# Részletes tervek

[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 szintakszisát felhasználva, hogy mely tesztekkel lesz a prototípus ellenőrizve.]

## Osztályok és metódusok tervei.

### Character

#### Felelősség

A játékban szereplő karaktereket megvalósító osztályok ősosztálya. Tárolja, hogy a pálya mely elemén (Node) tartózkodik, és az összes karakter közös viselkedését megvalósítja.

#### Ősosztályok

Nincs ősosztálya.

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **name**: karakter neve
* **uuid**: karakter egyedi azonosítója
* **standingOn**: eltárolja hogy a karakter a Map mely elemén, melyik Node-on áll.
* **stuckedInPipe**: random generált idő, ameddig a karakter nem léphet el a csőről ha az ragacsos.

#### Metódusok

* **void placeTo(Node startPoint)**: A karaktert lehelyezi a pálya egy elemére, a játék kezdetekor.
* **void moveTo(Node moveTo)**: A karakter mozgását valósítja meg. Áthelyezi a paraméterként megadott Nodera, és leveszi az előző Noderól.
* ***void setPump(Pipe in, Pipe out)****:* A karakter beállítja a Pumpán, amin áll. A Pumpához csatlakoztatott csövek közül melyiket nyitja meg, hogy a pumpába ki és be- folyjon a víz. Absztrakt metódus, leszármazottak implementálják.
* **string getUuid():** Visszaadja a karakter uuidjét.
* **string getName():** Visszaadja a karakter nevét.
* **Node getStaindingOn():** Visszaadja azt a Nodeot, amin a karakter áll.
* **void breakNode():** Megpróbálja megrongálni azt a Nodeot amin éppen áll. Ha az a Node nem pipe akkor kivételt dob.
* **void makePipeSticky():** Megpróbálja ragacsossá tenni azt a Nodeot amin áll. Ha az a Node nem pipe akkor kivételt dob.
* **void tick():** Tickable interfészt implementált metódusa. Körönként csökkenti a stuckedInPipe időt, miután ragacsos csőre lépett.
* **void stuck():** Ha ragacsos csőre lépünk generálunk egy véletlen számot, és beállítjuk ezt a stuckedInPipe időnek.

### Cistern

#### Felelősség

A játékban szereplő ciszternákat megvalósító osztály. Ide gyűjtik a vizet a szerelők, és készülnek itt új csövek és pumpák is.

#### Ősosztályok

Node → Cistern

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **prodicedPumps:** A ciszterna maga által készített pumpákat tárolja.

#### Metódusok

* **void tick()**: időközönként számolja a ciszternába beérkezett vizet, a hozzá kapcsolt csövekből.
* **Pipe makePipe()**: Új csöveket készít, amit lehelyez a Map-re és magához csatolja.
* **Pump makePump()**: Új pumpákat készít, amit elraktároz a producedPump-ba.
* **Pump givePump():** A rajta álló szerelőnek ad egy pumpát ha az akar felvenni.
* **void repairNode(Character character):** Próbálja megjavítja az objektumot. Kivételt dob.
* **void breakNode(Character character):** Próbálja elrontja az objektumot, kivételt dob.

### Map

#### Felelősség

A játékteret megvalósító osztály, pálya. Játéktért elemeit (Node-okat), és a játékos karaktereket csoportosítja.

#### Ősosztályok

Nincs ősosztály

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **waterArrived:** Nyilvántartja, hogy mennyi víz folyt be a ciszternába.
* **waterLost:** Nyilvántartja, hogy mennyi víz folyt el a lyukakon a sivatagi homokra.
* **players:** A pályán levő játékos Character-ek.
* **gameMap:** A pálya elemeit tartalmazó lista.

#### Metódusok

* **void FillMap ()**:Map feltöltése elemekkel és játékososkkal.
* **Node getNode(int x, int y):** Visszaadja a megadott koordinátán lévő Nodeot, ha van ilyen.
* **int getNodeCount():** Megszámolja és visszaadja a pályán lévő Nodeok számát.
* **void addNode (Node node, int x, int y)**: Új pályaelem (Node) hozzáadása a pályához a megadott koordinátákra
* **void removeNode(Node n)**: .Kiveszi a megadott Nodeot a pályából.
* **void addPlayer(Character c, Node n):** A paraméterként megadott játékos hozzáadása a játékhoz, és lehelyezése a megadott Nodera.
* **void Tick():** Időközönként (tick) számolja a ciszternába folyt, és a homokba elfolyt víz mennyiséségt.
* **void PlayerLostHandler(Character c):** Ha egy player alatt eltűnik a node akkor ez a metódus fogja visszarakni a megfelelő Nodera.
* **int playerCount():** Visszaadja a játékosok számát.
* **void printPlayers():** Kiírja a pályán lévő játékos nevét és hogy melyik Nodeon áll.
* **void clearMap():** Kiüríti a pályát.
* **Caharacter getPlayer(int idx):** Visszaadja az adott indexű játékost.
* **void loadMap(string fileName):** Betölt a megadott nevű fájlból egy pályát.
* **void saveMap(string fileName):** Kimenti a megadott nevű fájlba a pályát.
* **void printNodes():** Kiírja a pálya elemeinek, azaz a Nodeokank a koordinátáit és a uuidket.

