>	Player Moves To Pipe Fail
<b>Rövid leírás</b>	A játékos csőre történő sikertelen mozgásának tesztje.
<b>Teszt célja</b>	Cél, hogy a sikertelen csőre való mozgást teszteljük. Ehhez létrehozunk egy csövet, ráállítunk egy játékest, és megpróbájuk odaléptetni a Playert. Tesztelt osztályok: Player, Pipe

<b>Teszt-eset neve</b>	Player Moves To Pump
<b>Rövid leírás</b>	A játékos pumpára való lépését teszteli.
<b>Teszt célja</b>	Cél, hogy játékos pumpára való mozgását teszteljük. Ehhez létrehozunk egy pumpát, és megpróbáljuk odaléptetni a Playert. Tesztelt osztályok: Player, Pump

<b>Teszt-eset neve</b>	Player Moves To Mountain Spring
<b>Rövid leírás</b>	A játékos hegyi forrásra való mozgatását tesztelő teszteset.
<b>Teszt célja</b>	Cél a játékos hegyi forrásra való mozgatásának tesztelése. Ez mindenkor sikertelen lesz, mivel úgy döntöttünk, hogy a hegyi forrásra nem lehet lépni. Tesztelt osztályok: Player, MountainSpring

## 7.4 Tesztelést támogató segéd- és fordítóprogramok specifikálása

Tesztelést támogató segédprogramnak a JUnit keretrendszer fogjuk használni, azon belül a JUnit 4-et. Ezzel hatékonyan ellenőrizhetjük az egyes osztályok metódusainak helyes működését. Külső, nem kifejezetten Java alkalmazásokhoz kapcsolódó teszteket, illetve egyéb segédprogramot nem használunk.

A 7.3-as fejezetnek megfelelően minden érdemi funkció tesztelésre kerül, cél a kód 80%-os tesztfedettsége.

## 7.5 Napló

Kezdet	Időtartam	Résznevők	Leírás
2023.04.21 12:30	30 perc	Réti Ádám Kopach Artem Tisza Miklós Czifra Barnabás Molnár Márton	Értekezlet: Feladatok kiosztása
2023.04.21 20:00	7 óra	Réti Ádám	Tevékenység: 7.1 Prototípus interface-definíciója 7.1.1 Az interfész általános leírása 7.1.2 Bemeneti nyelv elkészítése 7.1.3 Kimeneti nyelv elkészítése
2023.04.21	8 óra	Tisza Miklós	Tevékenység: Újmódosult osztály/szekvenciadiagram elkészítése
2023.04.22	5 óra	Czifra Barnabás	7.4, 7.3 kommentek, 7.3 tesztelési terv, 7.3 tesztesetek
2023.04.22	6 óra	Kopach Artem	7.0.1 módosult osztály, 7.0.2 , 7.0.3 előtti kommenteket írtam, 7.0.3.17 kommentet írtam, ** kódban kommenteket, JavaDoc is címsablont, szerekezés
2023.04.22 12:00	5 óra	Molnár Márton	Tevékenység: 7.2 Use-case leírások elkészítése 7.3 tesztesetek elkészítése

# Részletes tervezek

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

**Réti Ádám -**

**csapatvezető**

Tisza Miklós

Czifra Barnabás

Molnár Márton

Kopach Artem

(AO7JX4)      ket.vill.adam@gmail.com

(E1LX0J)      tisza.miklos.16@gmail.com

(NX9GA4)      barna.czifra@gmail.com

(KLM600)      molnar.marton.hun@edu.bme.hu

(JQBOLI)      kopach.artem@edu.bme.hu

2023.04.30

## Részletes tervezés

[A dokumentum célja, hogy pontosan specifikálja az implementálandó osztályokat, beleértve a privát attribútumokat és metódusokat, ezek definícióját is.]

[A dokumentum második fele részletesen be kell mutassa a korábban definiált be- és kimeneti nyelv szintaxisát felhasználva, hogy mely tesztekkel lesz a prototípus ellenőrizve.]

### 8.1 Osztályok és metódusok tervezése.

#### 8.1.1 Player

- **Felelősség**

Ez az osztály felelős a játékosok megvalósításáért, azaz a játékosokhoz tartozó akciókért, állapotokért stb.

- **Ősosztályok**

Ez az osztály nem rendelkezik ősosztállyal

- **Interfészek**

Ez az osztály nem valósít meg semmilyen interfészt

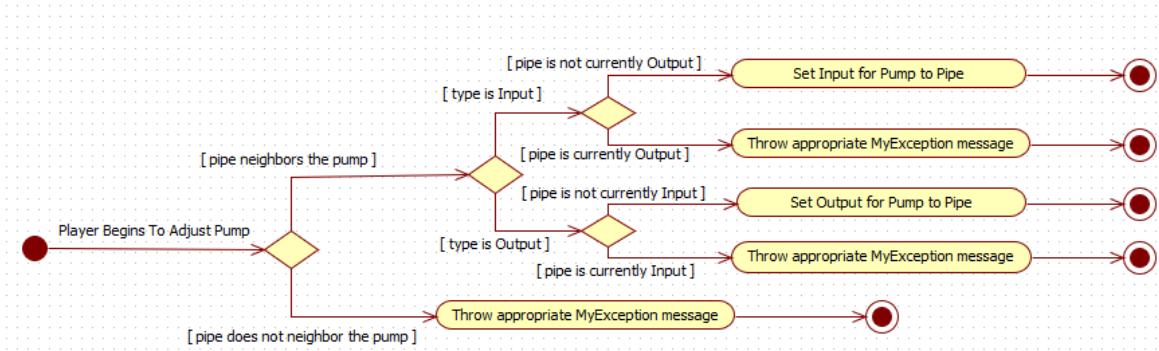
- **Attribútumok**

- **#position: Container:** Ez az attribútum adja meg, hogy a játékos éppen melyik konténeren helyezkedik el (Container -> Pipe, Pump, Mountain Spring, Cistern)
- **#carriedPipes: Pipe[]:** Ez az attribútum tárolja a játékosnál lévő csövek referenciaját egy generikusan növekvő tömbben
- **#carriedPump: Pump:** Ez az attribútum tárolja a játékosnál lévő (egy darab) pumpa referenciaját

- **Metódusok**

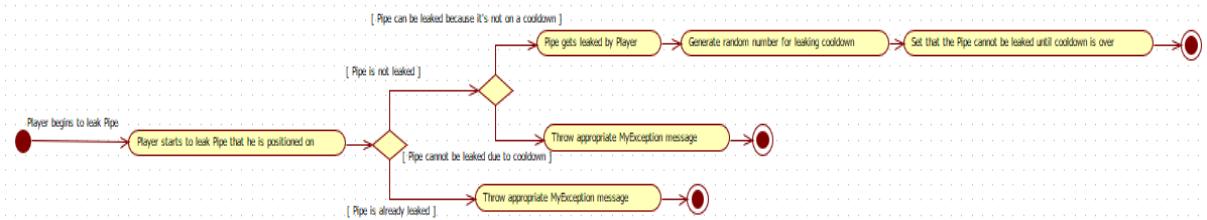
- **+Player(position: Container):** A játékos (Player) osztály konstruktora, itt argumentumként megadjuk, hogy a játékost mely konténeren akarjuk létrehozni, majd beállítjuk azon konténert a játékos pozíciójának. Emellett létrehozunk a játékos carriedPipes attribútumának egy üres ArrayList-et és a carriedPump-ot beállítjuk null-os értékre
- **+adjustPump(pi: Pipe, t: Type):** Ezen metódus segítségével képes a játékos a pumpának bemenetét vagy kimenetét változtatni (szigorúan csak annak a pumpának amelyiken éppen tartózkodik). A metódus argumentumában átadunk neki egy csövet és egy típust, a típus azt mondja meg, hogy bemenetet vagy kimenetet akarunk változtatni, a cső pedig, hogy melyik csőre akarjuk ezt a változtatást megtenni.

→ Metódus algoritmusához tartozó **Activity Diagram**:



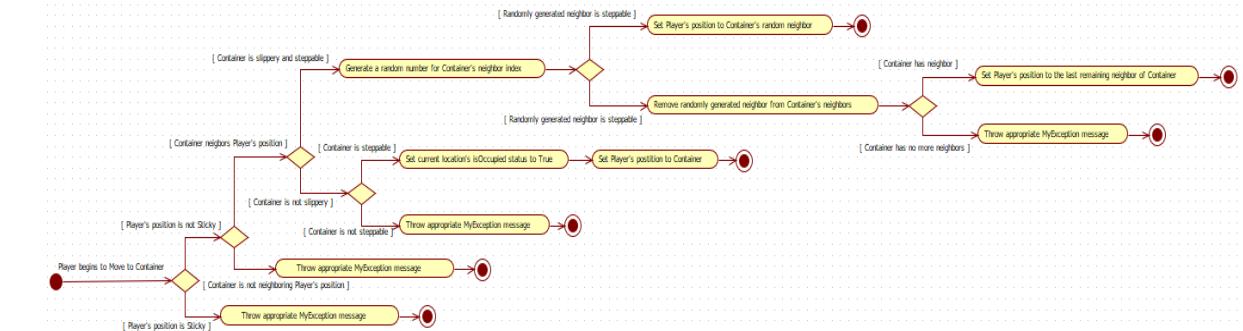
- **+makePipeSticky()**: A játékos ezen metódus segítségével képes egy csövet (szigorúan azt amelyen éppen áll) ragadóssá tenni.
- **+LeakPipe()**: A játékos (most már mind a szabotőr illetve szerelő képes) ezen metódussal képes elvégezni a cső lyukasztásának akcióját, szigorúan csak azon csövön amelyen a játékos éppen áll

→ Metódus algoritmusához tartozó **Activity Diagram**:



- **+Move(c: Container)**: A játékos ezen metódus segítségével képes a pozícióját az argumentumban megadott konténerre változtatni.

→ Metódus algoritmusához tartozó **Activity Diagram**:



- **+attachPipe()**: Ezen metódus felelős a játékos csőillesztési akciójáért, ahol is a játékos egy pumpához (szigorúan ahoz, amelyiken éppen tartózkodik) hozzácsatol egy csövet a saját általa hordozott csövek közül (a carriedPipes-ból)
- **+takePipe()**: A játékos a ciszterna elkészített csövei közül (madePipes) felvesz egyet és a saját hordozott csöveihez adja (carriedPipes)
- **+takePump()**: Hasonlóan a takePipe metódushoz itt is a játékos felvesz a ciszternából egy elkészített pumpát.
- **+attachPump()**: Ezen metódus segítségével képes a játékos egy csőhöz (szigorúan azon csőhöz amelyen éppen áll) pumpát kapcsolni, azaz a csövet ketté vágni és a két létrejött csőhöz, hozzákapcsolni
- **+detachPipe()**: Ezen metódus felelős a játékos cső pumpáról való leillesztési akciójáért, ahol is a játékos egy csövet (szigorúan arról a pumpáról amelyiken éppen tartózkodik) lecsatol egy csövet és hozzáadja az általa hordozott csövekhez (a carriedPipes-hoz)

## 8.1.2 Mechanic

- **Felelősség**

A szerelő elsősorban a kilyukadt csövek, illetve pumpák megjavításáért felelős. Ezen kívül tud ő is csövet lyukasztani, illetve egy rövid időre ragadóssá tudja tenni azt.

- **Ősosztályok**

**Player → Mechanic**

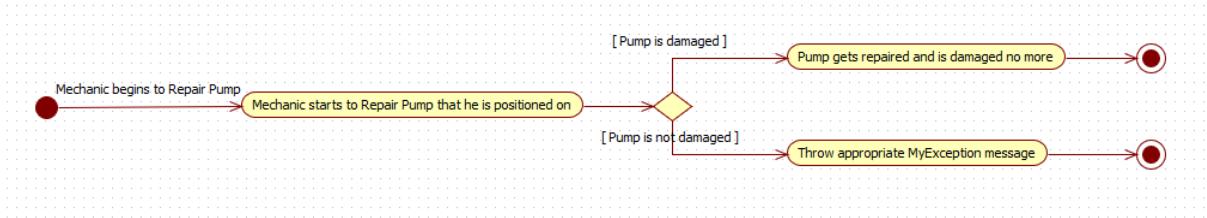
- **Attribútumok**

Csak a Player ősosztályból örökolt attribútumai vannak, melyek részletes leírásai a Player osztálynál találhatók.

- **Metódusok.]**

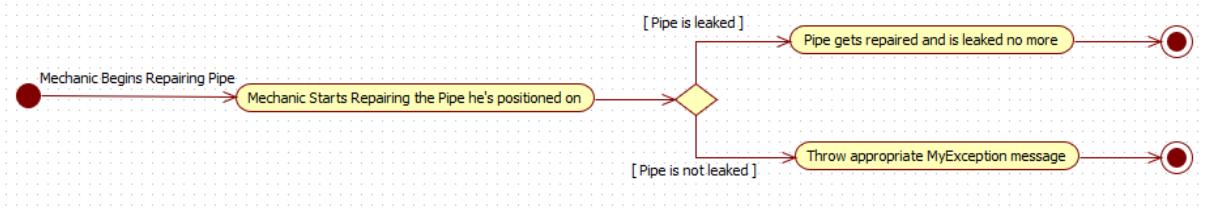
- **+Mechanic(position: Container):** Az osztály konstruktora, publikus láthatóságú. Paraméterként egy pozíciót(Container osztály) kapunk, amely megadja, hogy hol helyezzük el a játékost a pályán.
- **+RepairPump():** Pumpa megjavítását végző publikus metódus. A Pump osztály SetIsDamaged() metódusát használva beállítja a Pump isDamaged attribútumát false-ra.

→ Metódus algoritmusához tartozó **Activity Diagram:**



- **+RepairPipe():** Megjavítja a pozíciónál lévő csövet (getPosition() használ ), beállítja Pipe osztály isLeaked attribútumát false-ra.

→ Metódus algoritmusához tartozó **Activity Diagram:**



### 8.1.3 Saboteur

- **Felelősség**

A szabotör elsősorban kilyukasztja a csöveket, illetve tönkreteszzi a pumpákat. Ezen kívül csúszóssá tudja tenni a csöveket egy rövid időre, valamint a pumpákat ragadóssá tudja tenni.

- **Ósosztályok**

Player → Saboteur

- **Attribútumok**

Csak a Player ősosztályból örökolt attribútumai vannak, melyek részletes leírásai a Player osztálynál találhatók.

- **Metódusok.]**

- **+Saboteur(position: Container):** Az osztály konstruktora, publikus láthatóságú. Paraméterként egy pozíciót(Container objektumot) kapunk, amely megadja, hogy hol helyezzük el a játékost a pályán.
- **+makePipeSlippery():** publikus metódus amellyel egy cső csúszósságát (szigorúan azon csőnek, amelyen éppen állunk) tudjuk beállítani, azaz a Pipe osztály PipeGetsSlippery() publikus metódusát használva beállítjuk az IsSlippery attribútumát true-ra.

### 8.1.4 Map

- **Felelősség**

A játéktér egyes elemeiért való tárolásért, illetve azok kezeléséért felelős osztály.

- **Attribútumok**

- **- leakedWater: int :** statikus tagváltozó az elfolyt víz számítására
- **- players: Player[] :** a játékban résztvevő játékosokat generikus bővülő tömbben privát attribútum, csak referenciát tartalmaz a Player objektumokra
- **- containers: Container[] :** a Map objektumunkhoz tartozó Container leszármazottak(Pipe, Pump, Cistern, MountainSpring) tárolására szolgáló attribútum, szintén csak referenciákat tárol a Container objektumokra.

#### Metódusok.

1. **+Map():** az osztály publikus konstruktora. Létrehozzuk vele a tárolónkat.
2. **+connectedPumpToPipe(pu: Pump, pi: Pipe):** a paraméterként kapott pumpát és csövet egymáshoz csatlakoztatja, azaz egymásnak szomszédként állítjuk be őket.
3. **+increaseLeakWater():** Ez a függvény felelős növelni a leakedWater attribútum értékét.
4. **+removeElement(c : Container):** Eltávolít egy Containert a játéktérről, azaz kiveszi a container listából.
5. **+addElement(c : Container):** Hozzáad egy Containert a játéktérhez, azaz beleteszi a paraméterként kapott Container objektumot a containers listába.

### 8.1.5 Cistern

- **Felelősség**

A Cistern osztály felelős a csövek készítéséért, a víz tárolásáért és a ciszternában lévő vízszint növeléséért.

- **Ősosztályok**

Container → Cistern

- **Interfészek**

-

- **Attribútumok**

- **-input: Pipe** -A Cistern bemeneti csöve. Innen érkezik a víz a Cisternbe.
- **-collectedWater: int** -A Cisternbe eddig befolyt víz mennyiségét tárolja.
- **-freePump: Pump** -A szabadon mozgatható pumpát tárolja.
- **-randomPipeCreationTimet: int** -Ez felel a csövek véletlenszerű időközönkénti létrehozásáért.
- **-madePipes: Pipe[]** -A létrehozott csöveket tárolja, egy generikus, bővílő tömbben.

- **Metódusok**

- **+Cistern(Pipe input)** -Cistern konstruktora. Létrehozza a Cistern-t a paraméterként megadott bemeneti csővel. Inicializálja a Cistern változóit.
- **+void increaseCollectedWater()** -Növeli a CollectedWater attribútum értékét.
- **+void lifeCycle(int turnCount)** -Ha a paraméterként megadott turnCount érték hárommal való osztási maradéka 0, létrehoz egy új csövet. (Minden harmadik körben létrehoz egy új csövet.)
- **+boolean steppable()** -Lekérdezi, hogy rá lehet-e lépni a Cistern-re. Mindig true-val fog visszatérni, mivel a Cistern-en bármennyi játékos állhat egyszerre.
- **+void eval()** -A Cistern-hez tartozó kiértékelést valósítja meg. Ha folyik bele víz (inputState[0]), akkor növeli a bekerült víz mennyiségét.
- **+void setInputState()** -Beállítja a bemenet állapotát.
- **+boolean amInput(Container c)** -Lekérdezi, hogy a paraméterként kapott Container a Cistern inputja-e.

•

### 8.1.6 Container

#### Felelősség

Egy abstract osztály, egy általános tárolót valósít meg. Belőle származnak a Pipe, Pump, MountainSpring és Cistern osztályok. Felelős a mellette lévő Containerek tárolásáért, az `inputState` kezelésért.

- Ősosztályok
  - 
  - 
  -
- Interfészek
  - 
  -
- Attribútumok
  - `#neighbors: Container[]` -A Container mellett található Containerekkel referenciáját tárolja egy generikus, bővülő tömbben.
  - `#inputState: boolean[]` -Ez az attribútum felelős a víz mozgatásáért. Ebben az attribútumban tárolunk két értéket:
    - `0.index`-en tároljuk az előző *evaluation* kimenetét,
    - `1.index`-en tároljuk a mostani *evaluation* kimenetét.
- Ez az attribútum egy kis naplóként értelmezhető, mivel tároljuk, hogy előző körben mi történt, amellett, hogy most mi történik. A `false` (*hamis*) érték azt jelenti, hogy a Containerhez nem jutott víz. A `true` (*igaz*) érték azt jelenti, hogy a Containerhez jutott víz.
- Metódusok
  - `+boolean seeIfNeighbors(Container neighbor)` -Megnézi, hogy a paraméterként kapott Container szomszédja-e a Containerek.
  - `+String writeInputState()` -Ez a függvény felelős a `waterFlow()` függvényhez kapcsolódó kiiratásokért.
  - `+abstract void lifeCycle(int turnCount)` -Ezt a metódust implementálják a Pump és Cistern osztályok. A függvény azért felelős, hogy egy adott idő után

*megtörténjen valami a Container objektumunkkal. (Pl. Cistern-nél az új csövek létrehozása.)*

- **+boolean amInput(Container c)** -A metódus egy boolean értéket ad vissza abból adódóan, hogy az argumentumban megadott Container inputja-e az adott Containernek (this). A Pump és a Cistern osztály implementálja.
- **+abstract void movedFrom()** -A metódus azt a feladatot látja el, hogy amikor a játékos ellép egy Container-ről (Pipe-ról), beállítsa, hogy a Pipe nem occupied. A Pipe osztály implementálja.
- **+abstract void alterPump(Player player, Pipe pi, Type t)** -Ez a metódus felelős a pumpák outputjának és inputjának átállításáért. A Pump osztály implementálja.
- **+abstract void mendPipe()** -Ez a metódus felelős a cső megjavításáért. A Pipe osztály implementálja.
- **+abstract void mendPump()** -Ez a metódus felelős a pumpa megjavításáért. A Pump osztály implementálja.
- **+abstract void puncturePipe()** -A cső kilyukasztásáért felelős. A Pipe osztály implementálja.
- **+abstract void insertPump(Player player)** -A pumpa csőhöz való illesztéséért felelős. A Pipe osztály implementálja.
- **+abstract void extractPipe(Player player, Pipe pi)** -Ez a metódus felelős a csőhöz tartozó szabadvégű cső leszedéséért. A Pump osztály implementálja.
- **+abstract void insertPipe(Player player)** -A cső pumpához való illesztéséért felelős. A Pump osztály implementálja.
- **+abstract void pipeGetsSlippery()** -A cső csúszóssá tevéséért felelős.
- **+abstract boolean getIsSlippery()** -A Pipe osztály valósítja meg. Visszadja, hogy csúszós-e a Pipe.
- **+abstract void pipeGetsSticky()** -A cső ragadóssá tevéséért felelős.
- **+abstract boolean getIsSticky()** -A Pipe osztály valósítja meg. Visszaadja, hogy ragadós-e a Pipe.
- **+boolean steppbale()** -Visszadja, hogy rá lehet-e lépni a Container-re. A Pump, Pipe, Cistern és MountainSpring osztályok valóítják meg.
- **+abstract void eval()** -Ezt a függvényt megvalósítja a Pipe, Cistern, MountainSpring. Ez a metódus felelős a víz mozgásának kiértékeléséért.
- **+abstract void setInputState()** -Ez a metódus felelős az inputState megváltoztatásáért. A Pump, Pipe, Cistern és MountainSpring osztályok valóítják meg.
- **+void makeHistory()** -Az evaluation-t követően megváltoztaja az inputState értékeit.

