# 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

A képen diagram látható

Automatikusan generált leírás

### 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.

+BreakNode(): Plumber karakter megrongálja azt a Node-ot amin 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.

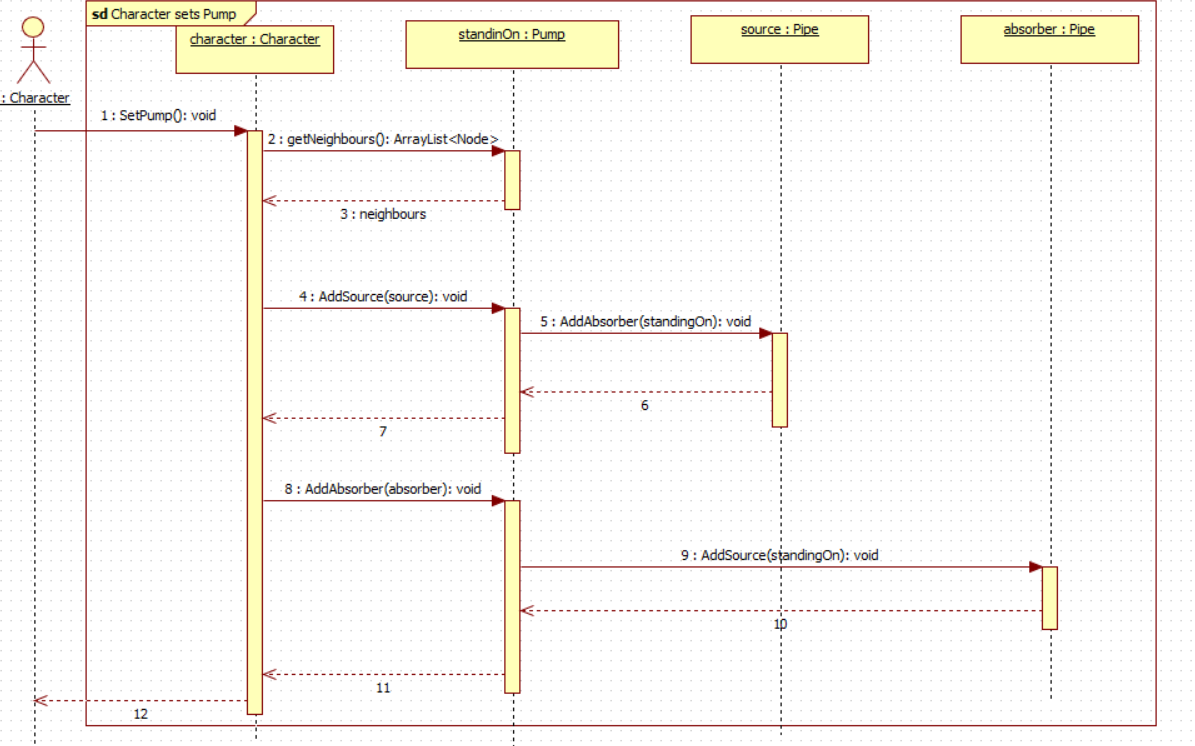
*+BreakNode():* Absztrakt metódus, karakter leszármzottai implementálják hogyan rongálnak meg egy Node-ot.