### Node

#### Felelősség

A Node osztály egy absztrakt osztály, ezt implementálja minden osztály, amik a elemei lesznek, és játéktérre tehetjük.

#### Ősosztályok

Nincs ősosztály.

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **logger:** egy logger ehhez az osztályhoz.
* **uuid:** a node egyedi azonosítója.
* **characters:** a jelenleg ezen az nodeon álló karakterek listája.
* **neighbours:** a jelenleg ezzel a node-dal szomszédos és a node-ba csatlakoztatott más nodeok listája.
* **sources:** azon nodeok, ahonnan folyik víz ebbe a nodeba.
* **absorbers:** azon nodeok, ahová folyik víz ebből a nodeba.
* **maxCharacters:** az ezen a nodeon a maximálisan elfogadott karakterek száma.
* **maxConnections:** az ehhez a nodehoz maximálisan csatlakoztatható elemek száma.
* **isBroken:** A node állapota, műkódik e vagy sem.
* **flowRate:** hogy éppen mennyi víz folyik keresztül a nodeon.
* **x:** Node x koordináltja a pályán.
* **y:** Node y koordináltja a pályán.

#### Metódusok

* **String getUuid()**: visszaadja az uuid attribútum értékét.
* **ArrayList<Character> getCharactersOn():** Visszaadja a nodeon álló karakterke listáját.
* **ArrayList<Node> getNeighbours():** Visszaadja a node-hoz kötött nodeok listáját.
* **void placeCharacter(Character character):** elhelyez egy karaktert a nodeon. Játék inicializáláakor használatos, vagy ha a játékos alól eltűnt elem miatt vissza kell helyezni a kezdő nodera.
* **void addCharacter(Character character):** Hozzáad egy karaktert a characters listához, ha még fér karakter a nodera.
* **void removeCharacter(Character character):** kivesz egy karaktert a characters listából.
* ***void repairNode(Character character):*** Megjavítja az objektumot. Absztrakt, leszármazottak implementálják.
* ***void breakNode(Character character):*** Elrontja az objektumot. Absztrakt, leszármazottak implementálják.
* **void addFlowRate(Node source, double excededFlow):** Növeli a flowRate-et a megadott értékkel.
* **void removeFlowRate(Node source, double flowRate):** Csökkenti a flowRate-et a megadott étékkel.
* **void setFlowRate(double flowRate):** Beállítja a flowRate-et a megadott értékre.
* **void tick():** Időközönként, azaz körönként számolja a flowRate-et.
* **void connect(Node node):** Hozzácsatlakoztatja a megadott Nodeot.
* **void disconnect(Node node):** Lecsatlakoztatja a megadott Nodeot.
* ***void setSurface(string surface, Character c):*** Absztrakt metódus amit a leszármazottak implementálnak. Célja, hogy a karakterek tudják ragacsossá/csúszóssá teni a pályaelemek egy részét.

### Nomad

#### Felelősség

A szabotőr nomád-karaktereket megvalósító osztály. A nomád ki tudja lyukasztani a csöveket (Pipe), el tudja állítani a pumpákat és ragacsossá illetve csúszóssá tudja tenni a csövetket.

#### Ősosztályok

Character → Nomad .

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **name**: karakter neve
* **uuid**: karakter egyedi azonosítója
* **standingOn**: eltárolja, hogy a karakter a Map mely elemén, melyik Node-on áll.
* **stuckedInPipe**: random generált idő, ameddig a karakter nem léphet el a csőről ha az ragacsos.

#### Metódusok

* **void setPump(Pipe in, Pipe out)**: A nomád beállítja a Pumpán, amin áll. A Pumpához csatlakoztatott csövek közül melyiket nyitja meg, hogy a pumpába ki és be- folyjon a víz.
* **void makePipeSlippery()**:A nomád csúszóssá teszi a cső felszínét.