### 8.1.7 MountainSpring

- **Felelősség**

A hegyi forrást valósítja meg. Innen származik a víz, amiért a játékosok versengenek.

- **Ősosztályok**

Container

- **Interfészek**

- **Attribútumok**

- **-output: Pipe** -Azt a csövet tárolja, amely kivezeti a vizet a forrásból.
- **-int waterCapac** -A forrás kapacitása. Ennyi vizet képes tárolni a forrás.

- **Metódusok**

- **+boolean steppable()** -Rá tudunk-e lépni a forrásra. Mindig false-al tér vissza, ugyanis a játékosok nem léphetnek rá a forrásra.
- **+void decreaseWaterAm()** -A forrásban lévő víz mennyiséget csökkenti.
- **+String writeInputState()** -Visszatér a forrás inputState-jének két elemével.
- **+boolean getIsSlippery()** -Lekérdezi, hogy csúszós-e a forrás. Mindig false-al tér vissza ugyanis a forrás nem lehet csúszós.
- **+boolean getIsSticky()** -Lekérdezi, hogy ragadós-e a forrás. Mindig false-al tér vissza ugyanis a forrás nem lehet ragadós.
- **+void eval()** -A forráshoz tartozó kiértékelő metódus. A metódus megnézi, van-e a forrásban víz. Ha igen, akkor meghívjuk a setInputState-et önmagára és csökkentjük a forrásban lévő víz mennyiségett. Ezután meghívjuk az output Pipe-ra a setInputState-et.
- **+void setInputState()** -Ez a metódus beállítja az inputState attribútum értékét.

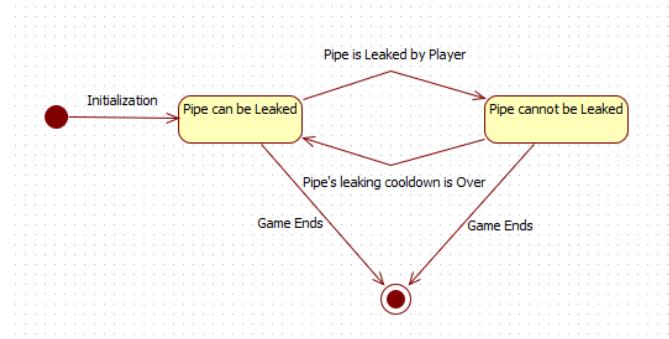
### 8.1.8 Pipe

- **Felelősség**

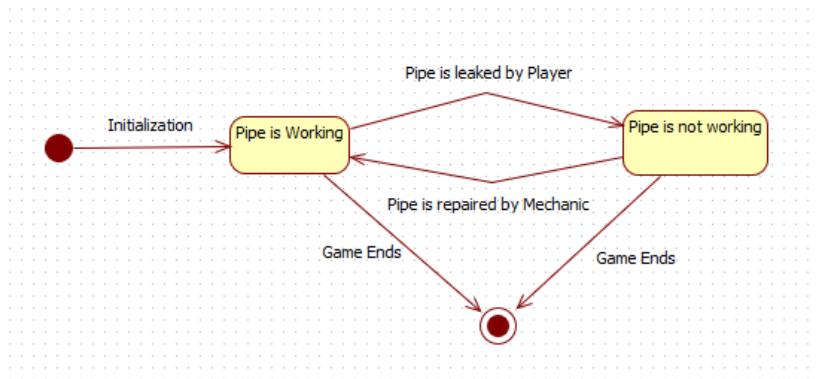
A Pipe osztály felelős a víz szállításáért, segítségevel szállítódik a víz a Pump-okon kereszül a Cistern-be.

- **State-Chartok**

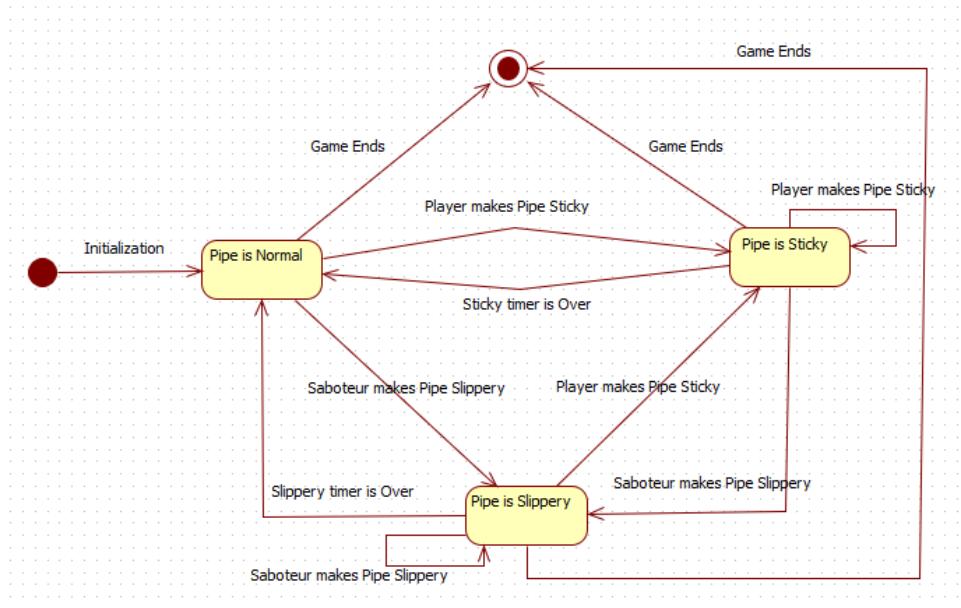
- Ez az állapotgép a cső lyukaszthatósági állapotjait mutatja be



- Ez az állapotgép a cső lyukasztott illetve nem lyukasztott állapotát mutatja be.



- Ez az állapotgép a cső csúszós, ragadós illetve sima állapotát mutatja be.



- Ősosztályok

**Container**

- Interfészek

-

- Attribútumok

- **-isLeaked: boolean** -Azt jelzi, hogy ki van-e lyukasztva a Pipe.
- **-canBeLeaked: boolean** -Azt jelzi, hogy a Pipe-ot ki lehet-e lyukasztani.
- **-leakedTimer: int** -Azt jelzi, hogy mennyi ideig nem lehet még kilyukasztani a Pipe-ot
- **-randomInterval: int** -A Pipe véletlenszerű időközönként történő lyukasztásának értéke.
- **-isOccupied: boolean** -Jelzi, hogy a cső foglalt-e. (Áll-e rajta játékos.)
- **-isSlippery: boolean** -Azt jelzi, hogy csúszós-e éppen a cső vagy nem.
- **-slipperyTimer: int** -Ennyi ideig marad csúszós a cső.
- **-isSticky: boolean** -Azt jelzi, hogy ragadós-e éppen a cső.
- **-stickyTimer: int** -Ennyi ideig marad ragadós a cső.

- Metódusok

- **+boolean steppable()** -Megadja, hogy az adott csőre léphet-e játékos. (isOccupied attribútum.)
- **+void movedFrom()** -Ez a függvény valósítja azt meg, hogy a cső amelyről a játékos ellépett, annak az isOccupied attribútuma false legyen.
- **+void mendPipe()** -A cső megjavításáért felelős függvény. Ha a cső lyukas, megjavítja, egyébként exception-t dob.
- **+void puncturePipe()** -A cső kilyukasztásáért felelős függvény. Ha a cső nincs még kilyukasztva és lyukasztható, akkor kilyukasztja, egyébként exception-t dob.
- **+void insertPump(Player player)** -A csőhöz hozzácsatlakoztatja a paraméterként kapott játékos által hordozott Pump-ot. Létrehoz egy új csövet,

- *hozzácsatlakoztatja a Pump-ot a régi és az új csőhöz, majd a Pump-hoz is hozzáadja a csöveket.*
- **+void pipeGetsSlippery()** -A cső csúszóssá tételeért felelős függvény.
- **+boolean getIsSlippery()** -Visszatér az isSlippery attribútum értékével.
- **+void pipeGetsSticky()** -A cső ragadóssá tételeért felelős függvény.
- **+boolean getIsSticky()** -Visszatér az isSticky attribútum értékével.
- **+void addPump(Pump pu, int index)** -A csőhöz hozzácsatlakoztaja a paraméterként kapott Pump-ot a paraméterként kapott indexű helyre. Ha van még hely, akkor a neighbors változóhoz hozzáadja a Pump-ot, egyébként exception-t dob.
- **+void addPump(Pump pu)** -A csőhöz hozzácsatlakoztaja a paraméterként kapott Pump-ot. Ha van még hely, akkor a neighbors változóhoz hozzáadja a Pump-ot, egyébként exception-t dob.
- **+void removePump(int index)** -Leszedi a csőről a paraméterként kapott indexű Pump-ot, ha nem üres a neighbors változó.
- **+void removePump(Pump pu)** -Leszedi a csőről a paraméterként kapott Pump-ot, ha nem üres a neighbors változó és az adott pump benne van a neighbors változóban.
- **+void lifeCycle(int turnCount)** -A pipe isSlippery, isSticky és canBeLeaked állapotát változtaja meg egy adott idő után.
- **+void eval()** -Ez a függvény valósítja meg a víz mozgásának csőnél való kiértékelését. Az első amit megnézünk, hogy az inputState[0] értéke igaz-e, azaz, hogy az előző körben volt-e benne víz,
- *ha nem akkor nem is foglalkozunk ezzel tovább.*
- *Ha volt akkor vizsgáljuk meg a többi esetet, mint például szabadvégű-e a cső (isLooseEnd()), ha igen akkor növeljük a kifolyt víz mennyiségettét.*
- *Ezután megkeressük a cső szomszédságában azt a Container-t akinek-ő az inputja. Ezt követően egy újabb elágazáshoz érkezünk, ahol is azt nézzük meg, hogy:*
  - *1. Ki van-e lyukaszta? - ha igen akkor növeljük a kifolyt víz mennyiségettét, máskülönben megyünk mélyebbre(ha a másik feltétel is teljesül).*
  - *2. Van-e outputja? - azaz, hogy létezik-e olyan Container akinek ő lenne az inputja*
  - *→ha van ilyen akkor erre meghívjuk a setInputState-et*
  - *→ha nincs ilyen akkor nem csinálunk semmit és visszatérünk*
  - **+ void setInputState()** -Amennyiben ez a függvény meghívódik az inputState[1] átállítjuk true (igaz) értékre.
  - **+boolean isLooseEnd()** -Visszatér azzal, hogy a cső vége szabadon lóg-e.

## 8.1.9 Pump

- **Felelősség**

A Pump osztály a pumpák megvalósításáért felelős. Ez az osztály teszi lehetővé a csövek közötti összeköttetést.

- **Ősosztályok**

Container

- **Interfészek**

-

- **Attribútumok**

- **-input: Pipe** -A Pump bemeneti csöve.
- **-output: Pipe** -A Pump kimeneti csöve.
- **-isDamaged: boolean** -Ez tárolja, hogy a Pump sérült-e.
- **-randomDamageValue: int** -Azt az értéket tartalmazza amely meghatározza, hogy az adott pumpa mely körben fog megsérülni.
- **-maxPipeAmount: int** -Az attribútum meghatározza, hogy a Pump-hoz hánny db Pipe csatlakozhat.

- **Metódusok**

- **+Pump(int maxPipeAmount)** -Pump konstruktora. Inicializálja a változókat és beállítja a maxPipeAMount attribútumot a paraméterként kapott értékre.
- **+void mendPump()** -A Pump megjavításáért felelős metódus. Ha a Pump el van romolva, megjavítjuk, egyébként exception-t dob.
- **+void insertPipe(Player player)** -Hozzácsatlakoztatja a paraméterként kapott Player által hordozott Pipe-ot a Pump-hoz.
- **+void extractPipe(Player player, Pipe pi)** -Elveszi a paraméterként kapott Pipe-ot a Pump-tól és a Player-hez adja hozzá.
- **+boolean steppable()** -Megnézi, hogy lehet-e a Pump-ra lépni. Mindig true-val tér vissza, ugyanis a Pump-ra akárhány játékos léphet.
- **+void lifeCycle(int turnCount)** -Elrontja a pumpát a randomDamageValue és a paraméterül kapott érték alapján.
- **+void addPipe(Pipe pi)** -Hozzáadja a paraméterül kapott Pipe-ot a Pump-hoz, amennyiben lehet még csövet hozzácsatlakoztatni.
- **+void removePipe()** -Elveszi a paraméterül kapott Pipe-ot a Pump-tól, ha nem üres a neighbors változó.
- **+boolean amInput(Container c)** -Meghatározza, hogy a paraméterül kapott Container inputja-e a Pumpnak.
- **+void setInputState()** -Ez a függvény egyszerűen csak annyit csinál, hogy az output Pipe-jára "továbbítja" a setInputState() függvény hívást.
- **+ boolean isAllConnected()** -Ha teli van a pumpa, vagyis már nem lehet több csövet hozzácsatlani, akkor true-val tér vissza.

**+void alterPump(Player player, Pipe pi, Type t)** -Ez a függvény felelős a pumpa outputjának illetve inputjának változtatásáért. A paraméterül kapott csőről megállapítjuk, hogy szomszédos-e ezzel a pumpával.

Ha igen, megnézzük, hogy melyiket akarjuk változtatni: a pumpa kimenetét (output) vagy bemenetét (input), ezt a t paraméterrel adjuk meg.

*Mind a két esetben megnézzük, hogy az egyik már nem másikhoz tartozik azaz, ha inputot akarunk változtatni nem az output-e amire változtatni akarjuk és fordítva.*

### 8.1.10 Controller

- **Felelősség**

*Ez az osztály felelős a víz mozgatásáért, illetve a pumpák véletlenszerű elrontásáért (ezen osztály egy singleton osztály, csak 1 példányban fogjuk majd létrehozni)*

- **Ősosztályok**

*Nem rendelkezik ősosztállyal*

- **Interfészek**

*Nem implementál semmilyen interfészt*

- **Attribútumok**

- **-map: Map**: Ezen attribútumban tároljuk a pálya referenciáját, ez a pályában tartott konténer referenciák miatt szükséges (hiszen ezek segítségével működik a Controller osztály több függvénye is)
- **-controller: Controller**: Ezen attribútum segítségével tesszük lehetővé azt, hogy az osztály csak 1 példányban valósul meg
- **-turnCount: int**: Itt tartjuk számon az eltelt körök számát (ez majd későbbi implementálásban a Turn osztály attribútuma lesz, azonban több funkció is igényli ennek meglétét)

- **Metódusok**

- **+getInstance(): Controller**: Ezen metódus segítségével, hozzuk létre a Controller osztályunkból az egyetlen példányt
- **+evaluateCycles()**: Ez a metódus felelős a "körben" történő kiértékelésért. Példaként ebben valósul meg a pumpáknak a véletlenszerű elrontása, a csőnek a ragadós illetve csúszós állapotának elmúlása, illetve a ciszternában a csöveknek adott időn (körön) belüli elkészítése
- **+waterFlow()**: A vízfolyását megvalósító metódus  
Vegyük egy példát:

*Legyen egy ilyen összeköttetésünk: **MS→PU→CS** -ahol **MS-Mountain Spring**, ( $\rightarrow$ )-**Pipe**, **PU-Pump** és végül **CS-Cistern***

*Mivel a **Mountain Springból** jön a víz, ennek **inputState**-je {true, true} addig amíg el nem fogy, ezt mindig a hozzá tartozó **setInputState**-tel állítjuk be*

**FONTOS!** Az **MS**-en kívül minden más **Container** **inputState**-je {false, false} -ként van inicializálva

*Jön a hozzácsatlakozó cső (azaz **MS input Pipe**-ja), legyen ez a cső '**pipe1**' '**pipe1**' cső csatlakozik **PU pumpához** akinek ő az **input** csöve lesz*

*Ezt követően jön a **PU** akinek pedig **output** csöve legyen '**pipe2**' aki legvégül csatlakozik a **CS** ciszternához is akinek az **input** csöve lesz*

*Magá a folyamat:*

1. lépés: lekérdezzük a pályához tartozó összes konténert
2. lépés: végigmegyünk az összes konténeren
3. lépés: profit

*A profithoz lehet kell egy kis magyarázat. A for ciklusunk belsejében meghívjuk az adott konténerre a **eval()** függvényt*

*Ez az eval függvény fogja meghatározni, hogy:*

*1 - Hogyan folyik a víz? Példaként ha egy csövet nézünk folyhat-e benne tovább víz ha lyukasztva van?*

*2 - Ő fogja meghívni a következő konténerre a setInputState függvény ezzel beállítva az aktuális körben, hogy mi történt vele*

*Maga a példa végigkövetve (fontos, hogy a konténerek sorrendje nem befolyásolja a víz működését tehát itt nincs probléma):*

*Legyen a 'containers' első eleme MS (hegyi forrás) erre meghívjuk az eval függvényt → ez beállítja MS inputValue-jét {true, true}-ra majd meghívja az outputjára az setInputState-t(jelen helyzetben 'pipe1')*

*A csőre meghívódik a setInputState(), beállítódik az inputValue[1] igazra, azaz, hogy igen is folyik benne víz*

*Ezt követően a cső-re (tegyük fel ő következik az MS után a konténer listában) meghívódik az eval() azonban itt nem történik még semmi*

*Visszatérünk ide és most történik a fontos dolog. Meghívódik a csőre a makeHistory(), azaz "felcserélődik"(igazából csak az 1. indexen lévő megvált a 0. indexen lévőre) a két értéke az inputValue-nek*

*Tehát 'pipe1' inputValue-je a következő lesz {[0]true, [1]false}*

*Ez után jön egy csomó unalmas semmit mondó függvényhívás (mivel, ugye mindenki a setInputState[0] értéke false)*

*Ez volt egy kör, de még egy kört megnézve világossá válik mi történt*

*Következő kör, megint meghívódik elsőnek MS-en az eval ami meghívja 'pipe1' setInputState-jét amiből fakadóan 'pipe1' inputValue-je {true, true} lesz*

*Majd meghívódik az eval 'pipe1'-re, most már fog történni valami itt is azt, hogy mi teljes egészében nem részletezem DE ami fontos, hogy meghívódik 'pipe1'-hez tartozó PU setInputState-je*

*PU setInputState-je meghívja az ő output-án rejlő cső 'pipe2' setInputState-jét (így {false, true} lesz a tartalma) és most már ebben a csőben is megjelenik a víz*

*És így megy végig a víz egészen a ciszternáig (CS) ahol is legvégül majd sok kör lefolyása után szintén megjelenik a víz*

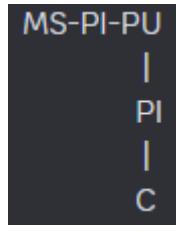
## 8.2 A tesztek részletes tervei, leírásuk a teszt nyelvén

Itt a tesztek inicializálásáról egy előre összeállított pályát fogunk alkalmazni amit a **8.2.0.-**ben definiálunk. (Ez a prototípus koncepciója nevű dokumentumban még nem létezik). Ezt a pályát az **operationLoadMapTest.txt**-vel lehet meghívni

### 8.2.0 Inicializálás

- **Leírás:**

Ez a teszt **operationLoadMapTest** bemenet esetén betölt a teszteléshez egy előre összeállított pályát. A konkrét bemeneteket, amelyekkel létrehoztuk a teszt pályát (Test.txt) a **Bemenet**-nél látható. Ez egy pálya 5 pálya elemmel. (Mindegyik típusból legalább 1)



- **Ellenőrzött funkcionalitás, várható hibahelyek:**

Itt azt ellenőrizzük, hogy sikeresen létrejöttek a pályaelemek, és össze lehet őket csatlakoztatni. Nem várunk itt hibát, mert úgy állítjuk előre össze az inicializáló pályát.