#### Nomad

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

### Szekvencia-diagramok

#### Character sets pump



#### Cistern makes Pump

A képen diagram látható

Automatikusan generált leírás

#### Cistern makes Pipe

A képen diagram látható

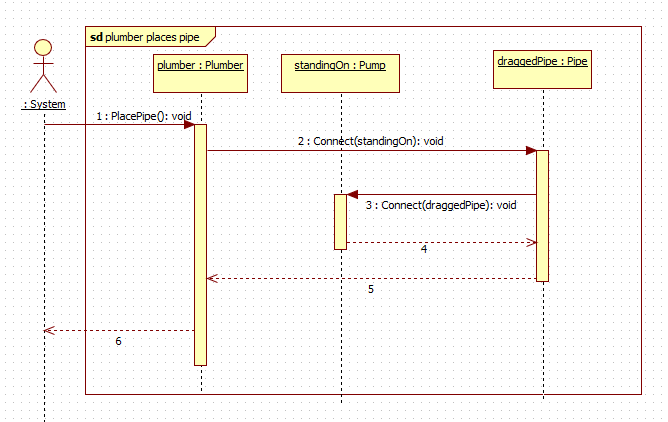
Automatikusan generált leírás

#### Plumber places Pump

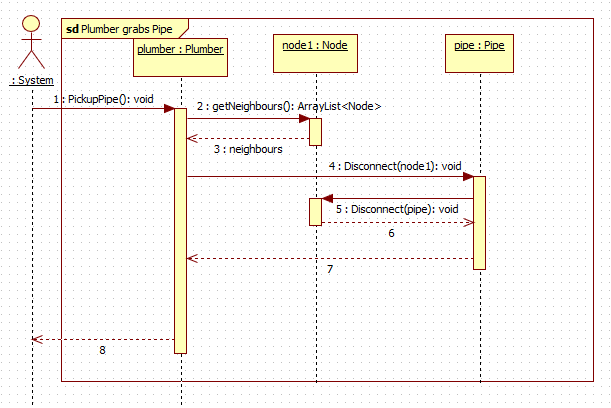
A képen diagram látható

Automatikusan generált leírás

#### Plumber places Pipe



#### Plumber grabs Pipe



#### Map initalizes

A képen diagram látható

Automatikusan generált leírás

#### Plumber grabs Pump

A képen diagram látható

Automatikusan generált leírás

### Új funkciók szekvenciái

#### Plumber breaks Pipe

A képen diagram látható

Automatikusan generált leírás

## 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).]

Parancsok:

* fill  
  Leírás: Feltölti a map-ot random elemekkel, ha nem akarjuk manuálisan felvenni az elemeket és játékosokat  
  Opciók: player count: hány darab játékost akarunk lerakatni
* load  
  Leírás: Beolvas egy megadott konfigurációs file-ból egy játék állapotot  
  Opciók: file path: melyik file-t akarjuk beolvasni
* save  
  Leírás: Kiírja a játék állapotát egy általunk megadott file-ba  
  Opciók: file path: hova mentse el a játékot
* character  
  Leírás: a karakterekhez tartozó akciókat tudjuk innen indítani  
  Opciók:
  + move: mozgás a karakterrel
  + break: a karakter alatti node elrontása
  + repair: a karakter alatti node megjavítása
  + set: a node átállítása
  + pick up: a node felvétele
  + place: a karakter hátizsákjából egy elem lehelyezése
* node  
  Leírás: a node-nak a kezelési opciói  
  Opciók:
  + break: kilyukad a cső vagy pumpa
  + repair: megjavul
* add  
  Leírás: új elemet vagy karaktert tudunk felvenni itt  
  Opciók:
  + character: karakter veszünk fel  
    Opciók:
    - name: a karakter neve
    - type: a karakter típusa (nomad vagy plumber)
    - node: ahova le szeretnénk helyezni
  + node:  
    Opciók:
    - type: a node fajtálya (cistern, waterSpring, pipe, pump)
    - position: egy x és y koordináta, ahova legyen lehelyezve
* map  
  Leírás: a térkép akciói  
  Opciók:
  + tick: az egész map minden elemének 1 körrel való léptetése
  + print: a térkép elemeinek kiírtatása átlátható módon
* exit  
  Leírás: a prototípus elhagyása

Konfigurációs file nyelve:

A konfigurációs file egy json alapú file amiben a következő sorrendben lehet definiálni a játék állapotát:

{

size: a játék térképének mérete,

players: egy lista amiben vannak a játékosok információi

map: egy lista, amiben a node objektumok szerepelnek nem feltétlenül sorrendben

}

A players objektum elemei a következő képpen néznek ki:

{

type: a player típusa

name: a player neve

x: az x koordinátája

y: az y koordinátája

inventory: a hátizsákja ami egy node objektum lista de ez csak Plumber típusú playereknek lehet

}

A map objektum elemei a következő képpen néznek ki:

{

type: a node típusa

x: a node x koordinátája

y: a node y koordinátája

connections: ez a tulajdonság nem feltétlenül kell hogy szerepeljen csak ha a típus Pipe vagy Pump, ez egy lista a szomszédok koordinátájával, amikkel a node össze van kötve

