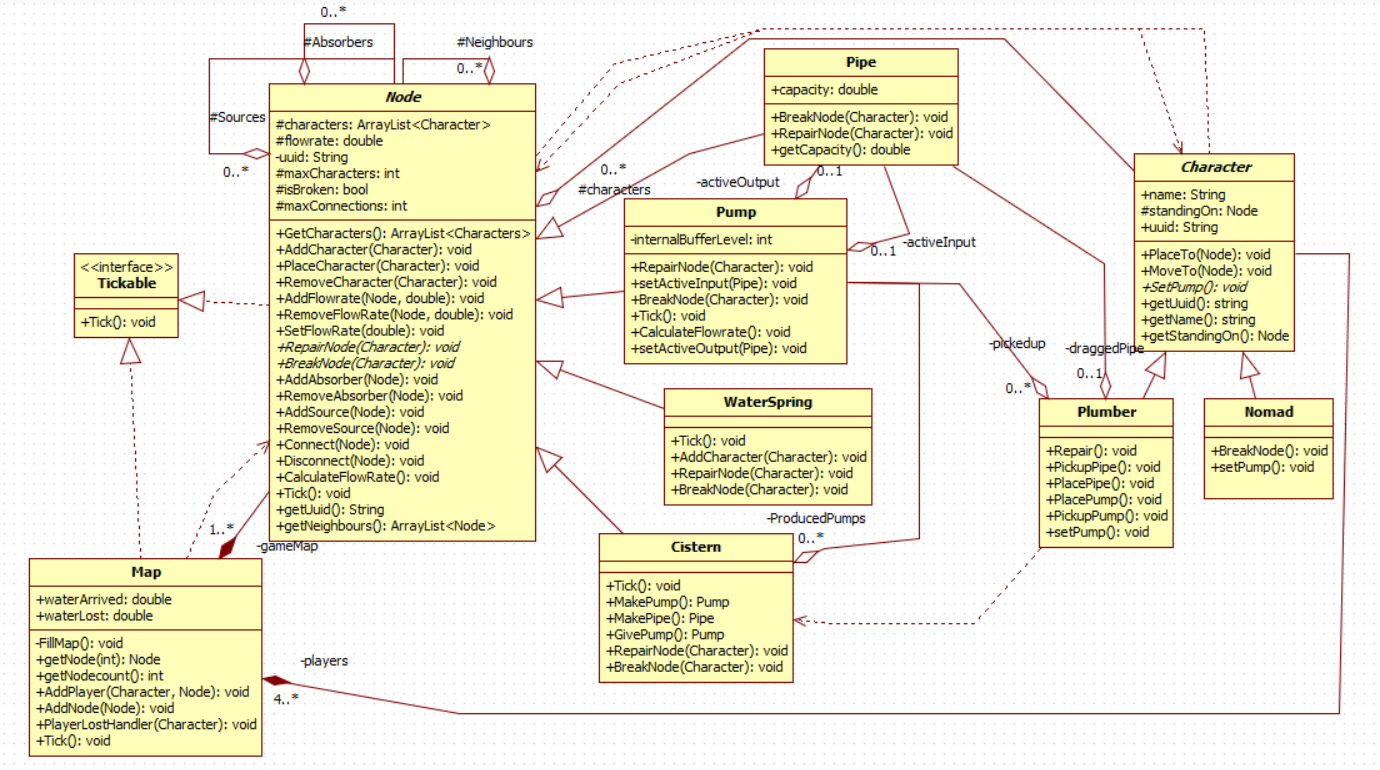
# Prototípus koncepciója

[A prototípus program célja annak demonstrálása, hogy a program elkészült, helyesen működik, valamennyi feladatát teljesíti. A prototípus változat egy elkészült program kivéve a kifejlett grafikus interfészt. Ez a program is parancssorból futtatható és karakteres ernyőkezelést alkalmaz. Az ütemezés, az aktív objektumok kezelése megoldott. A business objektumok - a megjelenítésre vonatkozó részeket kivéve - valamennyi metódusa a végleges algoritmusokat kell, hogy tartalmazza. A megjelenítés és működtetés egy alfanumerikus képernyőn vezérelhető és követhető, ugyanakkor a vezérlés fájlból is történhet és a megjelenítés fájlba is logolható, ezzel megteremtve a rendszer tesztelésének lehetőségét. Különös figyelmet kell fordítani a parancssori interfész logikájára, felépítésére, valamint arra, hogy az mennyiben tükrözi és teszi láthatóvá a program működését, a beavatkozások hatásait.]

## Változás hatása a modellre

### Módosult osztálydiagram



### Új vagy megváltozó metódusok

#### Map

+getNode(int): Visszaadja az adott indexen lévő Node-ot a Map-ból.

+getNodeCount(): Visszaadja a Map-ban lévő Node-ok számát.

#### Node

+getUuid(): Visszaadja az adott Node egyedi azonosítóját.

+getNeighbours(): Visszaad egy az adott Node-hoz tartozó listát annak szomszédairól.