- **Bemenet**

*operationCreateMap*

*operationCreateContainerMountainSpringAt0\_0*

*operationCreateContainerPipeAt1\_0*

*operationCreateContainerPumpAt2\_0*

*operationCreateContainerPipeAt2\_1*

*operationCreateContainerCisternAt2\_2*

*operationConnectContainerAt0\_0ToContainerAt1\_0*

*operationConnectContainerAt1\_0ToContainerAt2\_0*

*operationConnectContainerAt2\_0ToContainerAt2\_1*

*operationConnectContainerAt2\_1ToContainerAt2\_2*

*operationSaveMapTest.txt*

- **Elvárt kimenet**

*New map was successfully created*

*MountainSpring was successfully created at X:0 , Y:0*

*Pipe was successfully created at X:1 , Y:0*

*Pump was successfully created at X:2 , Y:0*

*Pipe was successfully created at X:2 , Y:1*

*Cistern was successfully created at X:2 , Y:2*

*Container at X:0 , Y:0 was successfully connected to container at X:1 , Y:0*

*Container at X:1 , Y:0 was successfully connected to container at X:2 , Y:0*

*Container at X:2 , Y:0 was successfully connected to container at X:2 , Y:1*

*Container at X:2 , Y:1 was successfully connected to container at X:2 , Y:2*

*Map successfully save into Test.txt*

### 8.2.1 Pályaelem törlés

- **Leírás** Ez a teszt eltávolít pályaelemeket a térképről. Ehhez magát a térképet is be kell tölteni (parancs: operationLoadMapTest.txt ), szintén a parancs operationDeleteContainerAtX\_Y eltávolítjuk a pályaelemeket a térképről az x,y koordinátákból.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ebben az esetben nem várunk különösebb hibát, inkább azt az egyértelmű következtést várjuk, hogy a 0,2 koordinátánál, mivel ott nincs semmi. Ez a funkció biztosítja, hogy a kimenet helyes és érthető legyen.
- **Bemenet**

*operationLoadMapTest.txt*

*operationDeleteContainerAt0\_2*

*operationDeleteContainerAt0\_0*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Container cannot be deleted at X:0 , Y:2 , because there is nothing there*

*Container was successfully deleted at X:0 , Y:0*

### 8.2.2 Nem megengedett pályaépítés

- **Leírás:** Ebben a tesztnél a különböző koordinátákon különböző típusú containerek létrehozását kell tesztelnünk.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ebben a tesztnél garantáljuk az érthető kimenetet, amely nem eredményezhet hibát, meg kell mondania a felhasználónak, hogy a teszt sikeresen befejeződött-e, ha nem fejeződött be időben, a felhasználónak helyes információt kell kapnia arról, hogy mi a hiba.  
pl. a MountainSpring nem hozható létre újra, ezért a felhasználót értesíteni kell, hogy egy ilyen tároló létrehozása nem fog működni, mivel csak egy MountainSpring lehet a térképen. Ennek a tesztnél minden ellenőriznie kell azt is, hogy a felhasználó által kiválasztott koordinátákon van-e már konténer. Ha már van konténer, akkor erről is értesíteni kell a felhasználót.
- **Bemenet**

*operationLoadMapTest.txt*

*operationCreateContainerPipeAt0\_0*

*operationCreateContainerMountainSpringAt0\_1*

*operationCreateContainerPipeAt1\_1*

*operationCreateContainerBuildingAt1\_2*

*operationConnectContainerAt1\_1ToContainerAt0\_1*

*operationConnectContainerAt1\_1ToContainerAt2\_2*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Pipe cannot be created at X:0 , Y:0 , because at this position a container already exists.*

*MountainSpring cannot be created at X:0 , Y:1 , because the maximum number of MountainSprings is 1.*

*Pipe was successfully created at X:1 , Y:1*

*Building is not a valid value*

*Container at X:1 , Y:1 cannot be connected to Container at X:0 , Y:1 , because they are the same type of containers.*

*Container at X:1 , Y:1 cannot be connected to Container at X:2 , Y:2 , because they are not neighbors.*

### 8.2.3 Játékosok létrehozása

- **Leírás:** Egy játékos létrehozásának tesztelése a térképen. Ebben a tesztesetben meg kell próbálnunk játékosokat létrehozni a térképen.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ennek a tesztnak szintén nem szabad hibát adnia, hanem inkább egy felhasználó számára érhető kimenetet, pl. ha egy felhasználó egy nem létező karaktert akar létrehozni, pl. Buildert, akkor a kimenetből le kell tudnunk következtetni, hogy ez nem érvényes karakter a játékunkban. Ellenőriznünk kell a steppable koordinátáit is, hogy van-e olyan eleme, amire rá tudunk állítani egy játékost.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerBuilderAt1\_0*

*operationCreatePlayerMechanicAt1\_0*

*operationCreatePlayerMechanicAt0\_0*

*operationCreatePlayerMechanicAt0\_2*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Builder is not a valid value*

*Mechanic was successfully created at X:1 , Y:0*

*Mechanic cannot be created at X:0 , Y:0 , because the container is not steppable.*

*Mechanic cannot be created at X:0 , Y:2 , because there is no container at X:0 , Y:2 to stand on.*

#### 8.2.4 Játékosok mozgatása

- **Leírás:** Ebben az esetben meg kell értenünk a játékos mozgását a térképen.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ebben a tesztből sem várunk hibákat, hanem feldolgozott, érthető kimenetet szeretnénk látni. Például, ha egy játékos megróbál egy nem létező karaktert mozgatni, akkor azt kell mondani, hogy a karakter nem létezik, és a játékos mozgásakor ellenőrizni kell, hogy a konténer léptethető-e. Arra is figyelnünk kell, hogy a konténer lehet maximális, ha úgy, akkor a konténerbe nem lehet mozogni is.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt1\_0*

*operationCreatePlayerSaboteurAt2\_1*

*player0moveToLeft*

*player1moveToLeft*

*player1moveToSouth*

*player1moveToRight*

*player1moveToDown*

*player2moveToUp*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:1 , Y:0*

*Saboteur was successfully created at X:2 , Y:1*

*There is no such player as Player 0*

*Player1 cannot move Left, because a container is not steppable.*

*South is not a valid value*

*Player1 successfully moved Right*

*Player1 cannot move Down, because the container this direction is occupied*

*Player2 successfully moved Up*

### 8.2.5 Játékos pumpa átállítás

- **Leírás:** Ebben az esetben ellenőriznünk kell a játékos által beállított szivattyút.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ebben a teszesetben nem azt várjuk, hogy a kód hibaüzenettel záruljon, hanem egyértelmű kimenetet szeretnénk látni, amely megmondja a felhasználónak, hogy ez jó vagy rossz volt-e és miért. Ha például a játékos megpróbál megjavítani egy olyan szivattyút, amely nincs a közelében, a felhasználónak erről üzenetet kell kapnia erről. Ha azonban a játékos egy olyan tárolóhoz megy, amelyben az adott szivattyú van, vagy kiválaszt egy olyan szivattyút, amelyet befolyásolni tud, akkor üzenetet kell kapnia a sikeres beállításról.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt1\_0*

*player1AdjustPumpToUp*

*player1moveToRight*

*player1AdjustPumpToUp*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:1 , Y:0*

*Player1 cannot adjust pump output to Up, because the player is not at a pump*

*Player1 successfully moved Right*

*Player1 successfully adjust pump output to Up*

### 8.2.6 Játékos csőlyukasztás

- **Leírás:** Ebben a teszben meg kell néznünk, hogy a játékos hogyan tudja lyukasztani a csövet.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Azt sem várjuk el, hogy a teszt hibával végződjön. Ha valami baj van a bemeneti paraméterekkel, a kimenetnek jeleznie kell ezt. Például, ha a játékos nem azon a csövön van, amelyet a lyukasztáshoz választott, a felhasználónak erről üzenetet kell látnia a kimeneten.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt2\_0*

*player1LeakPipe*

*player1moveToLeft*

*player1LeakPipe*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:2 , Y:0*

*Player1 cannot leak a pipe, because they are not standing on a pipe*

*Player1 successfully moved Left*

*Player1 successfully leaked the pipe at their position*

### 8.2.7 Játékos csőillesztés

- **Leírás:** Ebben a teszben ellenőriznünk kell a Játékos csőillesztését
- **Ellenőrzött funkcionalitás, várható hibahelyek:** mint legutóbb, most sem várunk hibákat a teszt futtatásakor, de elvárjuk, hogy a kimenet részletesen beszámoljon a felhasználónak az esetleges problémákról.  
Például, ha a játékos megpróbál egy csövet csatlakoztatni, miközben rossz pozícióban van, akkor egy üzenetet kell kapnia, hogy ez nem lehetséges.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt1\_0*

*manualCreateContainerPipeAtPlayer1*

*player1AttachPipeRight*

*player1movetoRight*

*player1AttachPipeRight*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:1 , Y:0*

*Pipe was successfully created in player1 's inventory*

*Player1 cannot attach a pipe at their position, because the container they are standing at is the same type*

*Player1 successfully moved Right*

*Player1 successfully attached a pipe at their position*

### 8.2.8 Játékos pumpa illesztés

- **Leírás:** Ebben a teszben ellenőriznünk kell játékos pumpa illesztését
- **Ellenőrzött funkcionalitás, várható hibahelyek :**Ez a teszt nem tervez hibát az indításkor, de valamilyen "rossz" bemeneti paraméter esetén érhető kimenetet várnánk. Például, ha a játékos megpróbál egy pumpát csatlakoztatni, miközben rossz pozícióban van, akkor egy üzenetet kell kapnia, hogy ez nem lehetséges.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt2\_0*

*manualCreateContainerPumpAtPlayer1*

*player1AttachPump*

*player1movetoLeft*

*player1AttachPump*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:2 , Y:0*

*Pump was successfully created in player1 's inventory*

*Player1 cannot attach a pump at their position, because the container they are standing at is the same type*

*Player1 successfully moved Left*

*Player1 successfully attached a pump at their position*

### 8.2.9 Játékos cső leszerelés

- **Leírás:** Ebben a teszben ellenőriznünk kell Játékos cső leszerelését
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnunk valamilyen "rossz" bemeneti paraméter esetén. Például, ha egy játékos leválaszt egy csövet, ha a játékos helyzete rossz, a felhasználónak üzenetet kell kapnia, hogy nem csövön van. Ha a játékos a helyes pozícióba mozog, és megpróbálja megismételni a műveletet, akkor sikerülnie kell.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerMechanicAt2\_0*

*player1DetachPipe*

*player1MoveToDown*

*player1DetachPipe*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Mechanic was successfully created at X:2 , Y:0*

*Player1 cannot detach a pipe at their position, because they are not standing on a pipe*

*Player1 successfully moved Down*

*Player1 successfully detached a pipe at their position*

### 8.2.10 Játékos csőjavítás

- **Leírás:** Ebben a teszben ellenőriznünk kell Játékos csőjavítását
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnunk valamilyen "rossz" bemeneti paraméter esetén. Ha például egy játékos megpróbál megjavítani egy olyan csövet, amelyen nem áll, a felhasználónak erről üzenetet kell kapnia. Ha a játékos sikeresen feláll egy megfelelő SÉRÜLT csőre, és újra megpróbálja megjavítani, a javításnak sikerülnie kell.

- **Bemenet**

*operationLoadMapTest.txt*

*manualDamageContainerAt1\_0*

*operationCreatePlayerMechanicAt2\_0*

*player1RepairPipe*

*player1MoveToLeft*

*player1RepairPipe*

*player1RepairPipe*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The container at X:1 , Y:0 was damaged successfully*

*Mechanic was successfully created at X:2 , Y:0*

*Player1 cannot repair a pipe at their position, because they are not standing on a pipe*

*Player1 successfully moved Left*

*Player1 successfully repaired a pipe at their position*

*Player1 cannot repair a pipe at their position, because the pipe they are standing on is not leaked*

### 8.2.11 Játékos pumpa javítás

- **Leírás:** Ebben a teszben ellenőriznünk kell a játékos pumpa javítását
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamelyen "rossz" bemeneti paraméter esetén. Ha például egy játékos megpróbál megjavítani egy olyan pumpát, amelyen nem áll, a felhasználónak erről üzenetet kell kapnia. Ha a játékos sikeresen feláll a megfelelő pumpára, és újra megpróbálja megjavítani, a javításnak sikerülnie kell. Ha a szivattyút már megjavította, a játékosnak üzenetet kell kapnia a javítás újra megkísérlesekor.

- **Bemenet**

*operationLoadMapTest.txt*

*manualDamageContainerAt2\_0*

*operationCreatePlayerMechanicAt1\_0*

*player1RepairPump*

*player1moveToRight*

*player1RepairPump*

*player1RepairPump*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The container at X:2, Y:0 was damaged successfully*

*Mechanic was successfully created at X:1 , Y:0*

*Player1 cannot repair a pump at their position, because they are not standing on a pump*

*Player1 successfully moved Right*

*Player1 successfully repaired a pump at their position*

*Player1 cannot repair a pump at their position, because the pump they are standing on is not damaged*

### 8.2.12 Játékos cső csúszósítása

- **Leírás:** Ebben a teszben ellenőriznünk kell játékos cső csúszósítását
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamelyen "rossz" bemeneti paraméter esetén. Például, ha egy játékos nem egy csőn áll, akkor nem tudja megcsinálni a csúszósítást és kell kapni egy üzenetet erőll. Ha Player erre a csőre fog lépni és próbál még egyszer slippery csinálni, akkor kell sikerül üzenetet kapni

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerSaboteurAt2\_0*

*player1MakePipeSlippery*

*player1moveToLeft*

*player1MakePipeSlippery*

*player1MakePipeSlippery*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Saboteur was successfully created at X:2 , Y:0*

*Player1 cannot make a pipe slippery at their position, because they are not standing on a pipe*

*Player1 successfully moved Left*

*Player1 successfully made a pipe slippery at their position*

*Player1 successfully made a pipe slippery at their position again*

### 8.2.13 Játékos cső ragadóssá tétele

- **Leírás:** Ebben a tesztnél ellenőriznünk kell, hogy hogyan fogja a játékos csövet ragadóssá tenni
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnunk valamelyen "rossz" bemeneti paraméter esetén. Például, ha egy játékos nem egy csón áll, akkor nem tudja megcsinálni a ragadóssát és kell kapni egy üzenetet erről. Ha Player erre a csőre fog lépni és próbál még egyszer sticky csinálni, akkor kell sikerül üzenetet kapni
- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerSaboteurAt2\_0*

*player1MakePipeSticky*

*player1moveToLeft*

*player1MakePipeSticky*

*player1MakePipeSticky*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Saboteur was successfully created at X:2 , Y:0*

*Player1 cannot make a pipe sticky at their position, because they are not standing on a pipe*

*Player1 successfully moved Left*

*Player1 successfully made a pipe sticky at their position*

*Player1 successfully made a pipe sticky at their position again*

### 8.2.14 Pályaelem manuális létrehozása ciszternánál

- **Leírás:** Ebben a tesztnél ellenőriznünk, hogyan pályaelem manuális létrehozása ciszternánál
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnunk valamelyen "rossz" bemeneti paraméter esetén. Például, amikor létrehozunk egy elemet a Cisternánál, akkor kapunk sikerül üzenet. De ha a Cistern tele van, akkor a felhasználónak üzenetet kell kapnia arról.

- **Bemenet**

*operationLoadMapTest.txt*

*manualCreateContainerPipeAtCistern*

*manualCreateContainerPipeAtCistern*

*manualCreateContainerPumpAtCistern*

*manualCreateContainerPumpAtCistern*

*operationDeleteContainerAt2\_2*

*manualCreateContainerPipeAtCistern*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Pipe was successfully created in cistern's inventory*

*Pipe was successfully created in cistern's inventory*

*Pump was successfully created in cistern's inventory*

*Pump cannot be created in cistern's inventory, because the inventory is full*

*Container was successfully deleted at X:2 , Y:2*

*Pipe cannot be created in cistern's inventory, because there is no cistern on the map*

### 8.2.15 Pályaelem manuális létrehozása játékosnál

- **Leírás:** Ebben a tesztnél ellenőriznünk, hogyan pályaelem manuális létrehozása játékosnál
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamilyen "rossz" bemeneti paraméter esetén. Például, amikor létrehozunk egy játékosnál egy elemet, akkor kapunk sikeres üzenet. De ha a játékos inventoryja tele van, akkor a felhasználónak üzenetet kell kapnia arról.

- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerSaboteurAt2\_0  
manualCreateContainerPipeAtPlayer1  
manualCreateContainerPipeAtPlayer1  
manualCreateContainerPumpAtPlayer1  
manualCreateContainerPumpAtPlayer1*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Saboteur was successfully created at X:2 , Y:0*

*Pipe was successfully created in player1 's inventory*

*Pipe was successfully created in player1 's inventory*

*Pump was successfully created in player1 's inventory*

*Pump cannot be created in player1 's inventory, because the inventory is full*

### 8.2.16 Víz manuális folyatása

- **Leírás:** Ebben a tesztnél ellenőriznünk, hogyan víz manuális folyatása
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamilyen "rossz" bemeneti paraméter esetén. Például, amikor van MountainSpring víz folyatása sikeres, de ha MountainSpringet töröljük akkor sikertelen üzenetet kell kapni, mert már nincs MountainSpring

- **Bemenet**

*operationLoadMapTest.txt*

*manualFlowWater*

*operationDeleteContainerAt0\_0*

*manualFlowWater*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The water is flowing in the containers*

*Container was successfully deleted at X:0 , Y:0*

*The water cannot flow, because there is no MountainSpring on the map*

### 8.2.17 Játék kör számának manuális beállítása

- **Leírás:** A feladat egy játék kör számának manuális beállítása és az ezzel kapcsolatos műveletek végrehajtása. A programnak lehetőséget kell biztosítania a felhasználó számára a játék kör számának manuális beállítására, valamint a körök számának növelésére és a jelenlegi körök számának lekérdezésére. A program továbbá egy adott térképfájl betöltését is lehetővé teszi.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnak valamilyen "rossz" bemeneti paraméter esetén. Hibás bemenet kezelése: A programnak ellenőriznie kell a felhasználó által megadott bemenetet, és helyesen kell kezelnie az esetleges hibás bemeneteket, például a nem szám típusú bemeneteket. Helytelen térképfájl formátum: A programnak ellenőriznie kell a betöltött térképfájl formátumát, és helyesen kell kezelnie az esetleges helytelen formátumot. Kör számának túllépése: A programnak ellenőriznie kell a körök számát, és helyesen kell kezelnie az esetleges túllépést. Ha a felhasználó olyan kör számot ad meg, amely nem integer típusú, akkor ezt a kérést sikertelennek kell tekinteni, mert csak integer típusú lehet a bemenet.

- **Bemenet**

*manualSetTurnCount5  
operationLoadMapTest.txt  
manualSetTurnCount5  
manualSetTurnCount1  
listCurrentTurn  
manualSetTurnCount3.5  
manualIncreaseTurnCountBy1  
manualIncreaseTurnCountBy1  
manualIncreaseTurnCountBy1  
listCurrentTurn*

- **Elvárt kimenet**

*Current turn count cannot be increased by 5, because there is no loaded/created map  
Map Test.txt successfully loaded  
Current turn count successfully set to 5  
Current turn count successfully set to 1  
3.5 is not an integer  
The current turn is: 1.  
Current turn count successfully increased by 1  
Current turn count successfully increased by 1  
Current turn count successfully increased by 1  
The current turn is: 4.*

### 8.2.18 Pályaelemek állapotának manuális manipulációja 1

- **Leírás:** A feladat egy játéktér elemeinek manuális manipulációja, különös tekintettel a konténerekre, amelyek állapotát a felhasználó képes lesz megváltoztatni. A programnak lehetőséget kell biztosítania a felhasználó számára a térképfájl betöltésére, majd a konténerek sérülési állapotának beállítására és a sérült konténerek lekérdezésére. A program továbbá lehetőséget biztosít a sérült konténerek helyreállítására, és a konténerek állapotának visszaállítására az eredeti állapotra.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamelyen "rossz" bemeneti paraméter esetén. Helytelen térképfájl formátum: A programnak ellenőriznie kell a betöltött térképfájl formátumát, és helyesen kell kezelnie az esetleges helytelen formátumot. Hibás bemenet kezelése: A programnak ellenőriznie kell a felhasználó által megadott bemenetet, és helyesen kell kezelnie az esetleges hibás bemeneteket, például a nem szám típusú bemeneteket. Helytelen állapotbeállítás: A programnak ellenőriznie kell az állapotbeállításokat, és helyesen kell kezelnie az esetleges helytelen állapotokat, például olyan állapotokat, amelyek nem léteznek a térképen. Nem megfelelő visszaállítás: A programnak helyesen kell kezelnie a visszaállítást, és biztosítania kell, hogy az eredeti állapotot helyreállítja minden manipuláció után.
- **Bemenet**

*operationLoadMapTest.txt*  
*manualDamageContainerAt1\_0*  
*manualDamageContainerAt2\_0*  
*listDamagedContainers*  
*manualResetContainerAt1\_0*  
*manualResetContainerAt2\_0*  
*listDamagedContainers*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*  
*The container at X:1, Y:0 was damaged successfully*  
*The container at X:2, Y:0 was damaged successfully*  
*There is a leaked pipe at X:1, Y:0*  
*There is a damaged pump at X:2, Y:0*  
*The container at X:1, Y:0 was reset to normal*  
*The container at X:2, Y:0 was reset to normal*  
*There are no damaged containers on the map*

### 8.2.19 Pályaelemek állapotának manuális manipulációja 2

- **Leírás:** Ez a funkció lehetővé teszi a felhasználó számára, hogy a pályán található elemek (Pl. csővezetékek, tartályok, szivattyúk stb.) állapotát manuálisan manipulálja. A felhasználó különböző parancsokat adhat ki a pályaelemekkel kapcsolatban, például sérültéssel, csúszóssággal vagy visszaállítással kapcsolatban.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamilyen "rossz" bemeneti paraméter esetén. A program képes betölteni egy tesztpályát, majd a felhasználó parancsait végrehajtani és megfelelően kezelní azokat. A program helyesen kezeli a helyes bemeneteket és azokra a megfelelő kimenetet adja vissza. A program a helytelen bemenetek esetén hibát jelez és megfelelő hibaüzenetet jelenít meg. Helytelen bemenetek esetén a program hibát jelezhet vagy nem megfelelő kimenetet adhat vissza.

- **Bemenet**

*operationLoadMapTest.txt*

*manualMakePipeSlipperyAt1\_0*

*manualMakePipeSlipperyAt1\_0*

*manualMakePipeSlipperyAt2\_0*

*listSlipperyPipes*

*manualResetContainerAt1\_0*

*listSlipperyPipes*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The container at X:1, Y:0 was successfully made slippery*

*The container at X:1, Y:0 was successfully made slippery again*

*The container at X:2, Y:0 cannot be made slippery, because the container is not a Pipe*

*There is a slippery pipe at X:1, Y:0*

*The container at X:1, Y:0 was reset to normal*

*There are no slippery pipes on the map*

### 8.2.20 Pályaelemek állapotának manuális manipulációja 3

- **Leírás:** Ez a feladat arra teszteli a rendszert, hogy helyesen tudjuk-e manuálisan manipulálni a pályaelemek állapotát. A bemenetként egy térképfájl és a különböző műveletek végrehajtása szerepel. Az elvárt kimenet a végrehajtott műveletek eredményeinek megfelelő kiírásokat tartalmazza.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnak valamelyen "rossz" bemeneti paraméter esetén. A rendszer helyesen tudja-e végreghajtani a különböző manuális műveleteket a pályaelemek állapotának manipulálására. Konkrétan a tesztnben ellenőrizzük, hogy helyesen tudjuk-e megváltoztatni egy cső elem állapotát ragadóssá, majd visszaállítani az eredeti állapotába. Ellenőrizzük továbbá, hogy a rendszer megfelelően kezeli az olyan műveleteket, amikor egy olyan elemet próbálunk manipulálni, ami nem alkalmas a kiválasztott műveletre. Az elvárt kimenetet a végrehajtott műveletek eredményei alapján határozzuk meg.