}

Pl:

{  
 "size": 10,  
 "players": [  
 {  
 "type": "plumber",  
 "name": "Mario",  
 "x": 0,  
 "y": 0,  
 "inventory": []  
 },  
 {  
 "type": "nomad",  
 "name": "Wario",  
 "x": 0,  
 "y": 0  
 },  
 ],  
 "map": [  
 {  
 "type": "cistern",  
 "x": 0,  
 "y": 0  
 },  
 {  
 "type": "pipe",  
 "x": 0,  
 "y": 1,  
 "connections": [  
 {  
 "x": 0,  
 "y": 0  
 }  
 ]  
 }  
 ]  
}

### 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

|  |  |
| --- | --- |
| **Teszt-eset neve** | Pálya kiírása |
| **Rövid leírás** | A program a betölti és kirajzolja a konzolra a játékpályát |
| **Teszt célja** | A pálya helyes betöltését, az egyes elemek megfelelő elhelyezkedését ellenőrzi. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Új pumpa elem felvétele |
| **Rövid leírás** | A szerelő magához vesz egy új pumpát a ciszternáknál |
| **Teszt célja** | Megvizsgálja, hogy a szerelő milyen elemen áll. Új pumpát csak ciszternánál vehet fel. Ha nem ciszternán áll, akkor ezt üzenet jelzi a játékosnak. Ha ciszternán áll, akkor felveszi az új pumpát. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Új cső elem felvétele |
| **Rövid leírás** | A szerelő magához vesz egy új csövet a ciszternáknál |
| **Teszt célja** | Megvizsgálja, hogy milyen elemen áll. Új csövet csak ciszternánál vehet fel. Ha nem ciszternán áll, akkor ezt üzenet jelzi a játékosnak. Ha ciszternán áll, akkor felveszi az új csövet. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Karakter mozgatása |
| **Rövid leírás** | A karakterek mozgatása egyik elemről a másikra |
| **Teszt célja** | Az adott karakter áthelyezését vizsgálja, ami történhet csőre, pumpára, valamint a szerelők esetében ciszterna elemre. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Cső eltörése |
| **Rövid leírás** | A nomád karakter eltöri a csövet, amin éppen áll |
| **Teszt célja** | Ellenőrzi, hogy cső típusú elemen áll-e a nomád és hogy ez a cső még nincs eltörve, majd eltöri azt. Ha már törött a cső, akkor ezt üzenet jelzi és a művelet nem hajtható végre. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Pumpa elállítása |
| **Rövid leírás** | A nomád karakter elállítja a pumpát, amin éppen áll |
| **Teszt célja** | Ellenőrzi, hogy pumpa típusú elemen áll-e a nomád és hogy a pumpa még nincs elállítva, majd elállítja azt. Ha ez nem teljesül, akkor azt a program jelzi. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Pumpa javítása |
| **Rövid leírás** | A szerelő megjavít egy pumpát |
| **Teszt célja** | Megnézi, hogy a szerelő pumpán áll-e és hogy az a pumpa törött-e. Ha igen, végrehajtja a műveletet. Egyéb esetben jelzi a problémát. |

|  |  |
| --- | --- |
| **Teszt-eset neve** | Cső javítása |
| **Rövid leírás** | A szerelő megfoltoz egy csövet |
| **Teszt célja** | Megnézi, hogy a szerelő csövön áll-e és hogy az a cső lyukas-e. Ha igen, végrehajtja a műveletet. Egyéb esetben jelzi a problémát. |

## Tesztelést támogató segéd- és fordítóprogramok specifikálása

A teszteseteket parancssorból lehet indítani és a tesztek eredménye is itt lesz látható. A konzolra kiírt eredmény összevethető az elvárt eredménnyel, így megállapítható a művelet helyessége.

## 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. |
| 2023. 04. 22. 14:00 | 3,5 óra | Kis | Változtatások a kódban, bemeneti nyelv és kimeneti nyelv implementálása |
| 2023. 04. 21. 19:00 | 1,5 óra | Sőregi | Tesztek kidolgozása |