#### Cistern

+RepairNode(Character): Nem hagyja, hogy a ciszternát megjavítsák.

+BreakNode(Character): Nem hagyja, hogy a ciszterna elromoljon.

+givePump(): Odaad egy már elkészített pumpát a rajta álló szerelőnek, ha van ilyen.

#### Pump

+setActiveInput(Pipe): Beállítja, hogy honnan folyik a víz a pumpába.

+setActiveOutput(Pipe): Beállítja, hogy hová folyik a víz a pumpából.

#### Pipe

+getCapacity(): Visszaadja, hogy mekkora az adott cső kapacitása.

#### Plumber

+setPump(): Beállítja a pumpát, amin a szerelő áll.

\*GrabPipe() 🡪 PickupPipe(): Fölvesz egy pumpát a ciszternáról.

\*AddPump() 🡪 PlacePump(): Leteszi a pumpát egy csőre.

\*GrabPump() 🡪 PickupPump(): Fölvesz egy pumpát a ciszternáról.

#### Character

+getUuid(): Visszaadja az adott karakter egyedi azonosítóját.

+getStandingOn(): Visszaadja hogy hol áll az adott karakter.

+getName(): Visszaadja az adott karakter nevét.

#### Nomad

+setPump(): Beállítja a pumpát, amin a nomád áll.

### Szekvencia-diagramok

*[Az analízis modell szekvenciadiagramjaiból a változás által érintett, előírt, módosított diagramok.]*

## Prototípus interface-definíciója

[Definiálni kell a teszteket leíró nyelvet. Külön figyelmet kell fordítani arra, hogy ha a rendszer véletlen elemeket is tartalmaz, akkor a véletlenszerűség ki-bekapcsolható legyen, és a program determinisztikusan is tesztelhető legyen.]

### Az interfész általános leírása

[A protó (karakteres) input és output felületeit úgy kell kialakítani, hogy az input fájlból is vehető legyen illetőleg az output fájlba menthető legyen, vagyis kommunikációra csak a szabványos be- és kimenet használható.]

### Bemeneti nyelv

[Definiálni kell a teszteket leíró nyelvet (szintakszis és szemantika). Külön figyelmet kell fordítani arra, hogy ha a rendszer véletlen elemeket is tartalmaz, akkor a véletlenszerűség ki-bekapcsolható legyen, és a program determinisztikusan is futtatható legyen. A szálkezelést is tesztelhető, irányítható módon kell megoldani. A programot egy adott konfigurációból is el kell tudni indítani, vagyis kell olyan parancs, amivel konkrét előre megadott állapotból indul a rendszer (pl. load).]

**Parancs1**

**Leírás:**

**Opciók:**

[Ha szükséges, meg kell adni a konfigurációs (pl. pályaképet megadó) fájlok nyelvtanát is.]

### Kimeneti nyelv

[Egyértelműen definiálni kell, hogy az egyes bemeneti parancsok végrehajtása után előálló állapot milyen formában jelenik meg a szabványos kimeneten. A program képes legyen olyan kimenetet előállítani, amellyel az objektumok állapota ellenőrizhető (pl. save). Ebben az alfejezetben is precízen definiálni kell, hogy a kimenet nyelve milyen elemekből és milyen szintakszissal áll elő.]

## Összes részletes use-case

[A use-case-eknek a részletezettsége feleljen meg a kezelői felületnek, azaz a felület elemeire kell hivatkozniuk a bemeneti nyelv parancsai alapján.

Alábbi táblázat minden use-case-hez külön-külön.]

|  |  |
| --- | --- |
| **Use-case neve** |  |
| **Rövid leírás** |  |
| **Aktorok** |  |
| **Forgatókönyv** |  |

## Tesztelési terv

[A tesztelési tervben definiálni kell, hogy a be- és kimeneti fájlok egybevetésével miként végezhető el a program tesztelése. Meg kell adni magas szintű teszt forgatókönyveket. Az egyes teszteket elég informálisan, szabad szövegként leírni, tesztesetenként egy-öt mondatban. Minden teszthez meg kell adni, hogy mi a célja, a proto mely funkcionalitását, osztályait stb. teszteli. Az alábbi táblázat minden teszt-esethez külön-külön elkészítendő.]

|  |  |
| --- | --- |
| **Teszt-eset neve** |  |
| **Rövid leírás** |  |
| **Teszt célja** |  |

## Tesztelést támogató segéd- és fordítóprogramok specifikálása

[Rövid bemutatással (elvárt funkcionalitás) specifikálni kell a tesztelést támogató segédprogramokat.]

## Napló

|  |  |  |  |
| --- | --- | --- | --- |
| **Kezdet** | **Időtartam** | **Résztvevők** | **Leírás** |
| 2023. 04. 19. 13:00 | 3,5 óra | Zsigmond | A feladat előkészítése, változások leírása. |
| 2023. 04. 19. 14:30 | 0,25 óra | Nyilas, Zsigmond | Értekezlet a változásokról. |
| 2023. 04. 19. 14:45 | 2 óra | Nyilas | A változások beírása a diagramokba. |