- **Bemenet**

*operationLoadMapTest.txt*  
*manualMakePipeStickyAt1\_0*  
*manualMakePipeStickyAt1\_0*  
*manualMakePipeStickyAt2\_0*  
*listStickyPipes*  
*manualResetContainerAt1\_0*  
*listStickyPipes*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The container at X:1, Y:0 was successfully made sticky*

*The container at X:1, Y:0 was successfully made sticky again*

*The container at X:2, Y:0 cannot be made sticky, because the container is not a Pipe*

*There is a sticky pipe at X:1, Y:0*

*The container at X:1, Y:0 was reset to normal*

*There are no sticky pipes on the map*

### 8.2.21 Játékos manuális áthelyezése egy tetszőleges pályaelemre

- **Leírás:** Ez a feladat arra teszteli a rendszert, hogy helyesen tudjuk-e manuálisan áthelyezni a játékost egy tetszőleges pályaelemre. A bemenetként egy térképfájl és a játékos áthelyezése során használt koordináták szerepelnek. Az elvárt kimenet a végrehajtott műveletek eredményeinek megfelelő kiírásokat tartalmazza.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamilyen "rossz" bemeneti paraméter esetén. A rendszernek helyesen kell végrehajtania az áthelyezési műveleteket, valamint kellő hibajelzésekkel kell kiírnia, ha a célmező nem érvényes (pl. nem járható mező, nincs konténer a mezőn stb.). Az áthelyezési műveletek során "hibák" léphetnek fel, ha a célmező nem érvényes. Emellett az is előfordulhat, hogy a játékos ki fog esni a térképről, ha a célmező a térkép határain kívülre esik. A fenti példa bemenetei azt mutatják, hogy a játékos a térkép bizonyos pozícióinak megadásával manuálisan áthelyezhető, de a célmezőnek meg kell felelnie bizonyos kritériumoknak (pl. járható mező, tartalmaz konténert stb.). Az elvárt kimenetek az áthelyezés eredményét és a jelenlegi játékos pozíciót tartalmazzák, valamint hibajelzést adnak ki, ha a célmező nem megfelelő.
- **Bemenet**

*operationLoadMapTest.txt*

*operationCreatePlayerSaboteurAt2\_1*

*listPlayer1Pos*

*manualTeleportPlayer1To2\_2*

*manualTeleportPlayer1To1\_2*

*manualTeleportPlayer1To0\_1*

*listPlayer1Pos*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*Saboteur was successfully created at X:2 , Y:1*

*Player1 is at X:2, Y:1*

*Player1 cannot be teleported to X:, Y:, because the target location is not steppable*

*Player1 cannot be teleported to X:1, Y:2, because there is no container at the target location*

*Player1 was successfully teleported to X:0, Y:1*

*Player1 is at X:0, Y:1*

### 8.2.22 Pályaelemek listázása

- **Leírás:** A pályaelemek listázása funkció arra szolgál, hogy kilistázza a jelenlegi pályán található összes konténert. A bemenetként megadott parancsok között szerepel az is, hogy előtte melyik térképfájlt kell betölteni. Az elvárt kimenet az összes pályaelem (konténerek) listája.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamilyen "rossz" bemeneti paraméter esetén. A program helyesen listázza ki az összes konténert a térképen. Sikertelen üzenetet kell kapnunk, ha nincs egyetlen konténer sem a térképen.

- **Bemenet**

*listContainers*

*operationLoadMapTest.txt*

*listContainers*

- **Elvárt kimenet**

*There are no containers on the map*

*Map Test.txt successfully loaded*

*There is a container at X:0, Y:0*

*There is a container at X:1, Y:0*

*There is a container at X:2, Y:0*

*There is a container at X:2, Y:1*

*There is a container at X:2, Y:2*

### 8.2.23 Pályaelemek közötti csatlakozás listázása

- **Leírás:** A pályaelemek közötti csatlakozások listázása a pálya elemek közötti kapcsolatokat mutatja meg. A bemenetként megadott térképfájl betöltése után a "listConnectedContainers" parancs kiírja a csatlakozó elemeket a pályán.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamelyen "rossz" bemeneti paraméter esetén. A program helyesen listázza ki az összekötés konténert a térképen. Sikertelen üzenetet kell kapnunk, ha nincs egyetlen konténer sem a térképen.

- **Bemenet**

*listConnectedContainers*

*operationLoadMapTest.txt*

*listConnectedContainers*

- **Elvárt kimenet**

*There are no containers on the map*

*Map Test.txt successfully loaded*

*The container at X:0, Y:0 is connected to the container at X:1, Y:0*

*The container at X:1, Y:0 is connected to the container at X:0, Y:0, and at X:2, Y:0*

*The container at X:2, Y:0 is connected to the container at X:1, Y:0, and at X:2, Y:1*

*The container at X:2, Y:1 is connected to the container at X:2, Y:0, and at X:2, Y:2*

*The container at X:2, Y:2 is connected to the container at X:2, Y:1*

### 8.2.24 Egy adott pályaelem közötti csatlakozás listázása

- **Leírás:** Ez a funkció a pályaelemek közötti csatlakozások listázását teszi lehetővé, koordinátok megadásával. A kimenet soronként felsorolja az összes csatlakozó elemet az azonosítója alapján.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várunk valamelyen "rossz" bemeneti paraméter esetén. A funkció helyesen listázza az összes csatlakozó pályaelementet az adott elemhez. Helyes hibaüzenetet kell adnia, ha a bemeneti azonosító nem szerepel a pályán.

- **Bemenet**

*operationLoadMapTest.txt*

*listConnectedContainersAt1\_0*

*listConnectedContainersAt2\_0*

*listConnectedContainersAt1\_2*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*The container at X:1, Y:0 is connected to the container at X:0, Y:0, and at X:2, Y:0*

*The container at X:2, Y:0 is connected to the container at X:1, Y:0, and at X:2, Y:1*

*There is no container at X:1, Y:2*

### 8.2.25 Összes játékos listázása

- **Leírás:** A teszt során a programnak a pályán elhelyezett játékosokat kell listálnia. Első lépésként a tesztnak három játékost hozunk létre, mindenki saboteurok, és az elvárt kimenetben az összes játékos pozícióját ki kell írni. A következő részben még egy saboteurt hozunk létre az (1,0) koordinátákon, és az elvárt kimenetben már ezt is ki kell írni a pozíciók mellett.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnak valamelyen "rossz" bemeneti paraméter esetén. A játékosok pozícióinak listázása.
- **Bemenet**

*operationLoadMapTest.txt*

*listPlayersPos*

*operationCreatePlayerSaboteurAt2\_0*

*operationCreatePlayerSaboteurAt2\_0*

*operationCreatePlayerSaboteurAt2\_1*

*listPlayersPos*

*operationCreatePlayerSaboteurAt1\_0*

*listPlayersPos*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*

*There are no players on the map*

*Saboteur was successfully created at X:2 , Y:0*

*Saboteur was successfully created at X:2 , Y:0*

*Saboteur was successfully created at X:2 , Y:1*

*Player1 is at X:2, Y:0*

*Player2 is at X:2, Y:0*

*Player3 is at X:2, Y:1*

*Saboteur was successfully created at X:1 , Y:0*

*Player1 is at X:2, Y:0*

*Player2 is at X:2, Y:0*

*Player3 is at X:2, Y:1*

*Player4 is at X:1, Y:0*

### 8.2.26 Pumpa tulajdonságainak listázása

- **Leírás:** A feladat arra vonatkozik, hogy egy adott pumpa tulajdonságait kell listázni, illetve a pumpa állapotát kell módosítani és ellenőrizni.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnak valamelyen "rossz" bemeneti paraméter esetén. A pumpa adott tulajdonságainak lekérdezése, a pumpa állapotának módosítása és ellenőrzése.
- **Bemenet**

*operationLoadMapTest.txt*  
*listPumpAt2\_1DamageTurn*  
*listPumpAt2\_0DamageTurn*  
*manualAdjustPumpAt2\_0ToDown*  
*listPumpAt2\_0Direction*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*  
*There is no pump at the target location*  
*The pump at X:2, Y:0 will be damaged in (randomly generated number) turns*  
*Successfully adjust pump at X:2, Y:0 output to Down*  
*The pump at X:2, Y:0 is pumping to Down*

### 8.2.27 Véletlen funkciók ki/bekapcsolása

- **Leírás:** A feladat egy olyan funkció tesztelése, amely lehetővé teszi a véletlen pumpa meghibásodás ki- és bekapcsolását. Az első kimenetben a teszt megmutatja, hogy az opció kikapcsolt állapotban van, és a pumpa nem fog meghibásodni véletlenszerű időpontokban. A második kimenetben a teszt megmutatja, hogy az opció bekapcsolt állapotban van, és a pumpa véletlenszerű időpontokban fog meghibásodni.
- **Ellenőrzött funkcionalitás, várható hibahelyek:** Ez a teszt nem tervez hibát az indításkor, de egyértelmű kimenetet várnak valamelyen "rossz" bemeneti paraméter esetén. Véletlen pumpa meghibásodás funkció helytelen működése esetén a teszt hibát eredményezhet. Azonban ha az opció helyesen működik, akkor nem várható hibahely.
- **Bemenet**

*operationLoadMapTest.txt*  
*randomPumpBreakdownTurnOff*  
*listPumpAt2\_0DamageTurn*  
*randomPumpBreakdownTurnOn*  
*listPumpAt2\_0DamageTurn*

- **Elvárt kimenet**

*Map Test.txt successfully loaded*  
*Pumps won't be damaged in a random turn*  
*The pump at X:2, Y:0 won't be damaged in (randomly generated number) turns*  
*Pumps will be damaged in a random turn*  
*The pump at X:2, Y:0 will be damaged in (randomly generated number) turns*

## 8.3 A tesztelést támogató programok tervezése

### Cél:

Célunk, hogy a tesztelést JUnit vagy terminal (cmd) segítségével ellenőrizni tudjuk a feladatok megoldásának helyességét. Ehhez készítünk egy teszttervet, amely leírja a tesztelés lépéseit és az elvárt eredményeket.

### Tesztek lefuttatása:

A tesztek lefuttatása a command line-ban történik. A tesztekhez szükséges bemeneti fájlokat a felhasználó adja meg parancssori argumentumként. A tesztek kimenetét a standard kimenetre írjuk ki.

### Tesztek tervezése

A tesztek tervezése során az alábbi lépésekkel járunk el:

#### 3.1 Bemenet előkészítése

A tesztekhez szükséges bemeneti fájlokat előre elkészítjük, vagy generáljuk.

#### 3.2 Tesztesetek definiálása

Definiáljuk a teszteseteket, amelyek az egyes funkciók működését ellenőrzik.

#### 3.3 Tesztesetek végrehajtása

Végrehajtjuk a teszteseteket, amelyek az elvárt eredményeket állítják elő.

#### 3.4 Tesztek eredményének kiértékelése

A tesztek eredményét összehasonlítjuk az elvárt eredménnyel, és megállapítjuk a tesztek eredményességét.

**Végre:** összehasonlítjuk a elvárt kimenetet a konzol kimenetével. Ha a teljes konzol kimenet megegyezik, akkor a teszt sikeres volt. Az alaposabb teszteléshez debagging-et végezhet, és megnézheti, hogy a szükséges objektumokat létrehozták/törölték-e a veremben

## Napló

Kezdet	Időtartam	Résztvevők	Leírás
2023.04.29. 12:30	0,5 óra	Réti Ádám Tisza Miklós Czifra Barnabás Molnár Márton Kopach Artem	Értekezlet: Feladatok kiosztása
2023.04.29. 13:00	5 óra	Réti Ádám	Tevékenység: 8.2 A tesztek bemenet/kimenet megírása a prototípus nyelvén
2023.04.29. 13:00	4 óra	Kopach Artem	Tevékenység: 8.2 A leírás, ellenőrzött funkcionálitás, várható hibahelyek megírása 8.3 A tesztelést támogató programok tervezének megírása
2023.04.29. 13:00	4 óra	Czifra Barnabás	8.1 Osztály leírások
2023.04.29. 13:00	4 óra	Molnár Márton	8.1 Container, Pump, Pipe, Cistern, MountainSpring leírása
2023.04.29. 13:00	7 óra	Tisza Miklós	8.1 Osztályok leírása, hozzájuk tartozó activity/állapot diagramok elkészítése

# Prototípus elkészítése

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

### Réti Ádám - csapatvezető

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)      ket.vill.adam@gmail.com  
(E1LX0J)      tisza.miklos.16@gmail.com  
(NX9GA4)      barna.czifra@gmail.com  
(KLM60O)      molnar.marton.hun@edu.bme.hu  
(JQBOLI)      kopach.artem@edu.bme.hu

2023.05.13

## 10. Prototípus beadása

A prototípustervezés és a kód a szoftverfejlesztésben használt eszközök, amelyekkel tesztelni és értékelni lehet egy jövőbeli program koncepcióját és funkcionalitását.

A prototípus a szoftver egy előzetes változata, amely alapvető funkciókat tartalmaz. A prototípus lehetővé teszi a fejlesztők és a felhasználók számára, hogy vizualizálják a jövőbeli szoftvert, és felmérjék, hogy az megfelel-e a követelményeknek és elvárásoknak. Ily módon a prototípus segít tisztázni és meghatározni a szoftverrel szemben támasztott követelményeket, mielőtt a szoftver végleges fejlesztése megkezdődne.

### 10.1 Fordítási és futtatási útmutató

#### 10.1.1 Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
List.java	27.1 KB	2023.05.11. 20:48	A pályaelemek különböző módon történő kilistázását megvalósító osztályt tartalmazza.
Main.java	2.6 KB	2023.05.11. 20:48	Tartalmazza a main metódust, ahol beolvassuk a testMap.txt nevű fájlból a teszpálya paramétereit.
Manual.java	11.4 KB	2023.05.11. 20:48	Ebben a fájlban található a manuális parancsokat megvalósító metódus. Segítségével tudunk manuálisan pályaelemeket létrehozni, a játékosok parancsait lefuttani illetve a vízfolyást lefuttatni.
Operation.java	27.8 KB	2023.05.11. 20:48	Ebben a fájlban valósulnak meg a különböző műveleteket (pl. loadMap) végrehajtó metódusok.
Playercmd.java	19.7 KB	2023.05.11. 20:48	A játékos által végrehajtható parancsokat (pl. adjustPump) végrehajtó metódus található itt.
Random.java	1.1 KB	2023.05.11. 20:48	Ebben a fájlban található az a metódus, amivel ki- és bekapcsolható a pumpák véletlenszerű elromlása.
StrFunctions.java	2.9 KB	2023.05.11. 20:48	A string műveleteket végrehajtó osztályt tartalmazza.

Cistern.java	8.1 KB	2023.04.16 12:00	A Cistern osztályt tartalmazó fájl. A Cistern logikája itt valósul meg.
Container.java	7.2 KB	2023.04.16 12:00	A Container osztály itt található, ebből származnak le a játéktér elemei(Cistern, Pump, Pipe, MountainSpring)
ContainerPos.java	2.6 KB	2023.05.11 20:48	A Containerek játéktéren való tárolásáért felelős osztály található itt.
MountainSpring.java	5.6 KB	2023.04.16 12:00	A MountainSpring osztály itt található.
Pipe.java	14.6 KB	2023.04.16 12:00	A Pipe osztály itt található.
Pump.java	10.7 KB	2023.04.16 12:00	A Pump osztályt tartalmazza.
Controller.java	5.0 KB	2023.04.16 12:00	A Controller osztályt tartalmazza, melynek feladata a játék vezérlésének megvalósítása.
MyException.java	168 B	2023.04.16 12:00	A MyException kivétel osztályt tartalmazza.
Map.java	8.6 KB	2023.04.16 12:00	A Map osztályt tartalmazza, melynek feladata a játékelemek egymáshoz csatlakoztatása, azok pályához adása és a fájlkezelés megvalósítása.
Mechanic.java	820 B	2023.04.16 12:00	A Mechanic osztályt tartalmazza.
Player.java	6.2 KB	2023.04.16 12:00	A Player osztályt tartalmazza, amely a Mechanic és a Saboteur osztályok ősosztálya.
Saboteur.java	750 B	2023.04.16 12:00	A Saboteur osztály tartalmazza.
Type.java	362 B	2023.04.16 12:00	A Type enumerációt tartalmazza, amely input vagy output lehet.

### 10.1.2 Fordítás és 10.1.3 Futtatás:

**Az alábbiak tartalmazzák a fordítási és futtatási tudnivalókat.**

**A projekt az alábbi GitHub repository-ban található:**

**GitHub link:** [https://github.com/kopach-artem/Pipes\\_gods\\_of\\_jar.git](https://github.com/kopach-artem/Pipes_gods_of_jar.git)

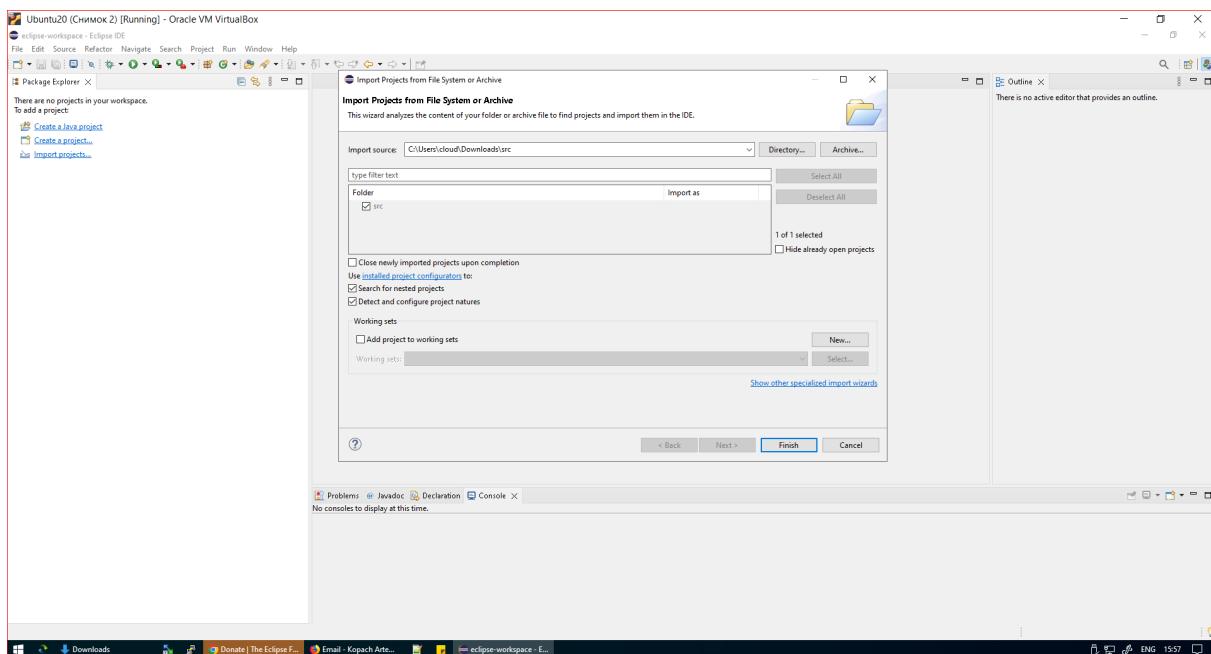
**Eclipse IDE:**

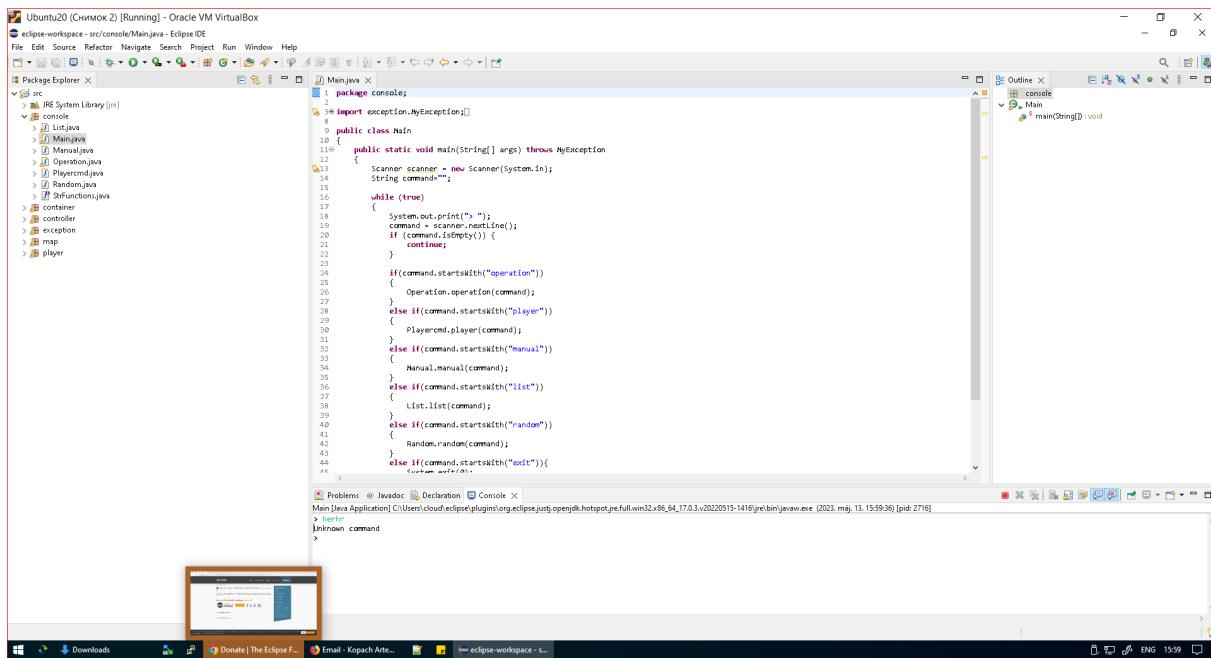
1. Töltsé le a projekt mappát, amely .java kiterjesztésű fájlokat tartalmaz, az alábbi módon: File -> Open Projects from File Systems -> Directory -> (csak ez kell)

válaszolni) src mappa. A projekt sikeres fordításához szükséges összes csomag (package) elérhető, beleértve a következőket: container, controller, exception, map, player és skeleton.