### Plumber

#### Felelősség

Ez az osztály a játékban lévő vízszerelők osztálya. A vízszerelők meg tudnak javítani csöveket és pumpákat, le tudnak rakni új csöveket és pumpákat, illetve beállíthatják a pumpákat, hogy a ciszternákba tereljék a vizet. A már lerakott csövek átmozgatásával is tudják a vizet a ciszternába terelni, továbbá ők is tudnak csövet lyukasztani

#### Ősosztályok

Character → Plumber

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok

* **name**: karakter neve
* **uuid**: karakter egyedi azonosítója
* **standingOn**: eltárolja, hogy a karakter a Map mely elemén, melyik Node-on áll.
* **stuckedInPipe**: random generált idő, ameddig a karakter nem léphet el a csőről ha az ragacsos.
* **pickedUpPump:** A plumber inventoryában a ciszternán felvett pumpákat tárolja.
* **pickedUpPipe:** A plumber inventoryában a mindkét végén lecsatlakoztatott majd felvett csöveket tárolja.
* **draggedPipe:** Az egyik végén lecsatlakoztatott, és átmozgatott cső.

#### Metódusok

* **void repair()**: megjavítja a Node-ot, amin áll.
* **void placePump()**: elhelyez egy pumpát oda, ahol áll. Ha csövön áll sikeresen lehelyezi, ha més nodeon akkor kivételt dob.
* **void placePipe()**: Lehelyezi egy csövet a pályára, és hozzáköti ahhoz a nodehoz amin áll.
* **void pickupPump():** Megpróbál felvenni egy pumpát. Ha nem ciszternán áll akkor kivételt dob.
* **void pickupPipe():** Lecsatlakoztatja a cső egyik végét és elhúzza egy másik elemhez. Ha a cső mindkét vége disconnected akkor felveszi az egészet az inventoryába.
* **void setPump(Pipe in, Pipe out)**: A plumber beállítja a Pumpán, amin áll. A Pumpához csatlakoztatott csövek közül melyiket nyitja meg, hogy a pumpába ki és be- folyjon a víz.

### Pipe

#### Felelősség

Ez az osztály implementálja a játékban a csöveket. A csöveket ki lehet lyukasztani és meg lehet javítani, továbbá csatlakoztatni lehet őket más objektumokhoz (összesen maximum kettőhöz), és összesen 1 játékos állhat rajtuk. A csöveket a játékosok ragacsossá tudják tenni, továbbá a nomad típusú játékosok csúszóssá is.

#### Ősosztályok

Node → Pipe

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok: *A felsoroltokon kívül rendelkezik még a Node absztrakt ősosztály attribútumaival.*

* **capacity**: Azt mutatja mennyi a kapacitása ennek a csőnek (vagyis azt, mennyi víz fér bele).
* **isSlippery:** Csúszós-e a cső felszíne.
* **isStickey:** Ragacsos-e a cső felszíne.
* **unbreakableTill:** Cső megfoltozásakor generált véletlen szám, ameddig nem lehet újra kilyukasztani.

#### Metódusok

* **double getCapacity()**: Visszaadja a capacity értékét, vagyis azt, mennyi víz fér még bele.
* **void repairNode(Character character)**: Megjavítja/megfoltozza a csövet és generál egy véletlen számot, ameddig nem lehet újra kilyukasztani, ezt a számot a unbreakableTill-be menti.
* **void breakNode(Character character)**: Kilyukasztja, elrontja a csövet akkor, ha az unbreakableTill 0, egyéb különben nem lehet lyukasztani
* **void tick()**: Körönként számolja a flowRatet, és csökkenti a unbreakableTill-t ha az nem érte el a 0-t.
* **void setSurface(string type, Character c):** A cső felszínét állítja be a paraméterként beadott típusúra. Ha a type paraméter sticky akkor az isStickey lesz igaz, ha slippery akkor az isSlippery, ha pedig none akkor mind a kettő false lesz.

### Pump

#### Felelősség

Ez a játékban szereplő pumpák osztálya. Egy pumpa állítható, el tud romlani, és meg lehet javítani. Egy pumpán több játékos is állhat egyszerre.

