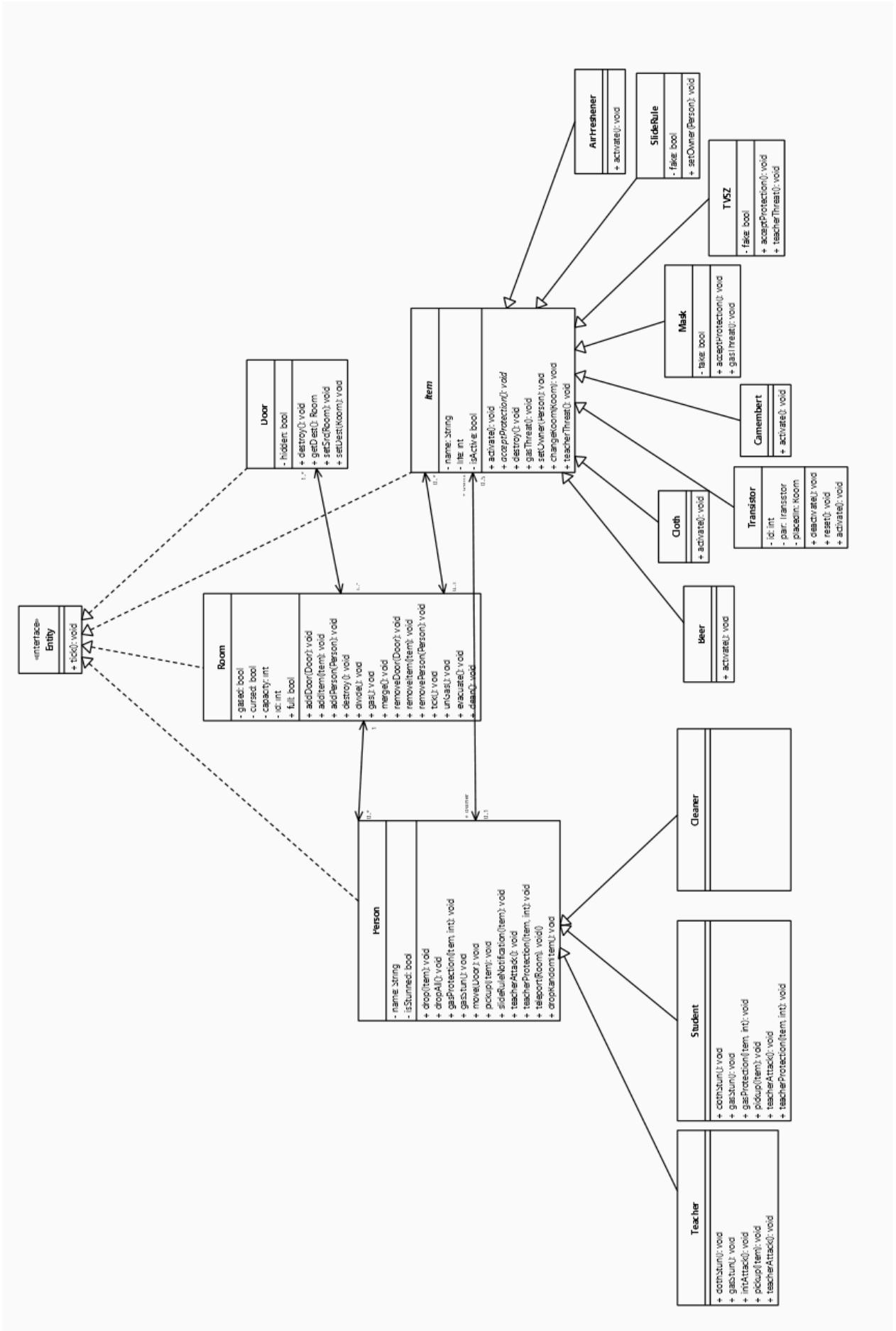
7. Prototípus koncepciója

7.0 Változás hatása a modellre

7.0.1 Módosult osztálydiagram

Új osztályok: AirFreshener, Cleaner
Új attribútumok: Room:cleanliness, TVSZ:fake, Mask:fake, SlideRule:fake