- Ha az Eclipse fejlesztői környezetben a "Futtatás" (Run) gombra kattintunk, a Java forráskódja automatikusan bytecode-vá (azaz .class fájlokká) fordul, majd a Java Virtual Machine (JVM) hajtja végre. Ez a folyamat a JDT (Java Development Tools) plugin segítségével történik a háttérben. Ha külön szeretnénk kezelní a fordítást, akkor először meg kell nyomni a "Build" gombot, majd miután a "Fordítás" (Build) sikeresen befejeződött, használhatjuk a "Futtatás" gombot.

Megjegyzés a 3. ponthoz: Az ezekben a csomagokban felsorolt fájlokat mind le kell fordítani és futtatni kell.





Saját tesztfuttatásunk az Eclipse IDE-vel Java fejlesztőknek - 2022-06:

## VM és CMD:

A megbízásban előírtak szerint a csapatunknak gondoskodnia kell arról, hogy a kódot a Linux operációs rendszer konzoljáról futtassuk a szokásos javac és java parancsok segítségével. Ezt a lehetőséget sikeresen teszteltük és megvalósítottuk a [https://niif.cloud.bme.hu/ "Windows 10 20H2 - JDK-Eclipse-WSU"](https://niif.cloud.bme.hu/) segítségével.

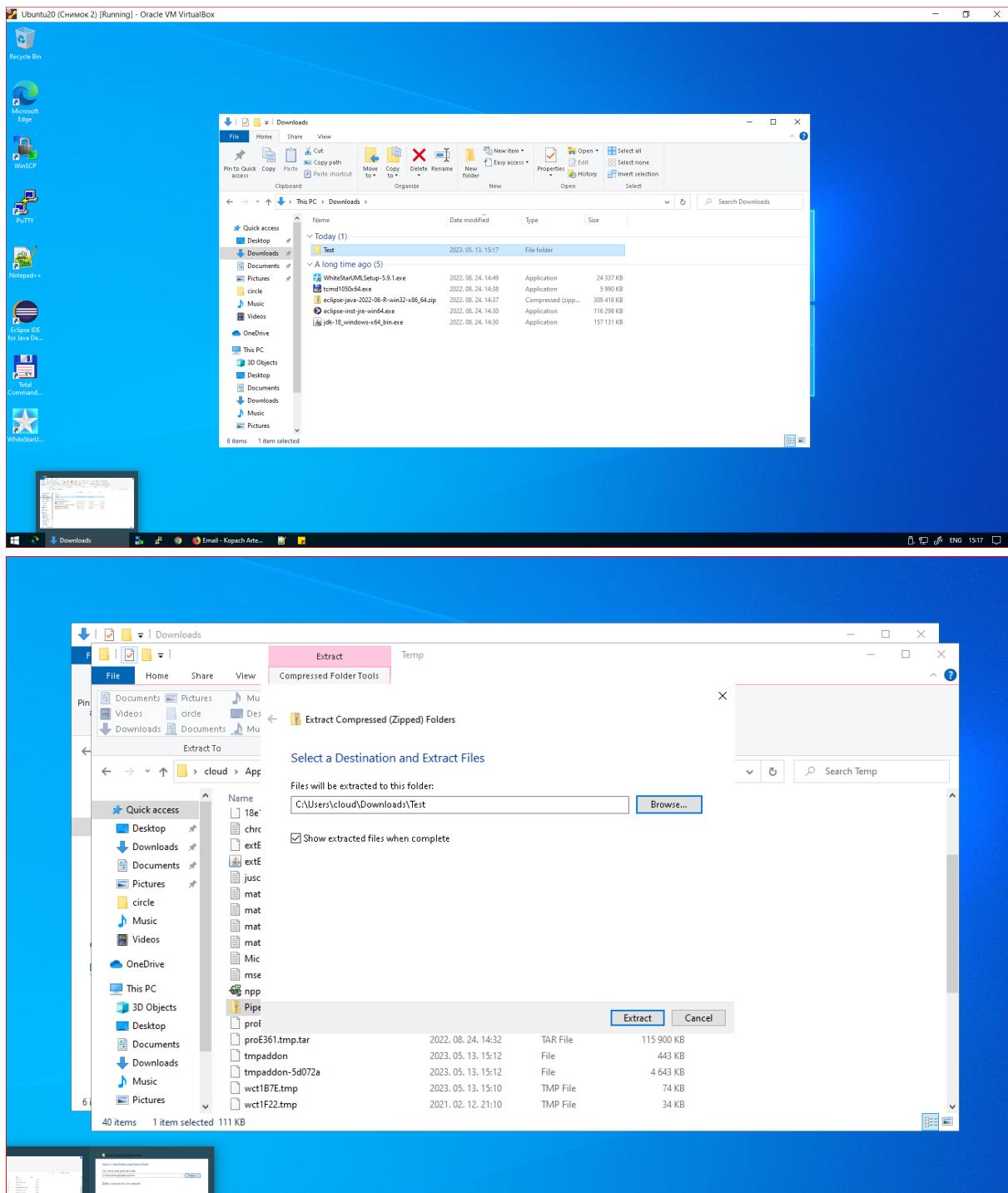
A kódot az rdesktop vm.niif.cloud.cloud.bme.hu:17760 -u cloud -p 8hr7AiF2B8 -f parancssal lehet futtatni a Linux rendszerek termináljáról (Mi például Ubuntu 20.04-et használtunk). Azonban a művelet végrehajtható a Windows rendszerből is.

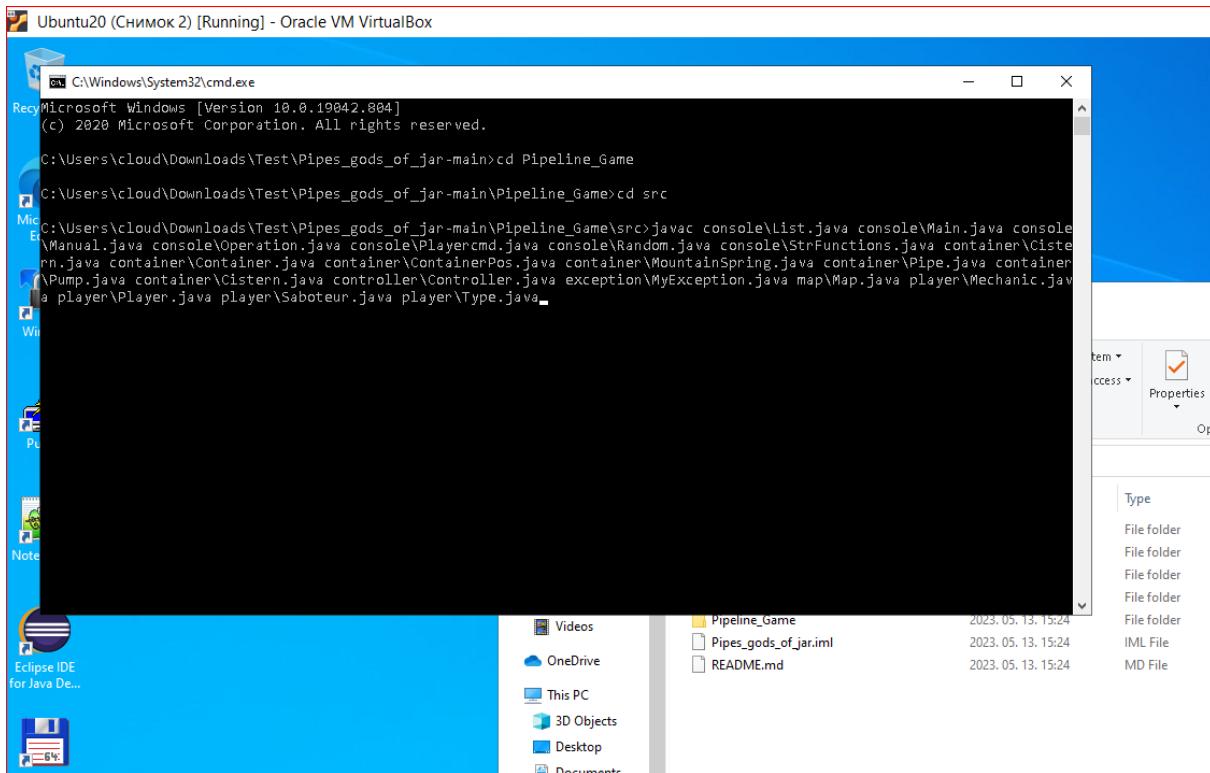
Először klónozzuk a kódunkat erről a linkről a számítógépünkre, vagy töltük le a .zip archívumot. Ezután navigálunk a projekt mappába, nyissuk meg a parancssort, majd a következő útvonalon használjuk a javac parancsot a fordításhoz:

A tesztmappát ott hoztam létre, ahol a projektet kicsomagoltam, de egyszerűen kicsomagolhatod a mappát is.

## 10. Prototípus beadása

gods\_of\_jar



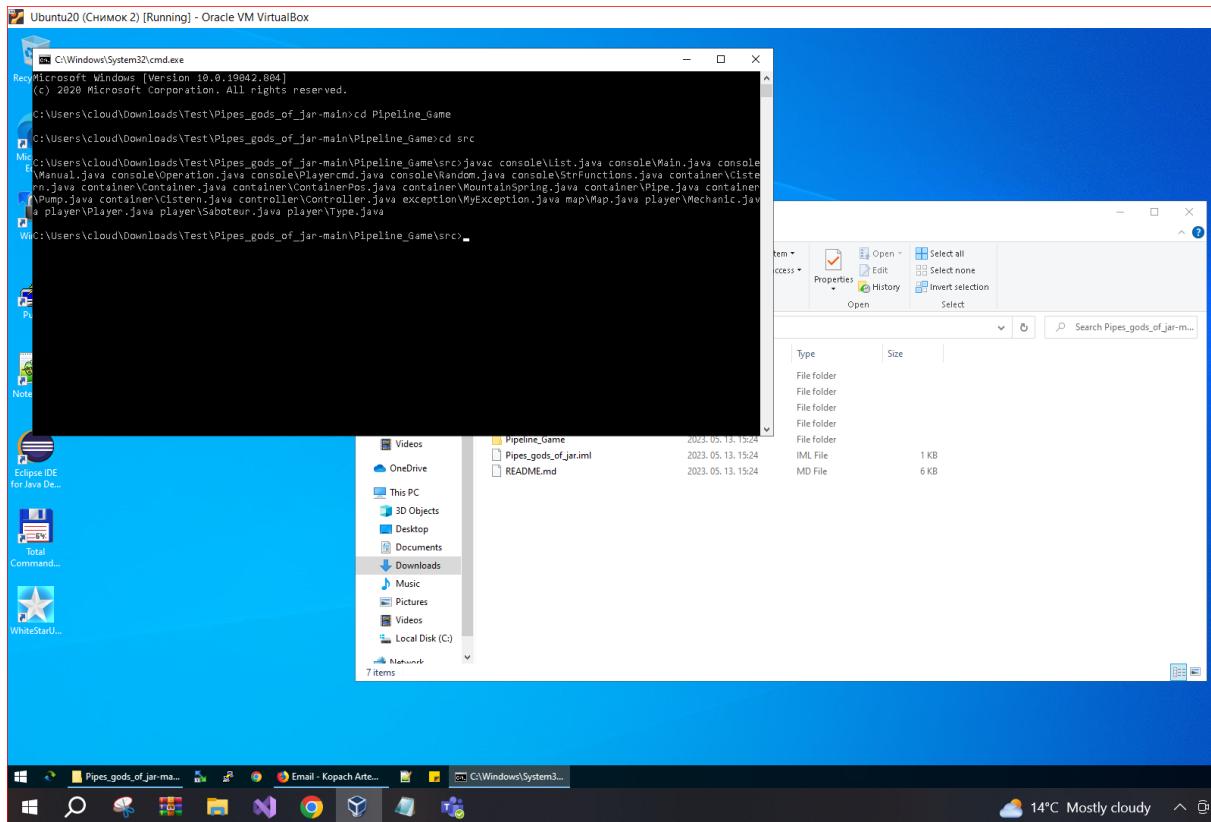


Ezek a parancsok kell fordításra:

1. C:\Users\cloud\Downloads\Test\Pipes\_gods\_of\_jar-main>cd Pipeline\_Game
2. C:\Users\cloud\Downloads\Test\Pipes\_gods\_of\_jar-main\Pipeline\_Game>cd src
3. C:\Users\cloud\Downloads\Test\Pipes\_gods\_of\_jar-main\Pipeline\_Game>javac console\List.java console\Main.java console\Manual.java  
console\Operation.java console\Playercmd.java console\Random.java  
console\StrFunctions.java container\Cistern.java container\Container.java  
container\ContainerPos.java container\MountainSpring.java container\Pipe.java  
container\Pump.java container\Cistern.java controller\Controller.java  
exception\MyException.java map\Map.java player\Mechanic.java  
player\Player.java player\Saboteur.java player\Type.java

## 10. Prototípus beadása

godz\_of\_jar



ha semmi utana konzolban ez jelent, hogy minden rendben, minden fordító sikerül és most meg van .class fájlok

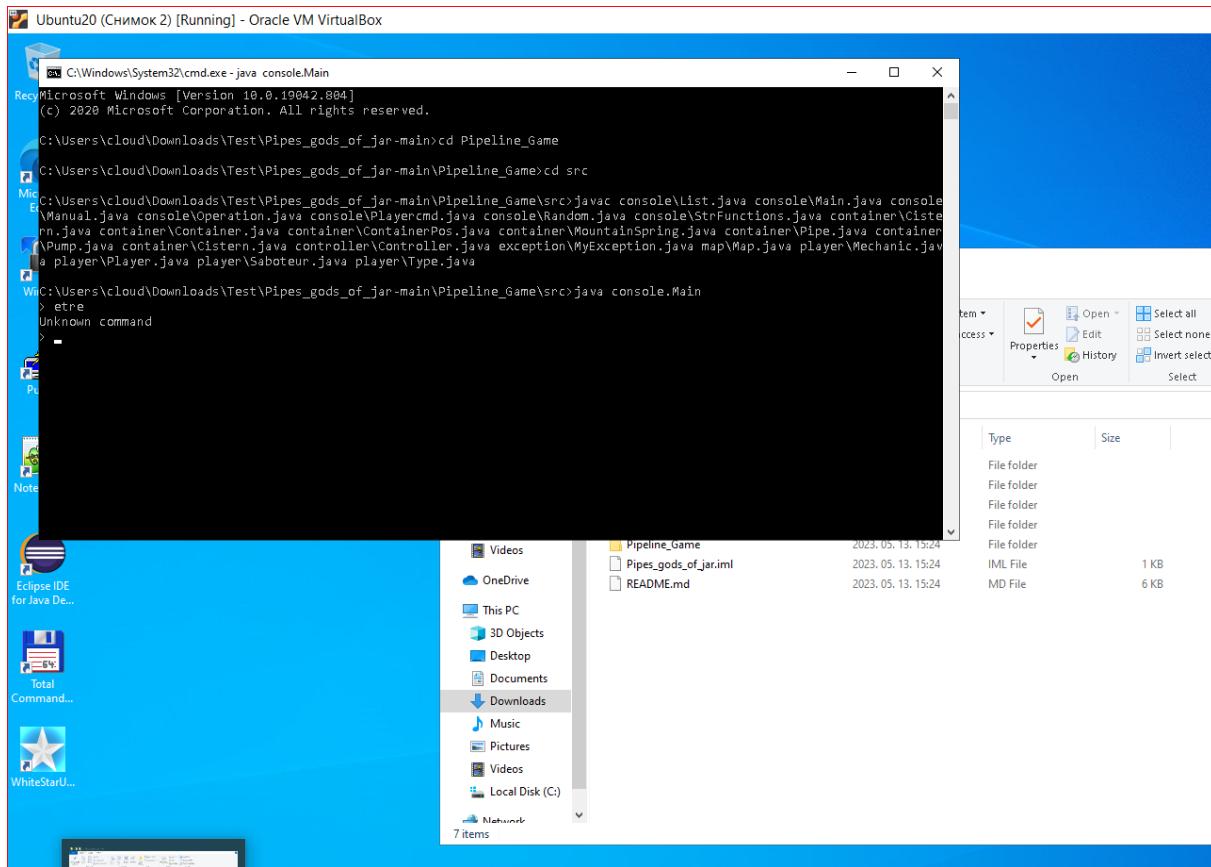
Miután a fordítási folyamat befejeződött, futtassuk a Java Virtual Machine (JVM) programot a java parancs segítségével, amelyet a fő metódust tartalmazó osztályfájl neve követ, az esetünkben a Main. Ez létrehozza a bináris futtatható kódot, amely bármely olyan rendszeren futtatható, amelyen kompatibilis JVM van telepítve.

Ezután futtassuk az elkészült fájlt a következő parancssal:

**java console.Main**

## 10. Prototípus beadása

*gods\_of\_jar*



itt láttunk, hogy Main osztály sikeres futat

Ide tudunk írni a parancsokat. Most csinálunk csak gyors teszt, ahol rossz parancsot írtunk.

**Megjegyzés:**

Fontos megjegyezni, hogy a 2. követelményben leírt környezetet biztosítjuk, amely magában foglalja az Eclipse IDE használatát és a Windows parancssorát a javac és java parancsokkal, amelyek az egyetlen szükséges eszközök a bináris futtatható kód forrásfájlokból történő generálásához. A kód kompatibilitásának és hordozhatóságának biztosítása érdekében minden más kiegészítő eszközt vagy függőséget kerülni kell.

Szinte bármilyen fejlesztői környezethben futtathatod, mert a csapatunk készített egy praktikus GitHub tárolót, ahol láthatod a kódváltoztatási fát, és könnyen klónozhatsz a számítógépedre.

## **10.2 Tesztek jegyzőkönyvei**

**Komment:**

A szoftver prototípusában található tesztforgatókönyvek lehetővé teszik a különböző funkciók működésének tesztelését és a program korai szakaszban történő hibakeresését. Ez segít a lehetséges problémák azonosításában és kijavításában, még a szoftver teljes kifejlesztése előtt. A tesztforgatókönyvek segítenek továbbá a program teljesítményének értékelésében, működésének optimalizálásában és a felhasználói felület javításában.

Ebben a tesztleírásban rámutatunk a változtatásainkra és az elvárás-változatosság-hibapontra.

Teszt interfaceről: Ezeket a teszteket tervezünk konzolban. Miért? Ez egy jó kérdés!

**Ahogy én tudom, Java előadásokban volt teszt téma is ott volt ne csak JUnit, hanem manuális konzol tesztek, amelyikben lehet kiírni elvárásokat default user printtel (Javaban System.out.println("vmit") ) if álapokkal (statement).**

**Mi láttunk, hogy ebben az esetben ez nekünk ugyanaz és írtunk konzol teszteket.**

### **Teszteset0**

**Komment:** Persze minden tesztnak lehet rossz parancs, vagy rossz betű parancsban. Ebben az esetben mindennek konzolban lesz Unknown command.

### **Teszteset1**

<b>Teszteleő neve</b>	<b>Kopach Artem (Incializálás)</b>
<b>Teszt időpontja</b>	2023.05.14 -> Első teszt
<b>Bement</b>	operationCreateTestMap
<b>Teszt eredménye</b>	Map creation started, will be saved as testMap.txt Test map has successfully been created as 'testMap.txt', you can load it with command 'operationLoadMap testMap.txt'
<b>Lehetséges hibaok</b>	IOException, de nekünk ott van try-catch blokk, amelyik detect ezt. Ezenkívül, nekünk van esete, ha nincs map "maps" és nincs fájl, akkor létrejönnek directoryban
<b>Változtatások</b>	Kicsit változtunk kimenet, most kisebb, de ugyanaz informatív

**FONTOS**

**Komment:** Utána nekünk kicsi kimeneti logiká kicsi változta, mert vége erderedmény tudunk nézni nem csak kiírás üzenetekben, hanem **operationPrintMap**, ahol fogunk rajzolni konzolban egész Map minden Containerekkel. Vagy fogunk list Player Postionnal nézni

**Teszeset2**

<b>Teszteleő neve</b>	<b>Kopach Artem - Pályaelem törlés</b>
<b>Teszt időpontja</b>	2023.05.14 -> Második teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. operationLoadMapTestMap.txt</li> <li>2. <i>operationDeleteContainerAt0_2</i></li> <li>3. <i>operationDeleteContainerAt0_0</i></li> <li>4. operationPrintMap</li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. (Térkép, ahol lehet nézni mi is hol delete)</li> </ol>
<b>Lehetséges hibaok</b>	Map load hiba, az nekünk van kiírva, hogy Map nem üres lesz. Nincs úgy container Mapban, amelyik koordináta irta, ez is nézünk for - if állapotban
<b>Változtatások</b>	-

**Teszeset3**

<b>Teszteleő neve</b>	<b>Kopach Artem - Nem megengedett pályaépítés</b>
<b>Teszt időpontja</b>	2023.05.14 -> Hármodik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreateContainerPipeAt0_0</i></li> <li>3. <i>operationCreateContainerMountainSpringAt0_1</i></li> <li>4. <i>operationCreateContainerPipeAt1_1</i></li> <li>5. <i>operationCreateContainerBuildingAt1_2</i></li> <li>6. <i>operationConnectContainerAt1_1ToContainerAt0_1</i></li> <li>7. <i>operationConnectContainerAt1_1ToContainerAt2_2</i></li> <li>8. <i>operationPrintMap</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. There's already a container stationed at:0,0 position</li> <li>3. -</li> <li>4. -</li> <li>5. Invalid type for container</li> <li>6. -</li> <li>7. -</li> </ol>

	8. (Térkép, ahol lehet találni minden Container, amelyik cinálta)
<b>Lehetséges hibaok</b>	Tudunk nem nézni Create vmelyik Container, ha már van Mapon ebben a koordinátában. Connected probléma lehet, hogy ott ugyanaz a típus vagy nem szomszédai. minden hibákat elnélünk if - else állapotban
<b>Változtatások</b>	Kicsit más kimenete, de ugyanaz a jelentés

**Teszeset4**

<b>Tesztele neve</b>	<b>Kopach Artem - Játékosok létrehozása</b>
<b>Teszt időpontja</b>	2023.05.14 -> Negyedik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreatePlayerBuilderAt1_0</i></li> <li>3. <i>operationCreatePlayerMechanicAt1_0</i></li> <li>4. <i>operationCreatePlayerMechanicAt0_0</i></li> <li>5. <i>operationCreatePlayerMechanicAt0_2</i></li> <li>6. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. <i>Invalid type for Container</i></li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. <i>Player1 is at X: 4 Y:0</i>  <i>Player2 is at X: 4 Y:0</i>  <i>Player3 is at X: 1 Y:0</i>  <i>Player4 is at X: 0 Y:0</i></li> </ol>
<b>Lehetséges hibaok</b>	Ellenőriznünk kell a steppable koordinátáit is, hogy van-e olyan eleme, amire rá tudunk állítani egy játékost.
<b>Változtatások</b>	Kicsi kimeneti változások vannak, de jelentés ugyanaz

**FONTOS**

**Komment:** Ezután a programunkban vannak hívások a játékosok mozgatására a térképen, de az akciók után nem mindig látjuk a kívánt eredményt a listPlayersPos-ban, hogy miért? Ez egy nagyon jó kérdés, ez lesz az első teszt, próbáljuk meg megváltoztatni a játékosok pozícióját a térképen.

