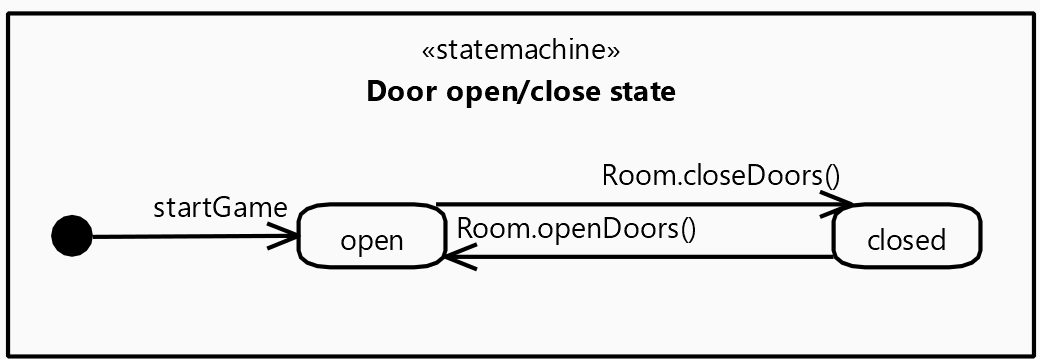
# Részletes tervek

## Osztályok és metódusok tervei.

### Door

#### Felelősség

Az ajtót reprezentáló osztály, amely lehetővé teszi a karakterek számára, hogy szobák között tudjanak közlekedni. Mágikus szobák ajtói véletlen időközönként bezáródnak majd újra kinyílnak.



#### Ősosztályok

-

#### Interfészek

-

#### Attribútumok

* **- bool isOneWay**: Meghatározza, hogy egy ajtót csak az egyik irányba lehet használni vagy mindkét irányba. Ha az értéke igaz akkor egyirányú, ha pedig hamis akkor kétirányú.
* **- bool isClosed**: Meghatározza, hogy egy ajtó csukva van vagy sem. Csukott ajtót nem tudnak a karakterek használni. Ha az értéke igaz akkor csukva van, ha pedig hamis akkor nyitva van.
* **- List<Room> rooms**: Meghatározza, hogy melyik két szobát köti össze az ajtó.

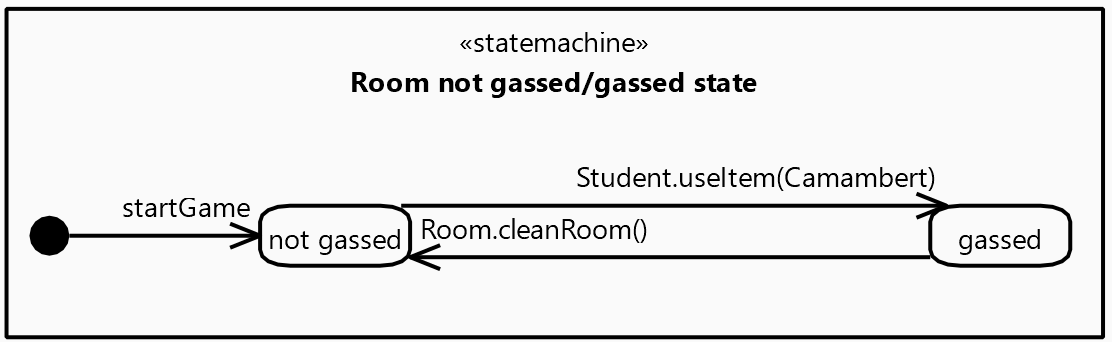
#### Metódusok

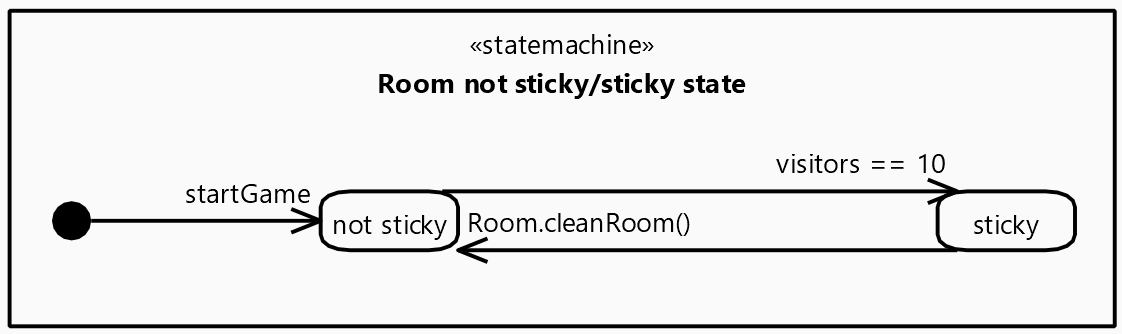
* **+ void changeRoom(Character c, Room r)**: A megadott c karaktert átlépteti az r szobából annak a párjába. Amennyiben ez sikeres akkor beállítja a c karakternek az új szobát, törli a c karaktert az r szobából és hozzáadja a c karaktert az új szobához.
* **+ void replaceRoom(Room x, Room y)**: Kicseréli az x szobát y szobára, azaz megváltoztatja, hogy melyik két szobát köti össze az ajtó.
* **+ void setConnectedRooms(Room existing, Room \_new)**: Beállítja, hogy az ajtó melyik két szobát kösse össze. Az ajtó existing és a \_new szobát fogja összekötni.