«interface»
Entity
+ tick(): void
Door
- hidden: bool
+ destroy(): void
+ getDest(): Room
+ setSrcRoom(): void
+ setDestRoom(): void
Item
- name: string
- life: int
- isActive: bool
+ activate(): void
+ acceptProtection(): void
+ destroy(): void
+ gasThreat(): void
+ setOwner(Person): void
+ changeRoom(Room): void
+ teacherThreat(): void
Airfreshener
+ activate(): void
SlideRule
- fake: bool
+ setOwner(Person): void
TVSZ
- fake: bool
+ acceptProtection(): void
+ teacherThreat(): void
Mask
- fake: bool
+ acceptProtection(): void
+ gasThreat(): void
Camembert
+ activate(): void
Transistor
- id: int
- pair: transistor
- placedIn: Room
+ deactivate(): void
+ reset(): void
+ activate(): void
Cloth
+ activate(): void
Beer
+ activate(): void
Room
- gased: bool
- cursed: bool
- capacity: int
- id: int
+ full: bool
+ addDoor(Door): void
+ addItem(Item): void
+ addPerson(Person): void
+ destroy(): void
+ divide(): void
+ gas(): void
+ merge(): void
+ removeDoor(Door): void
+ removeItem(Item): void
+ removePerson(Person): void
+ lock(): void
+ unGas(): void
+ evacuate(): void
+ clean(): void
Person
- name: string
- isStunned: bool
+ drop(Item): void
+ dropAll(): void
+ gasProtection(Item, int): void
+ gasStun(): void
+ moveLDoor(): void
+ pickup(Item): void
+ sideRuleNotification(Item): void
+ teacherAttack(): void
+ teacherProtection(Item, int): void
+ teleportRoom(): void()
+ dropRandom(Item): void
Cleaner
Student
+ cotStun(): void
+ gasStun(): void
+ gasProtection(Item, int): void
+ pickup(Item): void
+ teacherAttack(): void
+ teacherProtection(Item, int): void
Teacher
+ cotStun(): void
+ gasStun(): void
+ intAttack(): void
+ pickup(Item): void
+ teacherAttack(): void

7.0.2 Új vagy megváltozó metódusok

Person:

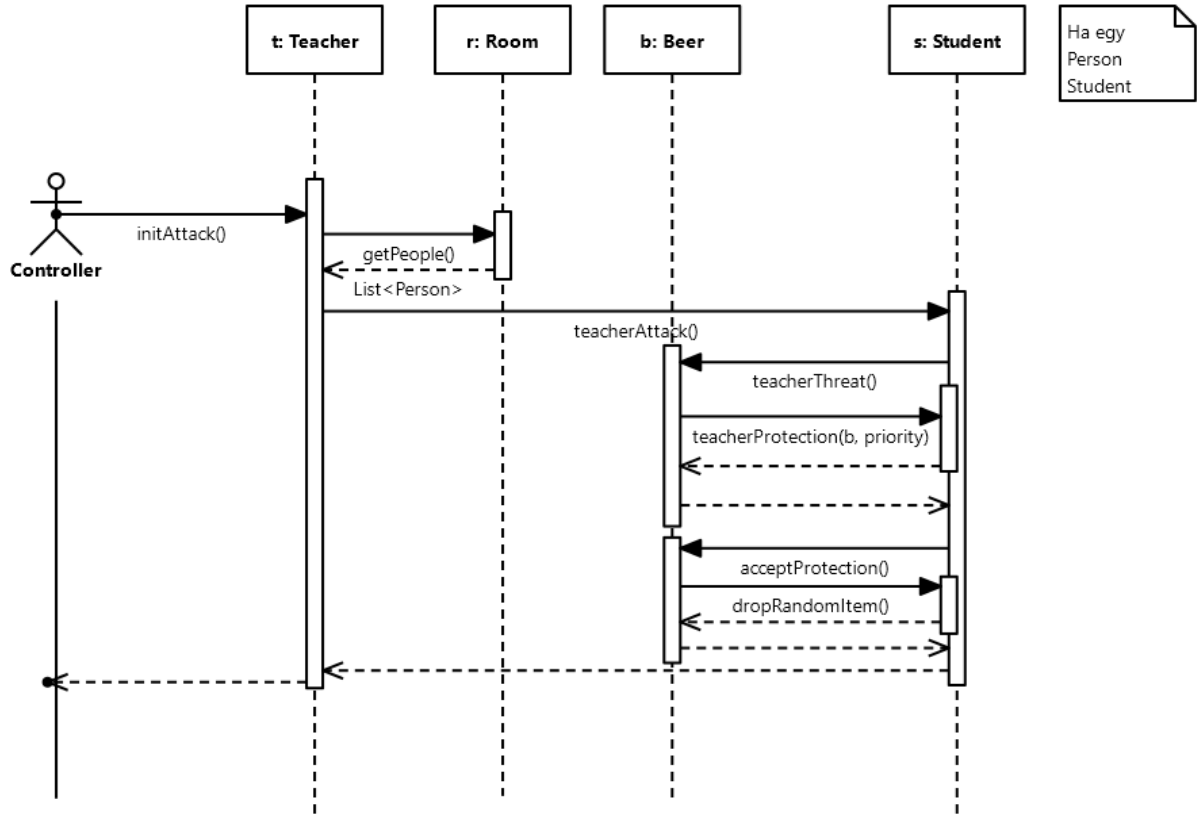
+ dropRandomItem(): void
  Inventoryból eldob egy tetszőleges tárgyat, támaszkodik a drop(Item) implementációjára