**ProbaTeszt\***

Tesztelő neve	Kopach Artem - Játékosok mozgatása
Teszt időpontja	2023.05.14 -> Ötödik teszt
Bemenet	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>listPlayersPos</i></li> <li>3. <i>player1moveToRight</i></li> <li>4. <i>listPlayersPos</i></li> <li>5. <i>player1moveToLeft</i></li> <li>6. <i>listPlayersPos</i></li> </ol>
Teszt eredménye	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. <i>Player1 is at X: 4 Y:0</i> <i>Player2 is at X: 4 Y:0</i></li> <li>3. -</li> <li>4. <i>Player1 is at X: 4 Y:0</i> <i>Player2 is at X: 4 Y:0</i></li> <li>5. -</li> <li>6. <i>Player1 is at X: 3 Y:0</i> <i>Player2 is at X: 4 Y:0</i></li> </ol>
Lehetséges hibaok	Arra is figyelnünk kell, hogy a konténer lehet maximális, ha úgy, akkor a konténerbe nem lehet mozogni is.
Változtatások	Kicsi kimeneti változások vannak, de jelentés ugyanaz

**Komment:** HELYES! Miért, azért mert jobb nem volt Container, ahova Player tud lépni. Az is lesz jó, ha Container van maximum Player ezen és stb.

**FONTOS****Komment:**

**Megjegyzés:** ha nyomon akarod követni a dolgokat, javasoljuk, hogy a lehető leggyakrabban használd a *listPlayersPos* és az *operationPrintMap* parancsokat.

**Teszeset5**

Tesztelő neve	Kopach Artem - Játékosok mozgatása
Teszt időpontja	2023.05.14 -> Ötödik teszt
Bemenet	<ol style="list-style-type: none"> <li>7. <i>operationLoadMapTest.txt</i></li> <li>8. <i>operationCreatePlayerMechanicAt1_0</i></li> <li>9. <i>operationCreatePlayerSaboteurAt2_1</i></li> <li>10. <i>player0moveToLeft</i></li> <li>11. <i>player1moveToLeft</i></li> <li>12. <i>player1moveToSouth</i></li> <li>13. <i>player1moveToRight</i></li> <li>14. <i>player1moveToDown</i></li> <li>15. <i>player2moveToUpper</i></li> <li>16. <i>listPlayersPos</i></li> </ol>

<b>Teszt eredménye</b>	7. Loading map from file: testMap.txt 8. - 9. - 10. - 11. - 12. <i>Invalid direction</i> 13. - 14. - 15. - 16. <i>kimeneti lista Player Positionnal</i>
<b>Lehetséges hibaok</b>	Arra is figyelnünk kell, hogy a konténer lehet maximális, ha úgy, akkor a konténerbe nem lehet mozogni is.
<b>Változtatások</b>	Kicsi kimeneti változások vannak, de jelentés ugyanaz

**Teszteset6**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos pumpa átállítás</b>
<b>Teszt időpontja</b>	2023.05.14 -> Hatodik teszt
<b>Bemenet</b>	1. <i>operationLoadMapTest.txt</i> 2. <i>operationCreatePlayerMechanicAt1_0</i> 3. <i>player1AdjustPumpOutputToUp</i> 4. <i>player1moveToRight</i> 5. <i>player1AdjustPumpOutputToUp</i> 6. <i>operationPrintMap</i>
<b>Teszt eredménye</b>	1. Loading map from file: testMap.txt 2. - 3. - 4. - 5. <i>Térkép lista</i>
<b>Lehetséges hibaok</b>	Ha azonban a játékos egy olyan tárolóhoz megy, amelyben az adott szivattyú van, vagy kiválaszt egy olyan szivattyút, amelyet befolyásolni tud.
<b>Változtatások</b>	Kicsi kimeneti változások vannak, de jelentés ugyanaz

**Teszteset7**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos csőlyukasztás</b>
<b>Teszt időpontja</b>	2023.05.14 -> Hetedik teszt
<b>Bemenet</b>	1. <i>operationLoadMapTest.txt</i> 2. <i>operationCreatePlayerMechanicAt2_0</i> 3. <i>player1LeakPipe</i> 4. <i>player1moveToLeft</i> 5. <i>player1LeakPipe</i> 6. <i>operationPrintMap</i> 7. <i>listPlayersPos</i>

<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. <i>Térkép</i></li> <li>7. <i>listPlayersPos</i>  <i>Player1 is at X: 3 Y:0</i>  <i>Player2 is at X: 4 Y:0</i>  <i>Player3 is at X: 2 Y:0</i></li> </ol>
<b>Lehetséges hibaok</b>	Például, ha a játékos nem azon a csövön van, amelyet a lyukasztáshoz választott.
<b>Változtatások</b>	Kicsi kimeneti változások vannak, de jelentés ugyanaz

**Teszteset8**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos pumpa illesztés</b>
<b>Teszt időpontja</b>	2023.05.14 -> Nyolcadik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreatePlayerMechanicAt1_0</i></li> <li>3. <i>manualCreateContainerPipeAtPlayer1</i></li> <li>4. <i>player1AttachPipeTo4_0</i></li> <li>5. <i>player1moveToRight</i></li> <li>6. <i>player1AttachPipe</i></li> <li>7. <i>operationPrintMap</i></li> <li>8. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. <i>Térkép</i></li> <li>8. <i>Players Pos list</i></li> </ol>
<b>Lehetséges hibaok</b>	Például, ha a játékos megpróbál egy csövet csatlakoztatni, miközben rossz pozícióban van.
<b>Változtatások</b>	Kimeneti logika.

**Teszteset9**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos cső leszerelés</b>
<b>Teszt időpontja</b>	2023.05.14 -> Kilencedik teszt

<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreatePlayerMechanicAt2_0</i></li> <li>3. <i>manualCreateContainerPumpAtPlayer1</i></li> <li>4. <i>player1AttachPump</i></li> <li>5. <i>player1movetoLeft</i></li> <li>6. <i>player1AttachPump</i></li> <li>7. <i>operationPrintMap</i></li> <li>8. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. <i>Térkép</i></li> <li>8. <i>Players Pos list</i></li> </ol>
<b>Lehetséges hibaok</b>	Például, ha a játékos megpróbál egy pumpát csatlakoztatni, miközben rossz pozícióban van..
<b>Változtatások</b>	Kimeneti logika

**Teszteset10**

<b>Teszteleő neve</b>	<b>Kopach Artem - Játékos csőjavítás</b>
<b>Teszt időpontja</b>	2023.05.14 -> Tizedik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreatePlayerMechanicAt2_0</i></li> <li>3. <i>player1DetachPipe</i></li> <li>4. <i>player1movetoDown</i></li> <li>5. <i>player1DetachPipe</i></li> <li>6. <i>operationPrintMap</i></li> <li>7. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. <i>Térkép</i></li> <li>7. <i>Players Pos list</i></li> </ol>
<b>Lehetséges hibaok</b>	Például, ha egy játékos leválaszt egy csövet, ha a játékos helyzete rossz.Ha a játékos a helyes pozícióba mozog, és megpróbálja megismételni a műveletet, akkor sikerülnie kell.
<b>Változtatások</b>	Kimeneti logika

**Teszteset11**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos pumpa javítás</b>
<b>Teszt időpontja</b>	2023.05.14 -> Tizenegyedik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>manualDamageContainerAt1_0</i></li> <li>3. <i>operationCreatePlayerMechanicAt2_0</i></li> <li>4. <i>player1RepairPipe</i></li> <li>5. <i>player1moveToLeft</i></li> <li>6. <i>player1RepairPipe</i></li> <li>7. <i>player1RepairPipe</i></li> <li>8. <i>operationPrintMap</i></li> <li>9. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. -</li> <li>8. <i>Térkép</i></li> <li>9. <i>Players Pos list</i></li> </ol>
<b>Lehetséges hibaok</b>	Ha például egy játékos megpróbál megjavítani egy olyan csövet, amelyen nem áll
<b>Változtatások</b>	Kimeneti logika

**Teszteset12**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos cső csúszósítása</b>
<b>Teszt időpontja</b>	2023.05.14 -> Tizenkettizedik Teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>manualDamageContainerAt2_0</i></li> <li>3. <i>operationCreatePlayerMechanicAt1_0</i></li> <li>4. <i>player1RepairPump</i></li> <li>5. <i>player1moveToRight</i></li> <li>6. <i>player1RepairPump</i></li> <li>7. <i>player1RepairPump</i></li> <li>8. <i>operationPrintMap</i></li> <li>9. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. -</li> <li>8. <i>Térkép</i></li> <li>9. <i>Players Pos list</i></li> </ol>

<b>Lehetséges hibaok</b>	Ha a játékos sikeresen feláll a megfelelő pumpára, és újra megpróbálja megjavítani, a javításnak sikerülnie kell.
<b>Változtatások</b>	Kimeneti logika

**Teszeset13**

<b>Tesztelő neve</b>	<b>Kopach Artem - Játékos cső ragadóssá tétele</b>
<b>Teszt időpontja</b>	2023.05.14 -> Tizenharmadik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>manualDamageContainerAt2_0</i></li> <li>3. <i>operationCreatePlayerMechanicAt1_0</i></li> <li>4. <i>player1RepairPump</i></li> <li>5. <i>player1moveToRight</i></li> <li>6. <i>player1RepairPump</i></li> <li>7. <i>player1RepairPump</i></li> <li>8. <i>operationPrintMap</i></li> <li>9. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. -</li> <li>8. <i>Térkép</i></li> <li>9. <i>Players Pos list</i></li> </ol>
<b>Lehetséges hibaok</b>	
<b>Változtatások</b>	kimeneti logika

**Teszeset14**

<b>Tesztelő neve</b>	<b>Kopach Artem - Pályaelem manuális létrehozása ciszternánál</b>
<b>Teszt időpontja</b>	2023.05.14 -> Tizenegyedik teszt
<b>Bemenet</b>	<ol style="list-style-type: none"> <li>1. <i>operationLoadMapTest.txt</i></li> <li>2. <i>operationCreatePlayerSaboteurAt2_0</i></li> <li>3. <i>player1MakePipeSticky</i></li> <li>4. <i>player1moveToLeft</i></li> <li>5. <i>player1MakePipeSticky</i></li> <li>6. <i>player1MakePipeSticky</i></li> <li>7. <i>operationPrintMap</i></li> <li>8. <i>listPlayersPos</i></li> </ol>
<b>Teszt eredménye</b>	<ol style="list-style-type: none"> <li>1. Loading map from file: testMap.txt</li> <li>2. -</li> <li>3. -</li> <li>4. -</li> <li>5. -</li> <li>6. -</li> <li>7. <i>Térkép</i></li> </ol>

	<i>8. Players Pos list</i>
<b>Lehetséges hibaok</b>	Ha egy játékos nem egy csőn áll, akkor nem tudja megcsinálni a ragadóssát.
<b>Változtatások</b>	Kimeneti logika

**Teszteset15**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelem manuális létrehozása játékosnál</b>
<b>Teszt időpontja</b>	2023.05.14 16:33
<b>Bemenet</b>	<i>operationLoadMapTest.txt operationCreatePlayerSaboteurAt2_0 manualCreateContainerPipeAtPlayer1 manualCreateContainerPipeAtPlayer1 manualCreateContainerPumpAtPlayer1 manualCreateContainerPumpAtPlayer1</i>
<b>Teszt eredménye</b>	<i>A player sikeresen létrejött, a csövek és pumpa is sikeresen létrejött</i>

**Teszteset17**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Játék kör számának manuális beállítása</b>
<b>Teszt időpontja</b>	2023.05.14 16:51
<b>Bemenet</b>	<i>manualSetTurnCount5 listCurrentTurn</i>
<b>Teszt eredménye</b>	<i>A program sikeresen beállította a turnCount-ot.</i>

**Teszteset18**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelemek állapotának manuális manipulációja 1</b>
<b>Teszt időpontja</b>	2023.05.14 18:30
<b>Bemenet</b>	<i>operationLoadMapTest.txt manualDamageContainerAt1_0 manualDamageContainerAt2_0 listDamagedContainers</i>
<b>Teszt eredménye</b>	<i>A program sikeresen elrontotta a kiválasztott Containereket..</i>

**Teszteset19**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelemek állapotának manuális manipulációja 2</b>
<b>Teszt időpontja</b>	2023.05.14 18:32
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt manualMakePipeSlipperyAt1_0 manualMakePipeSlipperyAt2_0 listSlipperyPipes</i>
<b>Teszt eredménye</b>	<i>A program sikeresen csúszóssé tette az 1_0 koordinátában lévő csövet. A 2_0 koordinátában nem csinált semmit, mivel ott nem cső található..</i>

**Teszteset20**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelemek állapotának manuális manipulációja 3</b>
<b>Teszt időpontja</b>	2023.05.14 18:34
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt manualMakePipeStickyAt1_0 manualMakePipeStickyAt2_0 listStickyPipes</i>
<b>Teszt eredménye</b>	<i>A program sikeresen ragadóssé tette az 1_0 koordinátában lévő csövet. A 2_0 koordinátában nem csinált semmit, mivel ott nem cső található..</i>

**Teszteset21**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Játékos manuális áthelyezése egy tétszőleges pályaelemre</b>
<b>Teszt időpontja</b>	2023.05.14 18:36
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt operationCreatePlayerSaboteurAt2_1 manualTeleportPlayer1To0_1 listPlayer1Pos</i>
<b>Teszt eredménye</b>	<i>A program sikeresen áthelyezte a játékost a kívánt pozícióba.</i>

**Teszteset22**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelemek listázása</b>
<b>Teszt időpontja</b>	2023.05.14 18:41
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt listContainers</i>
<b>Teszt eredménye</b>	<i>A program sikeresen kilistázta a pályán található Containereket és pozíciójukat.</i>

**Teszteset23**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pályaelemek közötti csatlakozás listázása</b>
<b>Teszt időpontja</b>	2023.05.14 18:43
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt listConnectedContainers</i>
<b>Teszt eredménye</b>	<i>A program sikeresen kilistázta a pályán található Containereket, hogy mely másik Containerekhez csatlakoznak és pozíciójukat.</i>

**Teszteset24**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Egy adott pályaelem csatlakozásainak listázása</b>
<b>Teszt időpontja</b>	2023.05.14 18:44
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt listConnectedContainersAt1 0</i>
<b>Teszt eredménye</b>	<i>A program sikeresen kilistázta a pályán a megadott pozícióban található Containereket és pozíciójukat.</i>

**Teszteset25**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Összes játékos listázása</b>
<b>Teszt időpontja</b>	2023.05.14 18:46

<b>Bemenet</b>	<i>operationLoadMaptestMap.txt</i> <i>listPlayersPos</i> <i>operationCreatePlayerSaboteurAt2_1</i> <i>listPlayersPos</i>
<b>Teszt eredménye</b>	<i>A program sikeresen kilistázta a pályán található játékosokat és pozíciójukat. Új játékos hozzáadása után is sikeresen működött.</i>

**Teszteset26**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Pumpa tulajdonságainak listázása</b>
<b>Teszt időpontja</b>	2023.05.14 18:48
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt</i> <i>listPumpAt2_0DamageTurn</i> <i>listPumpAt2_0Direction</i>
<b>Teszt eredménye</b>	<i>A program sikeresen kilistázta, hogy a pumpa mely körben fog elromlani és hogy melyik irányba van állítva.</i>

**Teszteset27**

<b>Tesztelő neve</b>	Molnár Márton
<b>Teszteset neve</b>	<b>Véletlen funkciók ki/bekapcsolása</b>
<b>Teszt időpontja</b>	2023.05.14 18:50
<b>Bemenet</b>	<i>operationLoadMaptestMap.txt</i> <i>randomPumpBreakdownTurnOff</i> <i>listPumpAt2_0DamageTurn</i> <i>randomPumpBreakdownTurnOn</i>
<b>Teszt eredménye</b>	<i>A program sikeresen ki- illetve bekapcsolta a pumpák véletlenszerű elromlását.</i>

**10.3 Értékelés**

Tag neve	Tag neptun	Munka százalékban
Tisza Miklós	E1LX0J	30%
Réti Ádám	AO7JX4	26%

Kopach Artem	JQBOLI	19%
Czifra Barnabás	NX9GA4	11%
Molnár Márton	KLM60O	14%

## ***10.4 Napló***

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résztvevők</b>	<b>Leírás</b>
2023.05.13 12:00	0,5 óra	Tisza Miklós Réti Ádám Kopach Artem Czifra Barnabás Molnár Márton	Értekezlet.
2023.05.13 12:30	2,5 óra	Molnár Márton	Tevékenység: 10.1 Fájlok leírása Kommentezés
2023.05.10-2023.05.14	29 óra	Réti Ádám Tisza Miklós	Értekezlet: Ádám elkészíti a bemeneti nyelvet Miklós a megvalósított függvényeket a bemeneti nyelvvel együtt összegyűrja
2023.05.14 16:00	10,5 óra	Molnár Márton	Tevékenység: 15-27 tesztelés elvégzése Kommentezés
2023.05.13 12:30	6 óra	Czifra Barnabás	Értekezlet: Futtatás, fordítás Kommentezés
2023.05.12 - 2023.05.13 12:00	13 óra	Kopach Artem	Tevékenység:  kódban: Random osztály, Operation osztály segítettem, hibákat kerestem is. Kommenteket.  doksi: 10.1 komment, vége fájlok méretei, 10.2 EGÉSZ, hibái javítottam , 10.3 kommentek, 1 - 14 tesztek

# Grafikus felület specifikációja

77 – gods\_of\_jar

Konzulens:  
Vörös András

## Csapattagok

**Réti Ádám -  
csapatvezető**

Tisza Miklós  
Czifra Barnabás  
Molnár Márton  
Kopach Artem

(AO7JX4)	ket.vill.adam@gmail.com
(E1LX0J)	tisza.miklos.16@gmail.com
(NX9GA4)	barna.czifra@gmail.com
(KLM60O)	molnar.marton.hun@edu.bme.hu
(JQBOLI)	kopach.artem@edu.bme.hu

2023.05.13

## 11. Grafikus felület specifikációja

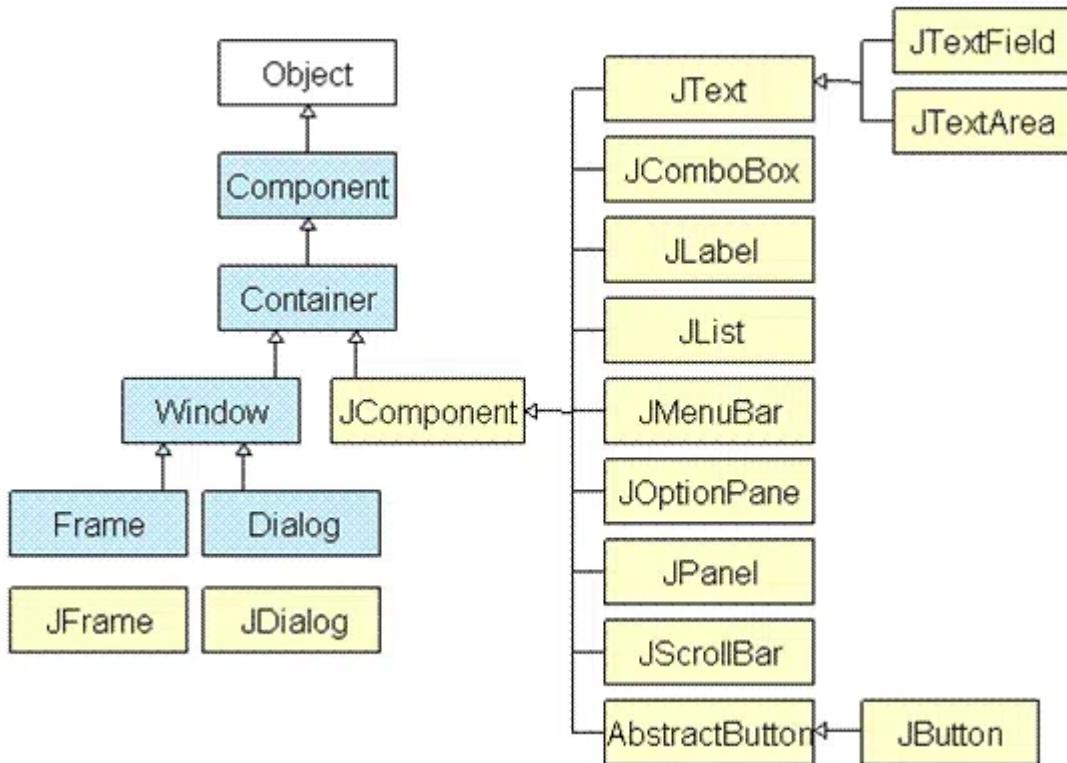
### Kezdeti megjegyezés:

Komment a felhasznált GUI frameworkról:

Úgy tervezük, hogy fogunk használni Swing GUI. Miért?