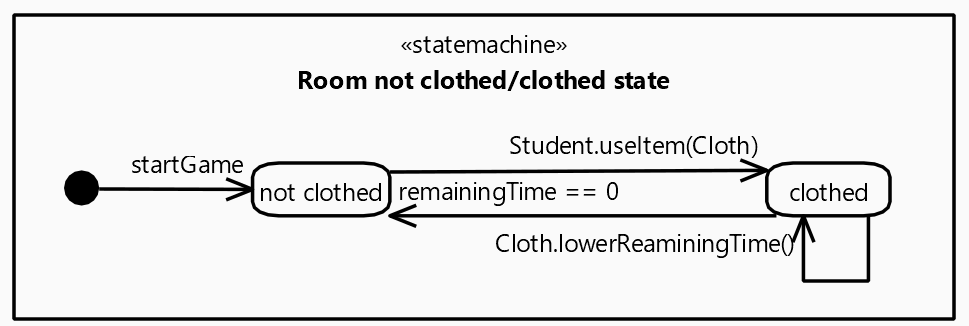
### Room

#### Felelősség

A szobát reprezentáló osztály, amely tárolja a benne található karaktereket, ajtókat, illetve tárgyakat. A szobát be lehet gázosítani, illetve ezt meg lehet szüntetni. A szobát egy időre rongyossá lehet tenni és adott számú látogató után ragacsos lesz.







#### Ősosztályok

-

#### Interfészek

-

#### Attribútumok

* **- bool gassedRoom**: Meghatározza, hogy egy szoba gázos vagy sem. Ha értéke igaz akkor gázos, ha pedig hamis akkor nem gázos.
* **- int clothedRoom**: Meghatározza, hogy egy szoba rongyosságából mennyi idő van hátra. Ha az értéke 0 akkor a szoba nem rongyos.
* **- int visitors**: Meghatározza, hogy a szoba tisztítása óta mennyi karaktert járt a szobában. Takarítás után az értéke 0.
* **- int maxCapacity**: Meghatározza, hogy egy szobában egyszerre maximum mennyi karakter tartózkodhat. Ha ez az érték megegyezik a szobában lévő karakterek számával akkor a szoba tele van, nem léphet be új karakter.
* **- bool magical**: Meghatározza, hogy egy szoba mágikus vagy sem. Ha értéke igaz akkor mágikus, ha pedig hamis akkor nem. Mágikus szobának az ajtói véletlen időközönként bezáródnak.
* **- bool sticky**: Meghatározza, hogy egy szoba ragacsos vagy sem. Ha értéke igaz akkor ragacsos, ha hamis akkor nem. Ragacsos szobában nem tudnak a karakterek tárgyakat felvenni.
* **- List<Door> doors**: Meghatározza, hogy melyik ajtók vannak a szobában. Ebből fogjuk tudni megkapni a szomszédos szobákat.
* **- List<Item> items**: Meghatározza a szobában lévő tárgyakat.
* **- List<Character> characters**: Meghatározza a szobában lévő karaktereket. Ennek a méretének nem szabad nagyobbnak lennie, mint a szoba kapacitása.