Room:

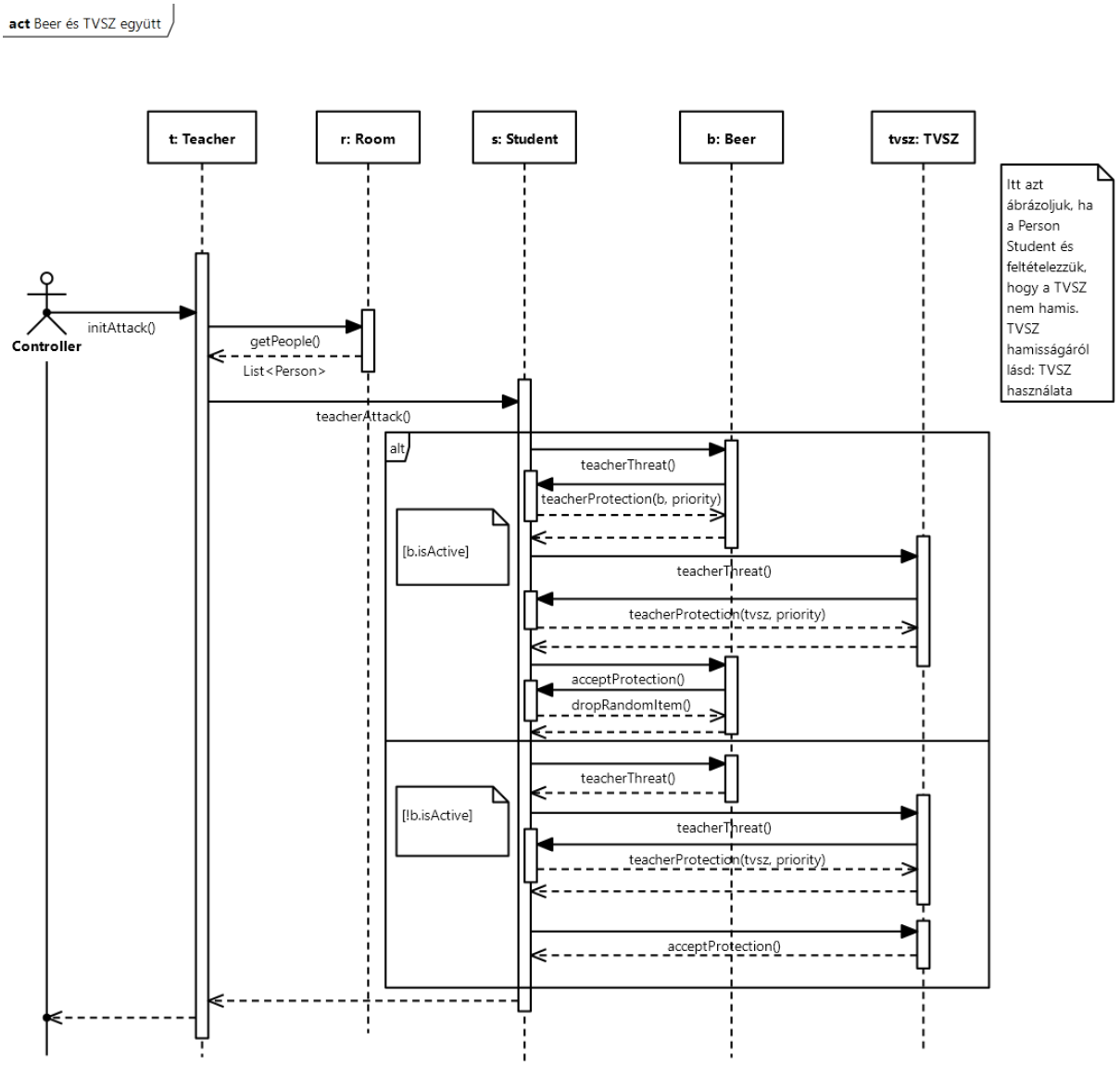
+ unGas(): void
  Gázmentesíti a szobát
+ evacuate(): void
  Kiüríti az embereket, minden mozogni képes Person átkerül egy szomszédos szobákba
+ clean(): void
  A cleanliness-t reseteli 10-re