Mert mindenki próbálta meg ezt előtt (Prog. 3 HFban). Most lesz idő, amelyik tudunk próbálni használni ezt csoportunkban. Ez egy egyszerű framework, amely minden tartalmaz, ami az nekünk kell GUI interfészben. Fogunk használni klasszikus Swing, amelyben van osztályok, mint JPanel, JButton és stb.

Ezen a képen teljesen látható melyik osztályok ott vannak (nem minden fogunk használni, de nagyobb erről):



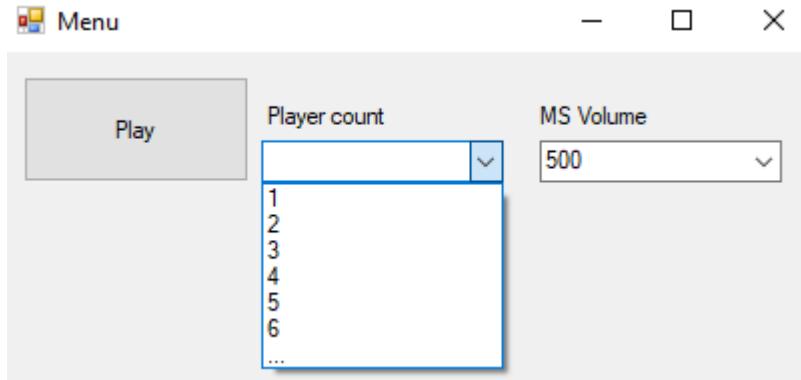
JButton és más események kezeléséhez lehet használni az actionPerformed(ActionEvent ae) metódust, amit a Prog. 3 előadáson tanultunk.

Layout játékban lesz egyszerűbb (GridLayout nélkül) pl. FlowLayout, BoxLayout és stb.

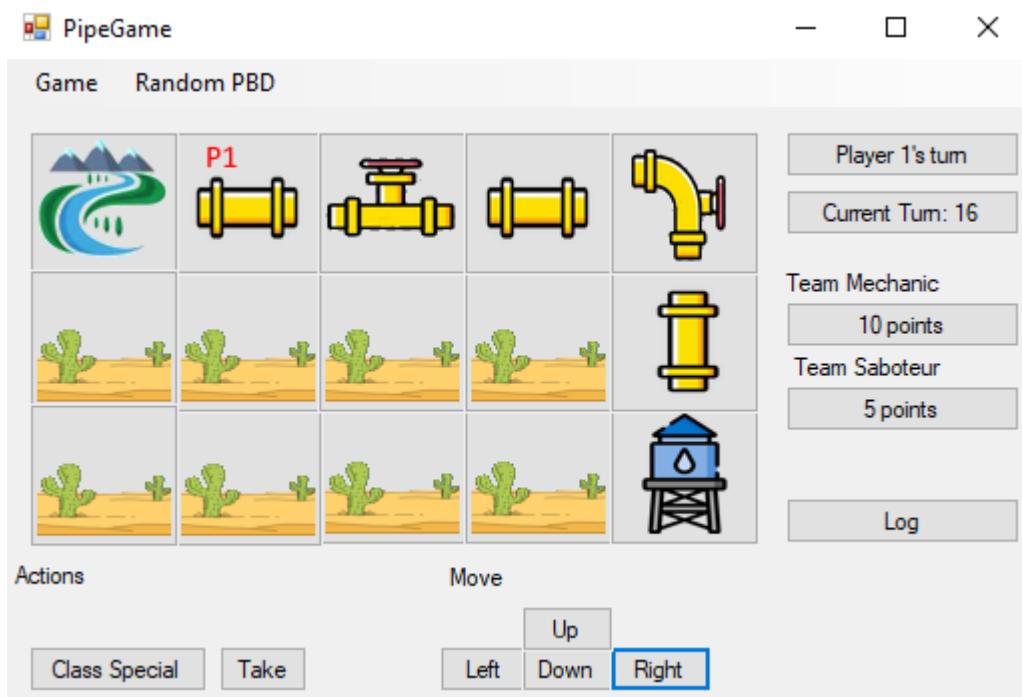
## 11.1 A grafikus interfész

*Megjegyzés: A következő képeket windows forms és photoshop segítségével készítettük el, de ezeket nyilván Java nyelven fogjuk megvalósítani.*

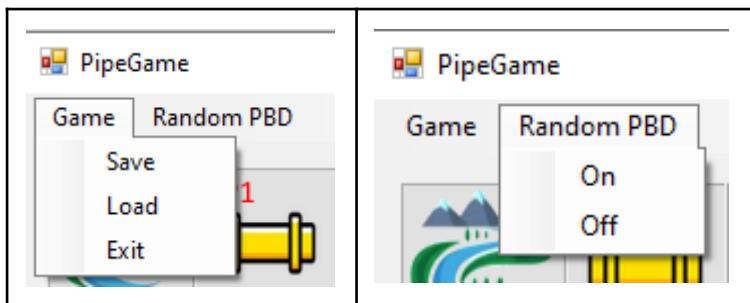
A játék indítása után egyből a menüben találja magát a felhasználó. Itt a play gombra kattintva elindítunk egy új játékot. Mellette található 2 ComboBox, amit majd nagy valószínűséggel egy Settings(beállítások) button/combobox fog megjeleníteni. A két ComboBox a játékosok számának és hegyi forrás víztartalmának beállításáért felel.



A játék pályáját egy gridben fogjuk tárolni, amiben az egyes pályaelemek (pipe, pump, ms, cs) lesznek. Így képzeltük el:



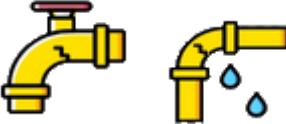
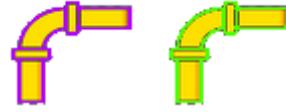
A pálya fölött egy menübar található, ahol majd a játékot el lehet menteni (Save)/egy másikat betölteni(Load). Az Exit-el lehet majd kilépni. Itt található még a véletlen funkciók kikapcsolása is:



A játékpálya bal oldalán található még néhány gomb, amik kiírják, hogy melyik játékosnak van a jelenlegi köre, és hogy hányadik kör van. Ezalatt lesz majd látható gombokkal, hogy melyik csapatnak hánny pontja van. Ezen gombok alatt található még egy log gomb, ami majd a játékosok által megnyomott akciógombok kimenetéért felel. Pl. sikeresen jobbra lépett a játékos. A játékpálya alatt pedig a játékos akció gombjai találhatóak. Pl. pumpa rongálása(Class special), lépés egy irányba(Move). A Take-el lehet majd pedig elemeket elvenni a ciszternából.

A következő táblázat a tervezett játék elem ikonokat tartalmazza:

	Ciszterna
	Sivatag (ahol nincs semmilyen konténer)
	Hegyi forrás
	Több irányú és sarokban lévő cső
	Több irányú és sarokban lévő pumpa
	Olyan cső, amin valamelyen számú játékos áll. (Ezt valószínűleg csak egy text-el fogjuk jelölni)

	Sérült elemeken egy repedés és vízcsepp ikon fog megjelenni
	Sticky Pipe: Zöld szélű cső Slippery Pipe: Lila szélű cső

## 11.2 A grafikus rendszer architektúrája

### 11.2.1 A felület működési elve

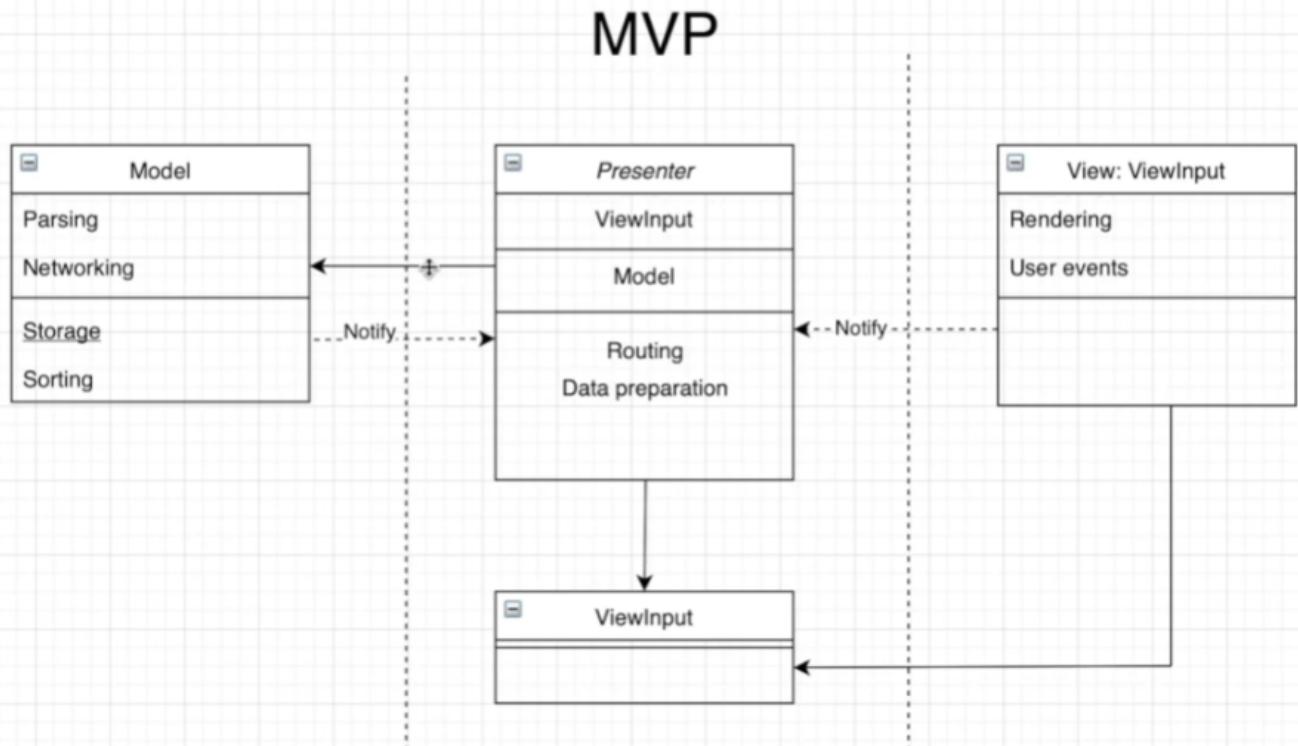
**MVC jó, de MVP jobb. Miért kell MVC vagy MVP használni, mert előtt volt OOP, ahola program gyors és egyszerű bővítése. MVC és MVP paternnel is egyszerűbb bővíteni a programot, mert ott is megosztunk felelősséget.**

**MVP pattern fogunk használni:**

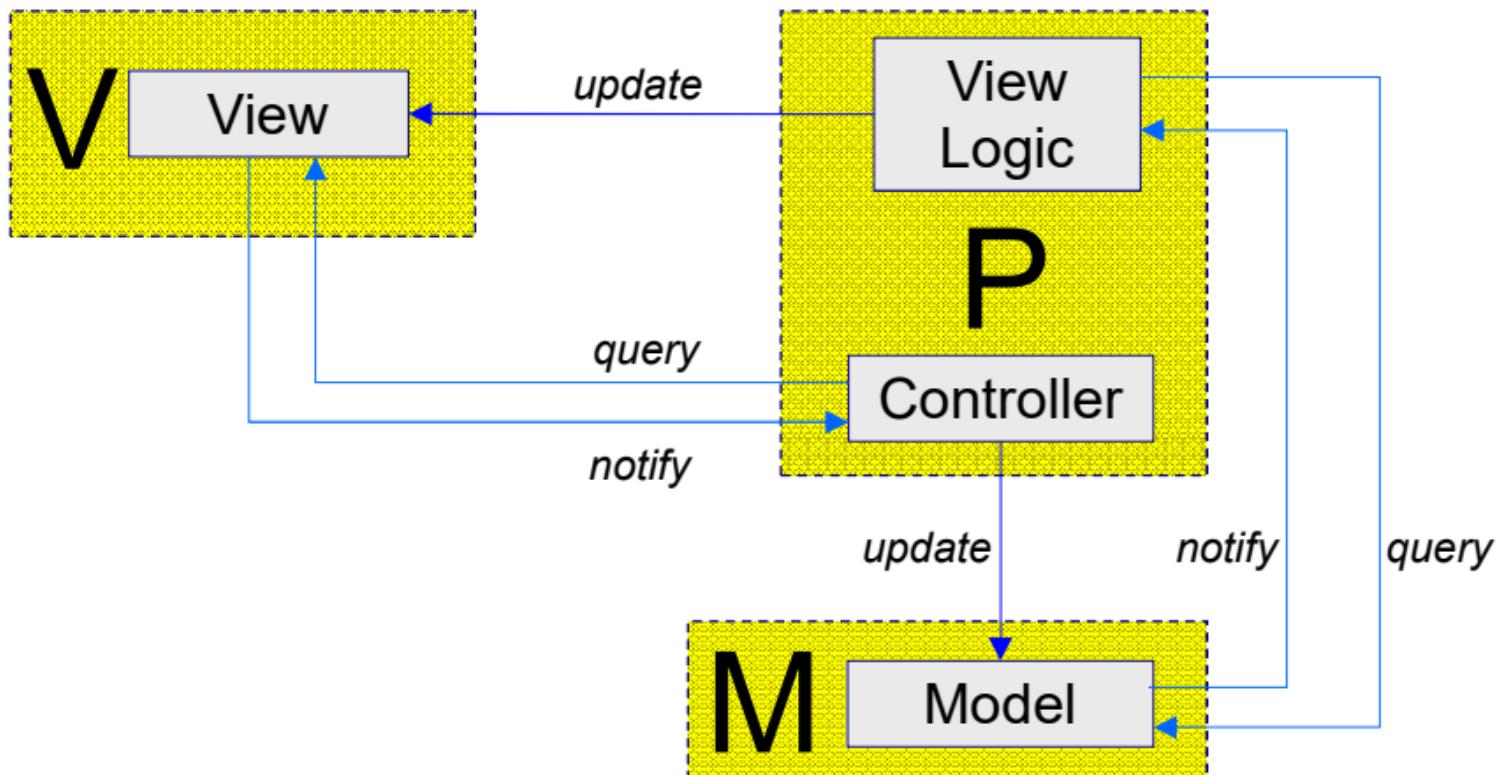
**Miért?**

Ez is jó kérdés. Nekünk van sok ViewInput, ahol fogunk bekapcsolni vmit. Pl. leak Pipe függvényet egyszerűbb fog működni MVPben.

View elküldi ezt az információt az Presenternek. A Presenter feldolgozza ezt a bemenetet, és eldönti, hogy milyen lépéseket tegyen.



# Model-View-Presenter



**Ezen kívül, a Presenter frissíti a modellt: amikor a Presenter feldolgozza a felhasználói bemenetet, kölcsönhatásba lép a modellel, hogy frissítse annak állapotát. A Presenter például meghívhatja a modell metódusait, hogy adatokat adjon hozzá vagy módosítson.**

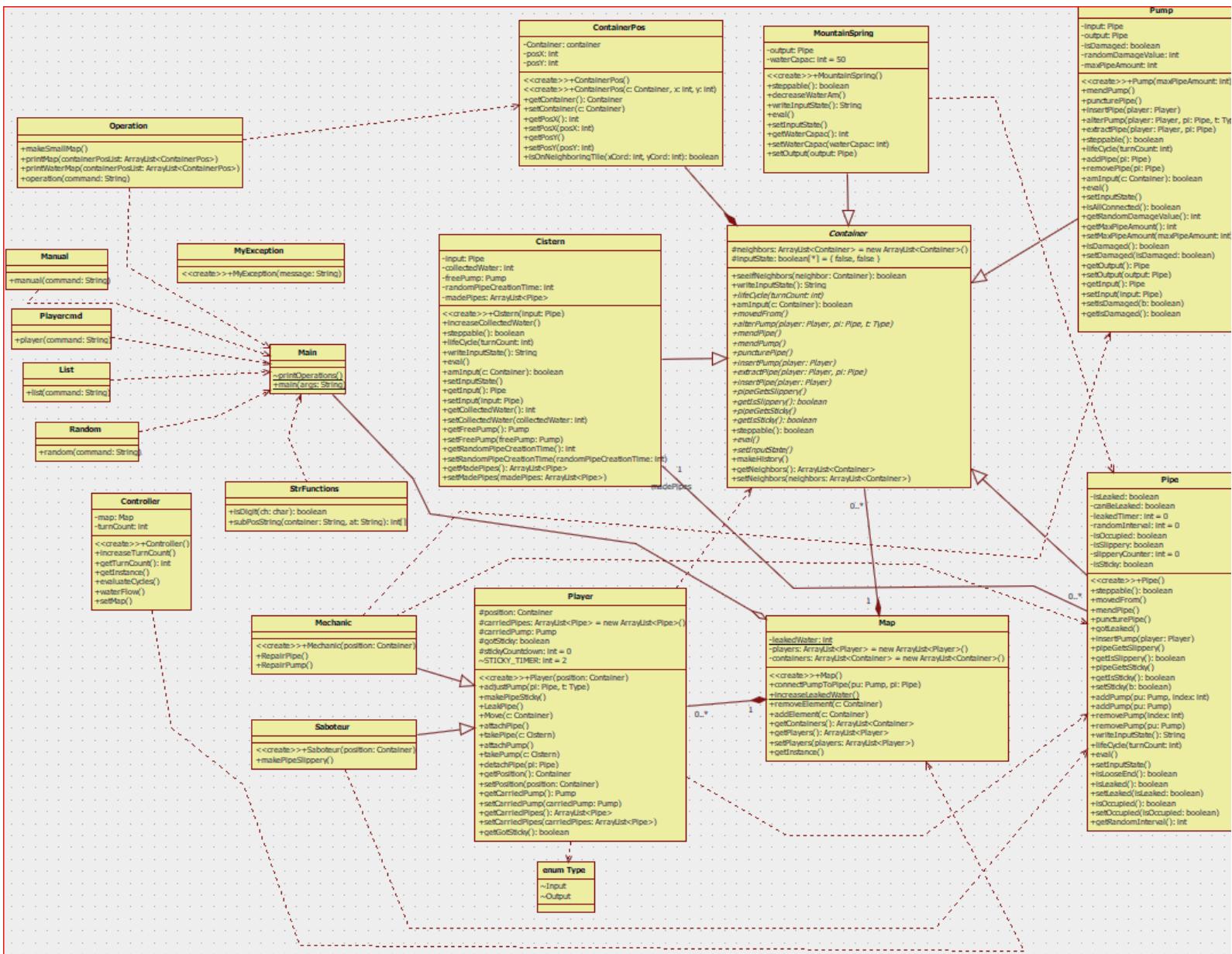
**A Presenter frissíti a View: a modell frissítése után a Presenter értesíti a View, hogy frissíteni kell. A View kapcsolatba lép a Presenterrel a naprakész adatokért, és megjeleníti azokat a képernyőn.**

**MVP kicsit nehezebb, mint MVC, de próbálunk úgy tervezni azt.**

## 11.2.2A felület osztály-struktúrája

Itt lesz az osztálydiagram, ahol lesz új metódusokat, osztályokat (amelyik Prototípusban létrehozzunk) és még GUI is:

Ez a diagramot új osztályokkal és metódusokkal:



**Ez a GUIvel osztálydiagram:**

Csapatunk szeretne egy apró megjegyzéssel elni, hogy ez a struktúra egyértelműen csak egy "tervrajz". Méghozzá egy nagyon illuzórikus. Mivel a GUI felületet nagyon nehéz megtervezni annak minden "szeszélyét" szem előtt tartva. Megjegyezzük, hogy

a legfontosabb az, hogy a Modelben, valamint a Displayben az üzleti logikáért felelős osztályokat és metódusokat rendeljük hozzá. A Presenter osztály megtalálása változhat a struktúrában. Végül is ismerünk olyan példákat, mint például az üzleti logika megtalálása egy Main Main osztályban. Ez tökéletesen működik, és intuitívan minden eseménykezelő egyértelmű.

Fontos megjegyezni, hogy az MVP minta a felelősség megosztására és az alkalmazás tesztelhetőségének támogatására törekszik. Az alapötlet az üzleti logika szétválasztása a modellben, a felhasználói bemenet kezelése a prezentálóban és az adatok megjelenítése a nézetben.

### **Modell:**

**PipeSystem:** Egy csőrendszeret reprezentáló osztály. Tartalmazza a csövek, szivattyúk, tartályok vezérlési logikáját, és szabályozza a vízvezeték- és szabotázsszerelők mozgását a csöveken keresztül.

### **View:**

**PipeSystemView:** A csőrendszer és elemeinek megjelenítéséért felelős. Tartalmazza az olyan grafikus elemeket, mint a csövek, szivattyúk, tartályok és karakterek.

GamePanel: A játékeret képviseli, amely a csőrendszer és az aktív elemeket jeleníti meg.