#### Metódusok

* **+ int getMaxCapacity()**: Visszaadja a maxCapacity értékét, azaz azt, hogy mekkora a szoba befogadó képessége.
* **+ int getclothedRoom()**: Visszaadja a clothedRoom értékét, azaz azt, hogy ha a szoba rongyos akkor még mennyi ideig.
* **+ bool getgassedRoom()**: Visszaadja a gassedRoom értékét, azaz azt, hogy a szoba gázos vagy sem.
* **+ void setgassedRoom(bool b)**: A paraméterül kapott értékre beállítja a gassedRoom értékét.
* **+ List<Character> getCharacters()**: Visszaadja a characters listát, azaz visszaadja a szobában tartózkodó karaktereket egy listában.
* **+ List<Item> getItems()**: Visszaadja az items listát, azaz visszaadja a szobában lévő tárgyak listáját.
* **+ List<Door> getDoors()**: Visszaadja a doors listát, azaz a szobában lévő ajtók listáját.
* **+ void removeItem(Item i)**: Törli az items listából a paraméterül kapott Item-et amennyiben az benne volt a listában.
* **+ void addItem(Item i)**: Hozzáadja az items listához a paraméterül kapott Item-et amennyiben annak az értéke nem null.
* **+ bool characterEnters(Character c)**: Megnézi, hogy a szobába befér egy új karakter vagy sem, azaz a szoba kapacitása nagyobb-e mint a szobában tartózkodó karakterek száma. Ha befér true-val tér vissza, ha nem akkor false-al.
* **+ void addCharacter(Character c)**: Hozzáadja a characters listához a paraméterül kapott Character-t amennyiben annak az értéke nem null. Ha a szoba gázos akkor meghívja a karakter megfelelő függvényét, illetve, ha van a karakternek valami egyéb kötelessége akkor azokat.
* **+ void removeCharacter(Character c)**: Törli a characters listából a paraméterül kapott Character-t amennyiben az benne volt a listában.
* **+ void addDoor(Door d)**: Hozzáadja az doors listához a paraméterül kapott Door-t amennyiben annak az értéke nem null.
* **+ void removeDoor(Door d)**: Törli a doors listából a paraméterül kapott Door-t amennyiben az benne volt a listában.
* **+ void split()**: Egy szobát ketté választ, azaz létrehoz egy új szobát aminek a kapacitása és állapotai(gassedRoom, clothedRoom, visitors, magical, sticky) megegyeznek az eredeti szoba állapotaival. Az eredeti szoba ajtóiból minden második az új szobának lesz az ajtaja. A tárgyak és a karakterek maradnak az eredeti szobában.
* **+ void dropThemOut()**: Megnézi, hogy a szobában lévő diákok meg tudják védeni magukat vagy sem. Ha ronggyal védekezik valaki akkor a szoba clothedRoom attribútuma nő. Akik nem tudták magukat megvédeni azokat törli a characters listából és eldobatja velük a náluk lévő tárgyakat.
* **+ void clothThem()**: A szobában lévő oktatók clothed állapotát beállítja a szoba clothedRoom értékére.
* **+ void mergeWithRoom(Room r)**: A szobát összeolvasztja a paraméterül kapott szobával úgy, hogy az int típusú attribútumokból a nagyobbat veszi alapul, a bool típusúakból pedig ha valamelyik igaz akkor azt. Az összeolvasztást csak akkor hajtja végre, ha a két szoba minden karaktere belefér az összevont szobába. Az összes karakter, tárgy és ajtó hozzáadódik a szoba megfelelő listájához.
* **+ void closeDoors()**: Bezárja a szoba összes ajtaját, azaz az összes ajtó isClosed attribútumát true-ra állítja.
* **+ void openDoors()**: Kinyitja a szoba összes ajtaját, azaz az összes ajtó isClosed attribútumát false-ra állítja.
* **+ void setSticky(bool b)**: A paraméterül kapott értékre beállítja a sticky értékét.
* **+ bool getSticky()**: Visszaadja a sticky értékét, azaz azt, hogy a szoba ragacsos vagy sem.
* **+ void cleanRoom()**: Minden olyan karaktert aki nem bénult/ájult kidob a szobából, beállítja a visitors értékét 0-ra, illetve a gassedRoom és a sticky értékét pedig false-ra állítja.
* **+ void increaseVisitors()**: A szoba visitors attribútumát növeli eggyel. Ez akkor fog bekövetkezni, ha egy karakter belép a szobába.

## 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 program tesztelése úgy történik, hogy a program teszteseteket olvas be (az előző pontban specifikált bemeneti nyelv szerint), ezután a programot futtatva végrehajtja az adott teszt esetet és ad egy kimenetet. Minden teszt esethez tartozik egy elvárt kimenet (az előző pontban specifikált kimeneti nyelv szerint). A kapott kimenetet összehasonlítjuk az elvárt kimenettel és ha ez a kettő nem egyezik meg akkor a teszt lefutását sikertelennek tekintjük. Ha viszont a kapott kimenet és az elvárt kimenet megegyezik akkor a tesztet sikeresnek tekintjük. Az összehasonlítást kézzel végezzük, mint felhasználó/tesztelő személy.Napló

|  |  |  |  |
| --- | --- | --- | --- |
| **Kezdet** | **Időtartam** | **Résztvevők** | **Leírás** |
| 2010.03.21. 18:00 | 2,5 óra | Horváth  Németh  Tóth  Oláh | Értekezlet.  Döntés: Horváth elkészíti az osztálydiagramot, Oláh a use-case leírásokat. |
| 2010.03.23. 23:00 | 5 óra | Németh | Tevékenység: Németh implementálja a tesztelő programokat. |
| … | … | … | … |