7.0.3 Szekvencia-diagramok

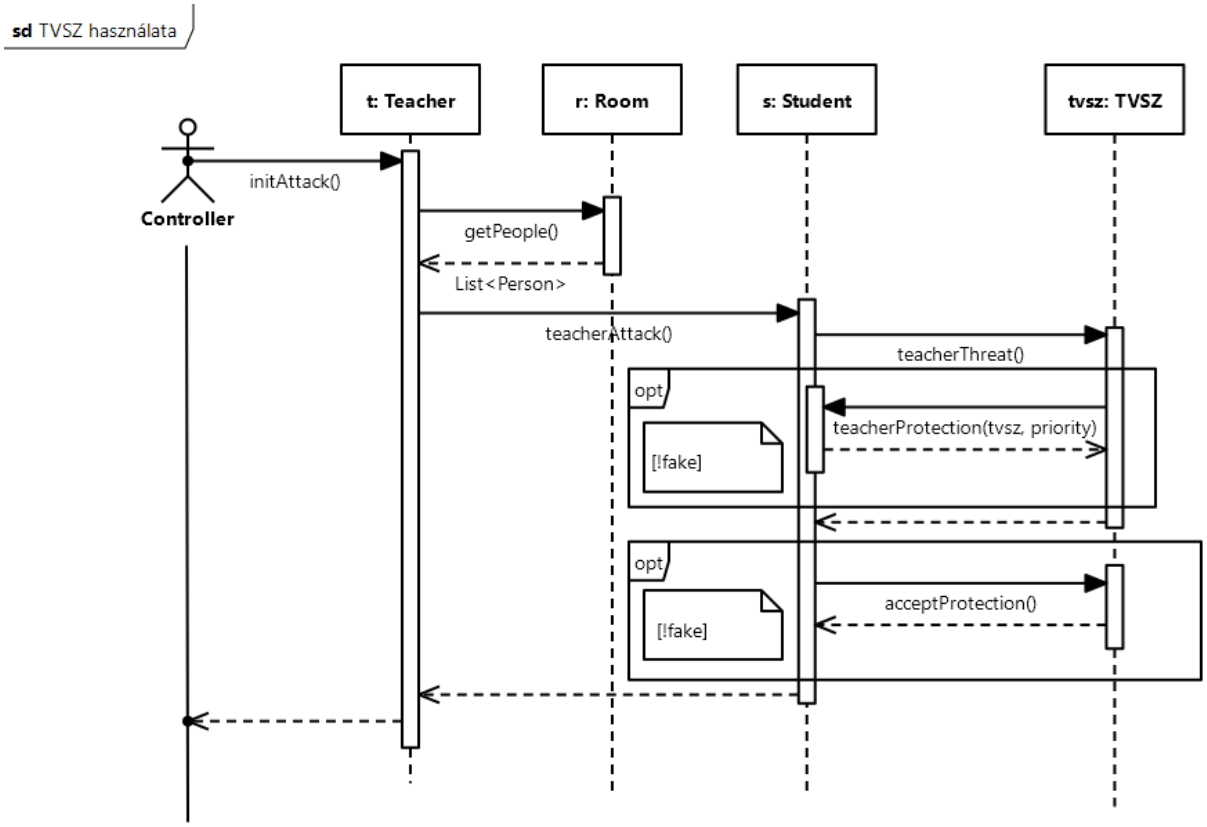
Beer használata



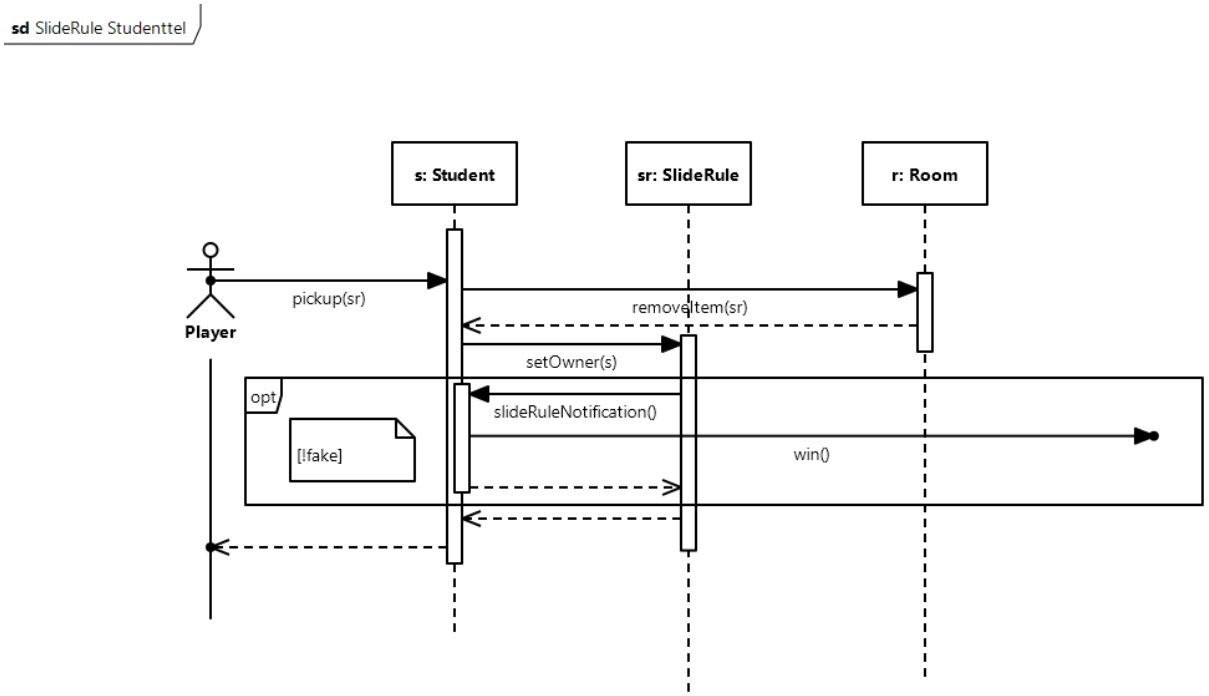
Beer és TVSZ együtt


act Beer és TVSZ együtt
t: Teacher
r: Room
s: Student
b: Beer
tvsz: TVSZ
Itt azt ábrázoljuk, ha a Person Student és feltételezzük, hogy a TVSZ nem hamis. TVSZ hamisságáról lásd: TVSZ használata
Controller
initAttack()
getPeople()
List<Person>
teacherAttack()
alt
teacherThreat()
teacherProtection(b, priority)
[b.isActive]
teacherThreat()
teacherProtection(tvsz, priority)
acceptProtection()
dropRandomItem()
teacherThreat()
[!b.isActive]
teacherThreat()
teacherProtection(tvsz, priority)
acceptProtection()