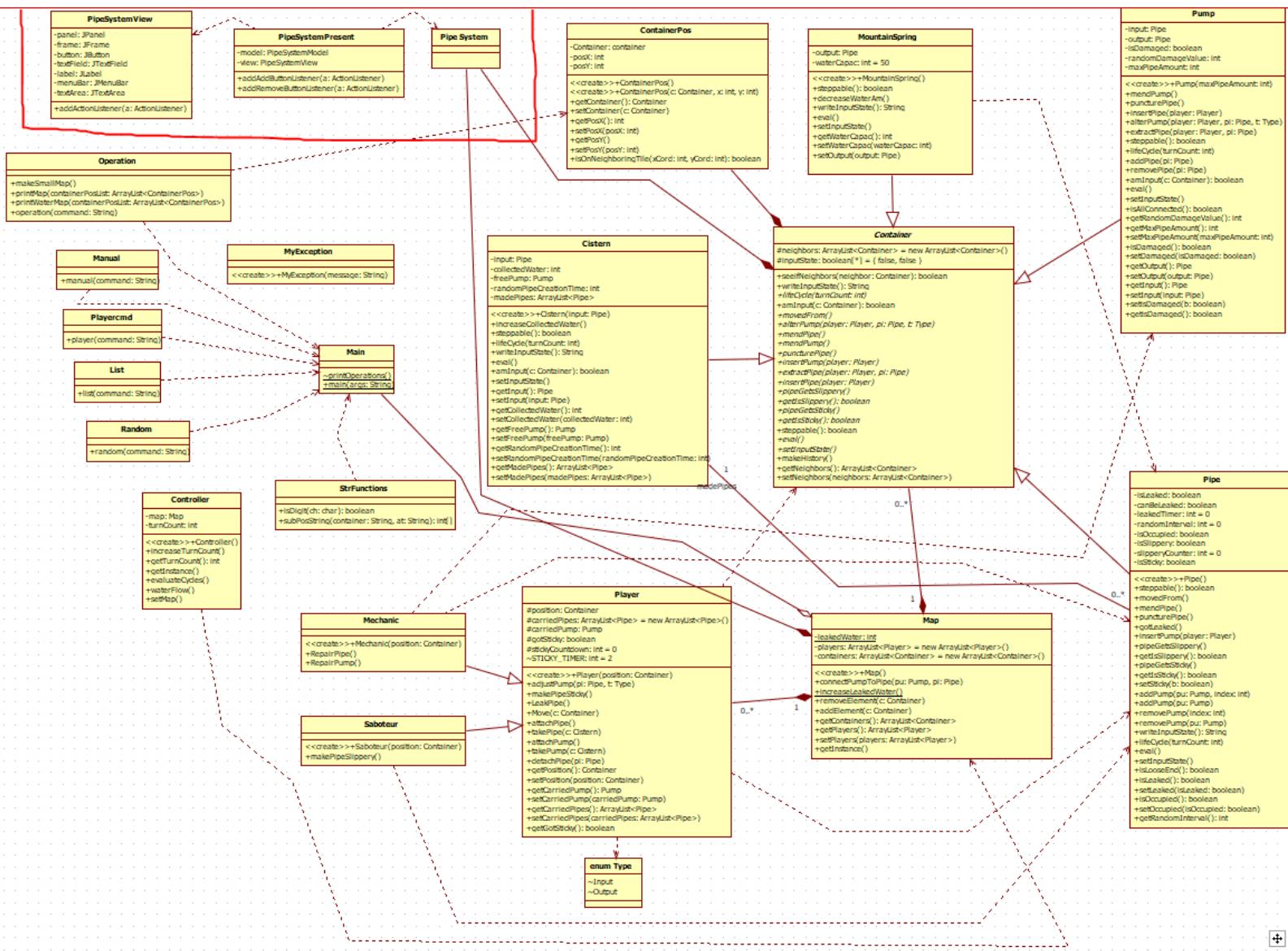
### **Presenter:**

**PipeSystemPresenter:** Összekapcsolja a modellt és a prezentert. Kezeli a felhasználói bemenetet, átadja a megfelelő parancsokat a modellnek, és frissíti a nézetet a modell állapotá alapján. A játékciklus kezeléséért is felelős, beleértve a játék állapotának ellenőrzését és a győztes meghatározását.

### **Interfészek:**

**PipeSystemContract.View:** Meghatározza azokat a metódusokat, amelyeket a nézetnek implementálnia kell a prezenterrel való interakcióhoz.

**PipeSystemContract.Presenter:** Meghatározza azokat a metódusokat, amelyeket a prezenternek implementálnia kell a modellel és a nézzel való interakcióhoz.



## 11.3 A grafikus objektumok felsorolása

### 11.3.1 PipeSystem

- Felelősség

A "PipeSystemModel" modellosztály az MVP-mintán belül a következő célokat és feladatokat látja el:

A csőrendszer adatainak és állapotának tárolása: A modell felelős a csövek, szivattyúk, tartályok és egyéb rendszerelemek állapotára és tulajdonságaira vonatkozó információk tárolásáért. Módszereket biztosít az olyan adatok lekérdezéséhez és módosításához, mint például a tartályok aktuális vízmennyisége, a szivattyúk állapota, a csövek szabad kapacitása stb. A modell a felhasználói műveletek, a rendszerelemekkel való interakciók és egyéb tényezők alapján frissül.

A csővezetékrendszer vezérlési logikája: A modell tartalmazza a csövek, szivattyúk, tartályok, valamint a vízvezeték-szerelők és szabotörök csöveken keresztüli mozgásának vezérlésével kapcsolatos logikát. Meghatározza a rendszerelemekkel kapcsolatos mozgásra, interakcióra és eseménykezelésre vonatkozó szabályokat és korlátozásokat.

Változásértesítés: A modell értesíti a nézetet (PipeSystemView) az adatok vagy a rendszerállapot változásáról. Ez lehetővé teszi a nézet frissítését és a felhasználók számára a naprakész információk megjelenítését. A modell olyan mechanizmusokat valósít meg, amelyekkel a nézetet értesítheti a változásokról, például a megfigyelő minta vagy események segítségével.

Kölcsönhatás más komponensekkel: A modell kölcsönhatásba léphet más osztályokkal és komponensekkel, például a Presenter (PipeSystemPresenter) osztállyal, hogy parancsokat közvetítsen és a felhasználói műveletek alapján frissítse a rendszer állapotát. A modell interakcióba léphet az adatok mentésére és betöltésére, az üzleti logika végrehajtására és a csővezetékrendszer kezelésével kapcsolatos egyéb feladatok elvégzésére szolgáló osztályokkal is.

- **Ősosztályok**

*Composition PipeSystemtől Containerre. Ezzel fogunk kapni minden Metódusok, amelyik van Pipeben, Cisternában és Pumpában.*

- **Interfészek**

-

*Nem láttunk, hogy most kell, de tervezünk, de ha szükség lehet a következők végrehajtására:*

*Runnable*" interfész: Ha a PipeSystem osztálynak képesnek kell lennie egy játékciklus indítására és leállítására, vagy események feldolgozására egy külön szálban, akkor implementálhatja a "Runnable" interfészt. Ez az interfész definiál egy "run()" metódust, amely tartalmazza a feladat külön szálban történő végrehajtásának alapvető logikáját.

*Interfész a csövek kezeléséhez: Ha a "PipeSystem" osztálynak metódusokat kell biztosítania a csövek kezeléséhez, például a csövek hozzádásához, eltávolításához, tulajdonságaik megváltoztatásához stb., akkor implementálhat egy erre a célra definiált speciális interfészt. Például a "PipeSystemInterface" vagy a "PipeManagementInterface".*

*Eseménykezelő interfész: Ha a PipeSystem osztálynak reagálnia kell különböző eseményekre, például gombnyomásokra, karaktermozgásokra és egyéb felhasználói műveletekre, akkor implementálhat egy eseménykezeléssel kapcsolatos interfészt. Például egy "EventListener" vagy egy speciális interfész a játékkörnyezetben történő események kezelésére.*

- **Attribútumok**

- - List<Pipe> pipes;
- - List<Pump> pumps;
- - List<Tank> tanks;
- - List<Plumber> plumbers;

- - List<Saboteur> saboteurs;
- // További attribútumok
- - boolean isRunning;
- - boolean isGameOver;
- - int currentPlayerIndex;

- **Metódusok**

```
public void startSystem() {
    //indítja a csővezetékrendszer
}
```

```
public void stopSystem() {
    // Leállítja a csővezetékrendszer
}
```

Összhangban van a Container osztály compositionjával

### 11.3.2 PipeSystemView

- **Felelősség**

*Az MVP (Model-View-Presenter) mintán belül a PipeSystemView osztály felelős a csőrendszer és elemeinek megjelenítéséért. Fő célja a modell adatainak megjelenítése és a nézet frissítése a modellben bekövetkező változások alapján. A PipeSystemView osztály néhány feladata lehet:*

*Grafikus elemek megjelenítése: A PipeSystemView osztály felelős a grafikus elemek, például a csövek, szivattyúk, tartályok és karakterek képei létrehozásáért és megjelenítéséért.*

*Létrehozza és konfigurálja a megfelelő UI-komponenseket, például gombokat, paneleket stb.*

*A nézet frissítése: Amikor a modellben változások történnek, például egy cső vagy szivattyú állapotának változása, a PipeSystemView frissíti a megfelelő felhasználói felület elemeit, hogy tükrözze a változást. Például a modell állapota alapján megváltoztathatja a grafikus elemek színét vagy állapotát.*

*Felhasználói bemenet feldolgozása: A PipeSystemView figyeli a felhasználói bemenetet, például a gombkattintásokat vagy egéreseményeket. Kezeli ezeket az eseményeket, és a megfelelő parancsokat vagy adatokat visszaküldi a bemutatónak feldolgozásra. Például a start gomb megnyomásakor értesítheti a prezenteret, hogy indítson el egy játékhurkot.*

*Játékciklus-kijelző vezérlése: A PipeSystemView rendelkezhet a játékciklus-kijelzőhöz kapcsolódó logikával, például animációk indításával és leállításával, a képernyő frissítésével stb. Felelős lehet a játék állapotának ellenőrzéséért és a győztes meghatározásáért.*

- **Ősosztályok**

A PipeSystemView osztály öröklési hierarchiája a konkrét megvalósítástól és a felhasználói felület létrehozásához használt keretrendszertől vagy könyvtártól függ. Általában a PipeSystemView osztály a következő osztályokból örökölhető:

A GUI létrehozásához használt keretrendszer vagy könyvtár által biztosított osztályok: Például a Java Swing használata esetén a PipeSystemView osztály örökölhet olyan osztályokból, mint a JFrame vagy a JPanel. Ezek az osztályok biztosítják az alapvető funkciókat a felhasználói felület ablakainak és paneljeinek létrehozásához.

- **Interfészek**

-

- **Attribútumok**

-panel: JPanel

-frame: JFrame

-button: Button

-textField: JTextField

-label: Label

-menuBar: JMenuBar

-textArea: Textarea

- **Metódusok**

Ebben az osztályban fogunk megvalósítani addActionListener buttonnak és stb.

### **11.3.3 PipeSystemPresent**

- **Felelősség**

A PipeSystemPresent osztály az MVP (Model-View-Presenter) architektúrában a következő célokat és feladatokat látja el:

A modell és a nézet összekapcsolása: A Presenter fontos összekötő kapocsként szolgál a modell és a nézet között, biztosítva a köztük lévő interakciót. Frissítéseket kap a modelltől, és azokat megjelenítésre átadja a nézetnek, valamint felhasználói bemenetet kap a nézettől, és azt feldolgozásra átadja a modellnek.

A felhasználói bemenet kezelése: A bemutató felelős a nézetből érkező felhasználói bemenet kezeléséért. Reagál az olyan eseményekre, mint a gombnyomások, a szövegbevitel és egyéb felhasználói műveletek. Amikor ilyen bemenetet kap, a Presenter megfelelő műveleteket hajt végre, elvégzi a szükséges műveleteket, és frissíti a modell és a nézet állapotát.

Üzleti logika: A Presenter a tárgykörhöz tartozó alkalmazási logikát tartalmazza. Adatfeldolgozási műveleteket, számításokat, külső szolgáltatásokkal való interakciót és

egyéb, az alkalmazás üzleti logikájához kapcsolódó műveleteket végezhet. A Presenter kezeli e műveletek hívásait, és koordinálja végrehajtásukat a modellel és a nézettel.

Nézet frissítése: A Presenter felelős a nézet állapotának frissítéséért a modellben bekövetkezett változások alapján. Értesítéseket kap a modelltől az adatváltozásokról, és frissíti a nézetet, hogy a felhasználó számára az adatok aktuális állapotát jelenítse meg. Ez magában foglalhatja a szöveg, az elem láthatóságának, színének és a nézet egyéb attribútumainak módosítását.

Munkafolyamat-kezelés: A Presenter az alkalmazás munkafolyamatát is kezelheti. Meghatározhatja a műveletek sorrendjét, a modell és a nézet metódusait a megfelelő sorrendben hívhatja meg, és koordinálhatja a feladatok végrehajtását. A Presenter képes kezelni és vezérelni az alkalmazás életciklus eseményeit is, például az indítást, leállítást, újraindítást és egyebeket.

- **Ősosztályok**

Az MVP-minták kontextusában a PipeSystemPresent osztály általában nem örökli közvetlenül a többi osztályt, mivel elsőleges feladata a modell (PipeSystemModel) és a nézet (PipeSystemView) összekapcsolása, nem pedig az öröklés.

A PipeSystemPresent azonban használhatja a PipeSystemView és a PipeSystem definiált interfészeket, és függhet azoktól, hogy kapcsolatot biztosítson közöttük.

- **Interfészek**

-

- **Attribútumok**

-model: Pipe System Model

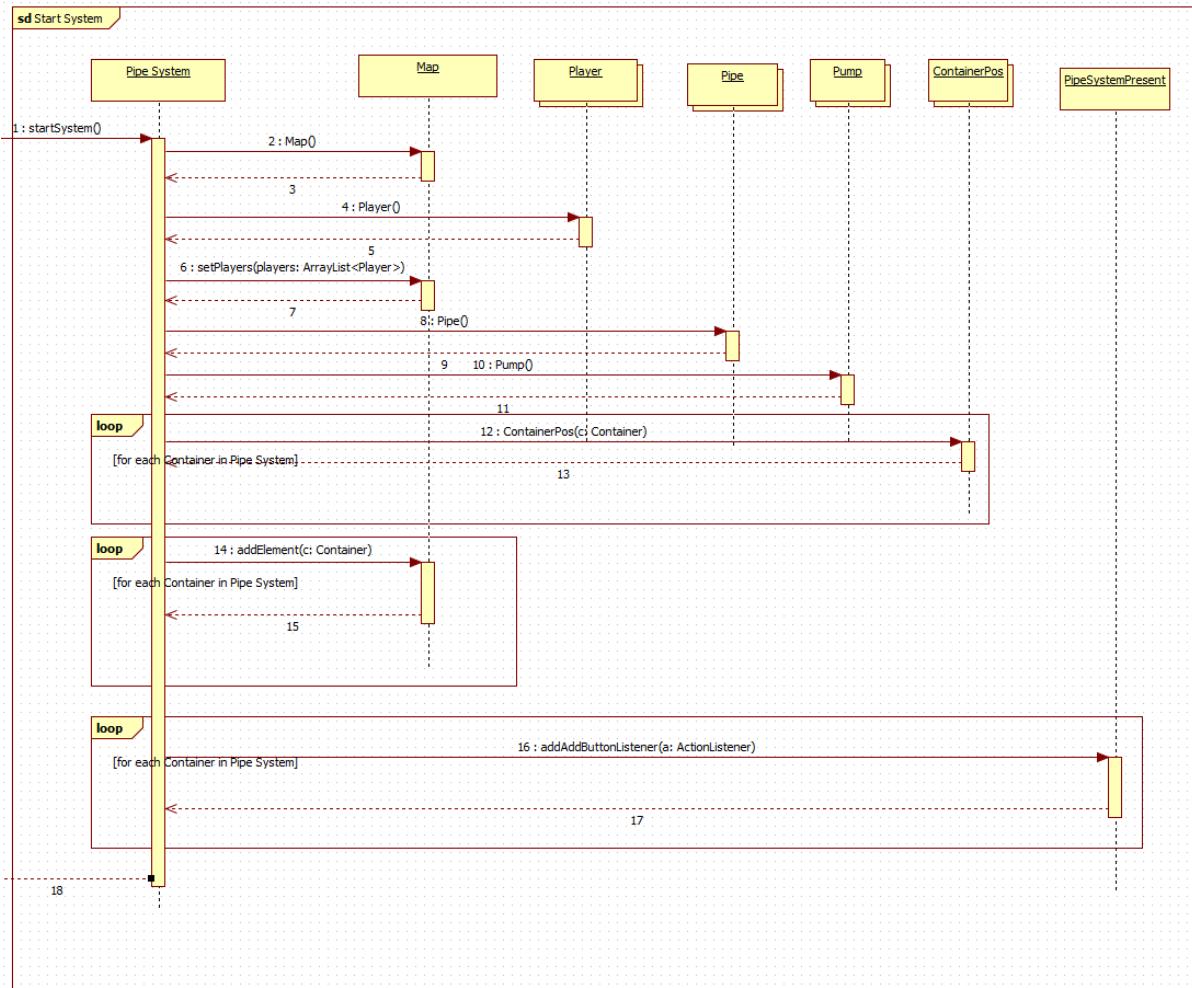
-view: Pipe System View

- **Metódusok**

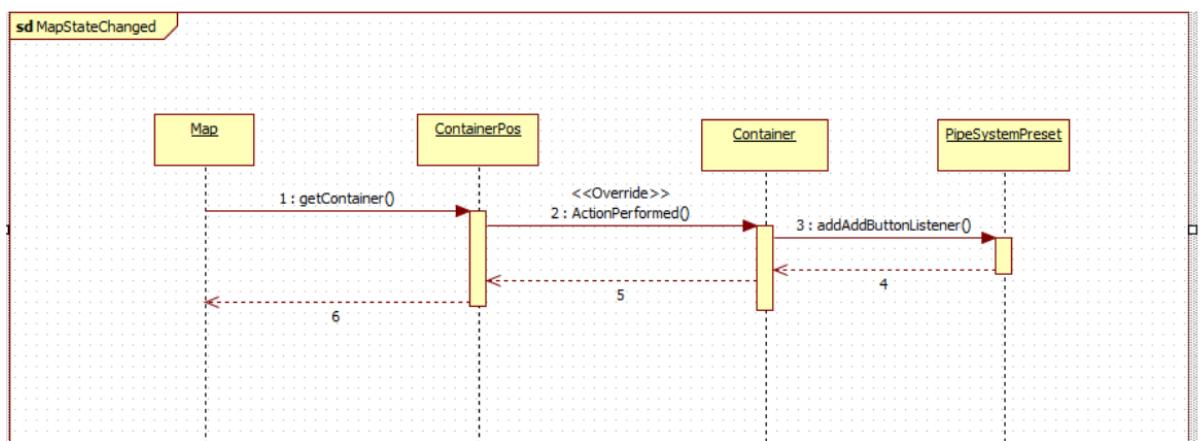
Ebben az osztályban fogjuk megvalósítani az addButtonListener-t, amelyben meghívjuk az addAddActionListener-t és kommunikálunk a PipeSystemView osztályjal, valamint az addRemoveButonListener-t fordított logikával.

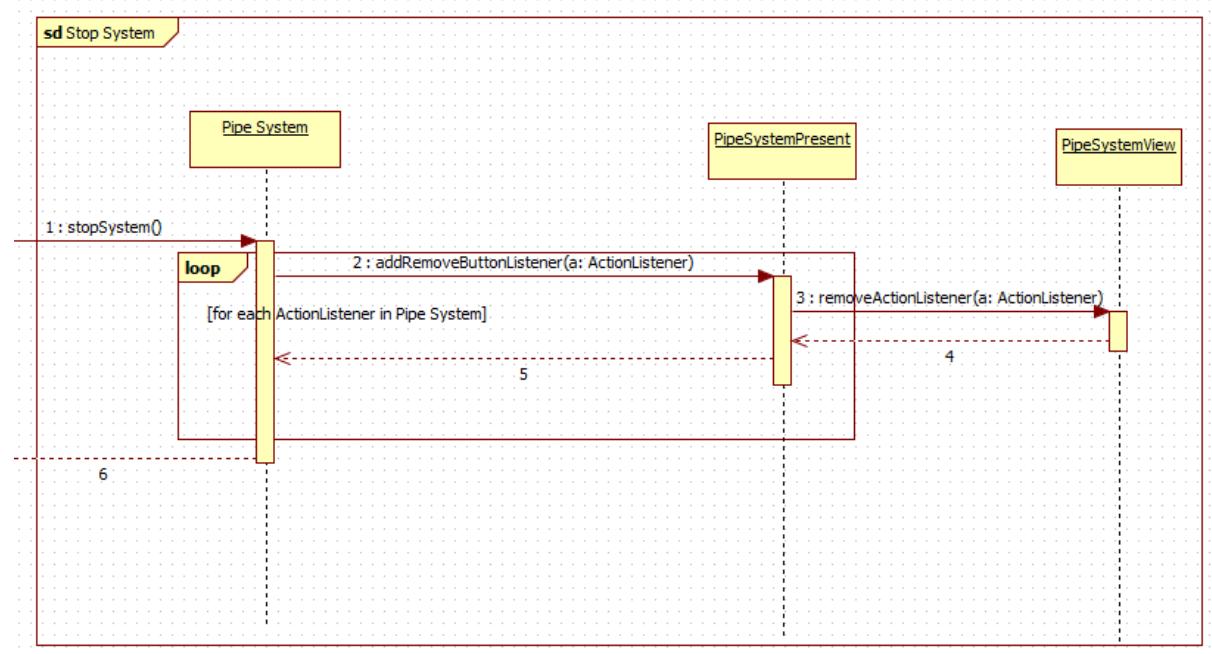
## **11.4 Kapcsolat az alkalmazói rendszerrel**

## Creating the map:



## Map State Changed



**Destroying the map:**

**Napló**

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résztvevők</b>	<b>Leírás</b>
2023.05.20 12:00	0.5 óra	Tisza Miklós Réti Ádám Kopach Artem Czifra Barnabás Molnár Márton	Értekezlet.
2023.05.20 14:00	3 óra	Réti Ádám	Tevékenység: 11.1 A grafikus interfész megtervezése
2023.05.20 - 05.21 12:00	7 óra	Kopach Artem	Tevékenység: Kezdi megjegyezés írtam, 11.2 egész, 11.3 egész.
2023.05.21 13:00	2 óra	Czifra Barnabás Molnár Márton	11.4 elkészítése