#### Ősosztályok

Node → Pump

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok: *A felsoroltokon kívül rendelkezik még a Node absztrakt ősosztály attribútumaival.*

* **internalBufferLevel**: ezt arra használjuk, hogy a flowRate-et számoljuk.
* **activeInput**: ez a pumpa bemenetei közül tárolja azt a csövet, amelyik aktív.
* **activeOutput**: a pumpa kimenetei közül tárolja azt a csövet, amelyik aktív.

#### Metódusok

* **void repairNode(Character character)**: Megjavítja a pumpát.
* **void breakNode(Character character)**: Elrontja a pumpát.
* **void setActiveInput(Pipe p)**: Beállítja az activeInput-ot, azaz hogy, a szomszédjai közül a megadott csőből folyjon be víz a pumpába.
* **void setActiveOutput(Pipe p)**: Beállítja az activeOutput-ot azaz hogy, a szomszédjai közül a megadott csőbe folyjon ki víz a pumpából.
* **void calculateFlowRate()**: Ha a pumpa nincs elromolva és van aktív bemeneti csöve, akkor megnézi hogy van e aktív kimeneti csöve. Ha van akkor beállítja a pumpán átfolyó víz mennyiségét arra, ami a bemeneti csőben van. Ha nincs aktív kimenet akkor az internalBufferLevel-t növeli. Ha csak aktív kiemet van akkor a flowRate vagy az internalBufferLevel lesz vagy a kimeneti cső flowRatje, attól függően hogy melyik a kevesebb.
* **void tick()**: Időközönként (random) elrontja a pupmpát, és számolja a flowRate-et.

### Tickable

#### Felelősség

Ez az interfész minden objektumnak, aminek időközönként kell csinálnia valamit.

#### Ősosztályok

Nincsenek ősosztályok.

#### Metódusok

* **void tick()**: Az adott időközönként végzett tevékenységre való megvalósítandó metódus.

### WaterSpring

#### Felelősség

A WaterSprring egy olyan speciális elem, amivel a ciszternát tudjuk feltölteni. Nem lehet sem elrontani, sem megjavítani, illetve nincs maximum kapacitása.

#### Ősosztályok

Node → WaterSpring

#### Interfészek

Tickable interfészt implementálja.

#### Attribútumok: *A Node absztrakt ősosztály attribútumaival rendelkezik.*

#### Metódusok

* **void addCharacter(Character character)**: Hozzáad egy karaktert a charactersOn listához.
* **void repairNode(Character character)**: Beállítjuk, hogy ez nem lehetséges, kivételt fog dobni.
* **void breakNode(Character character)**: Beállítjuk, hogy ez nem lehetséges, kivételt fog dobni.
* **void tick()**: Időközönként számolja a flowRate-et.

## A tesztek részletes tervei, leírásuk a teszt nyelvén

[A tesztek részletes tervei alatt meg kell adni azokat a bemeneti adatsorozatokat, amelyekkel a program működése ellenőrizhető. Minden bemenő adatsorozathoz definiálni kell, hogy az adatsorozat végrehajtásától a program mely részeinek, funkcióinak ellenőrzését várjuk és konkrétan milyen eredményekre számítunk, ezek az eredmények hogyan vethetők össze a bemenetekkel. A tesztek leírásakor az előző dokumentumban (proto koncepciója) megadott szintakszist kell használni.]

### Teszteset1

* **Leírás**

[szöveges leírás, kb. 1-5 mondat.]

* **Ellenőrzött funkcionalitás, várható hibahelyek**
* **Bemenet**

[a proto bemeneti nyelvén megadva (lásd előző anyag)]

* **Elvárt kimenet**

[a proto kimeneti nyelvén megadva (lásd előző anyag)]

### Teszteset2

* **Leírás**

[szöveges leírás, kb. 1-5 mondat.]

* **Ellenőrzött funkcionalitás, várható hibahelyek**
* **Bemenet**

[a proto bemeneti nyelvén megadva (lásd előző anyag)]

* **Elvárt kimenet**

[a proto kimeneti nyelvén megadva (lásd előző anyag)]

## A tesztelést támogató programok tervei

[A tesztadatok előállítására, a tesztek eredményeinek kiértékelésére szolgáló segédprogramok részletes terveit kell elkészíteni.]

## Napló

|  |  |  |  |
| --- | --- | --- | --- |
| **Kezdet** | **Időtartam** | **Résztvevők** | **Leírás** |
| 2023.04.26. 12:00 | 10 perc | Farkas  Kis  Sőregi  Nyilas | Értekezlet.  Heti munka felosztása |
| 2023.04.26. 12:15 | 2,5 óra | Farkas | Osztályok, metódusok dokumentálása. |