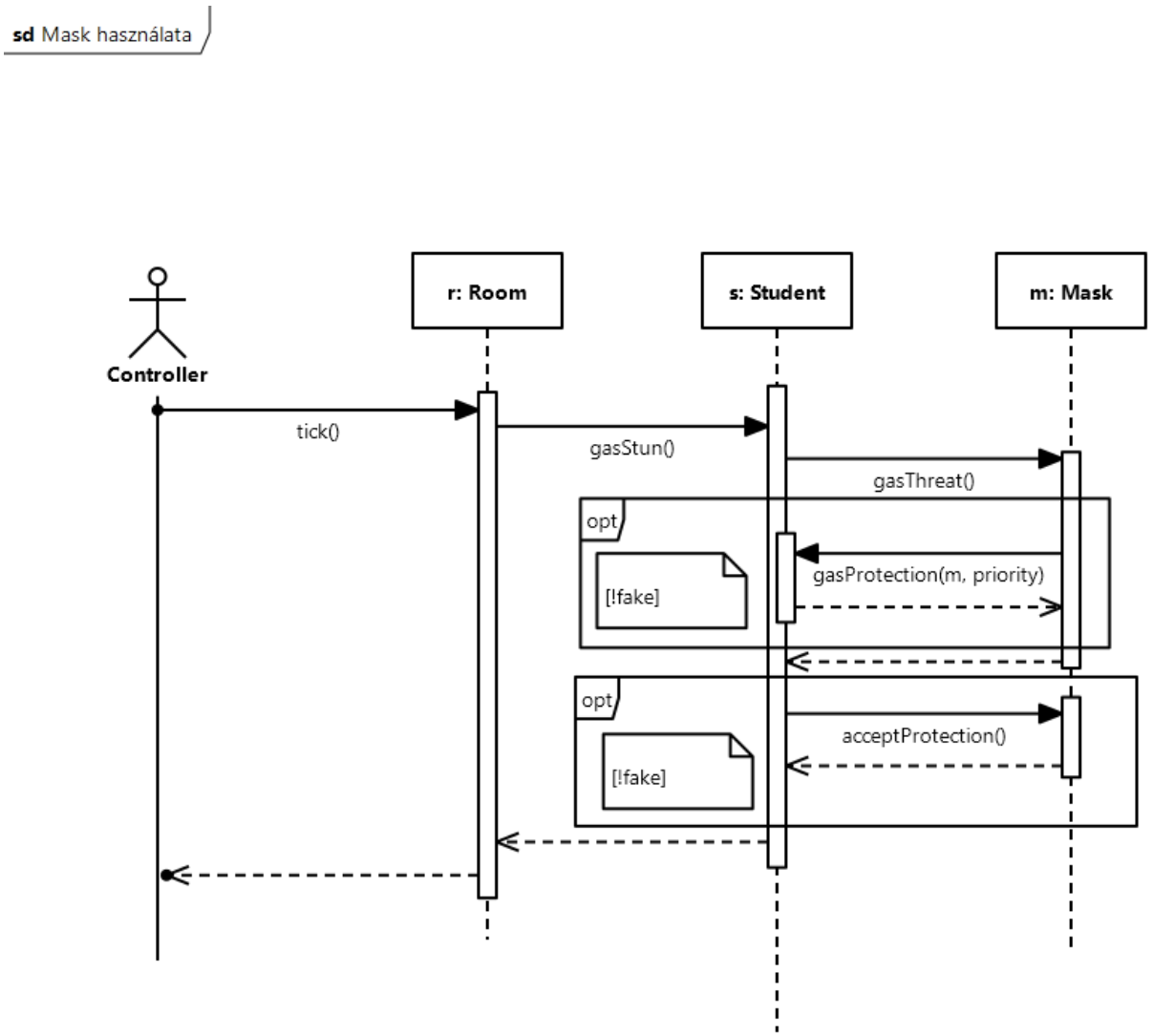
TVSZ használata



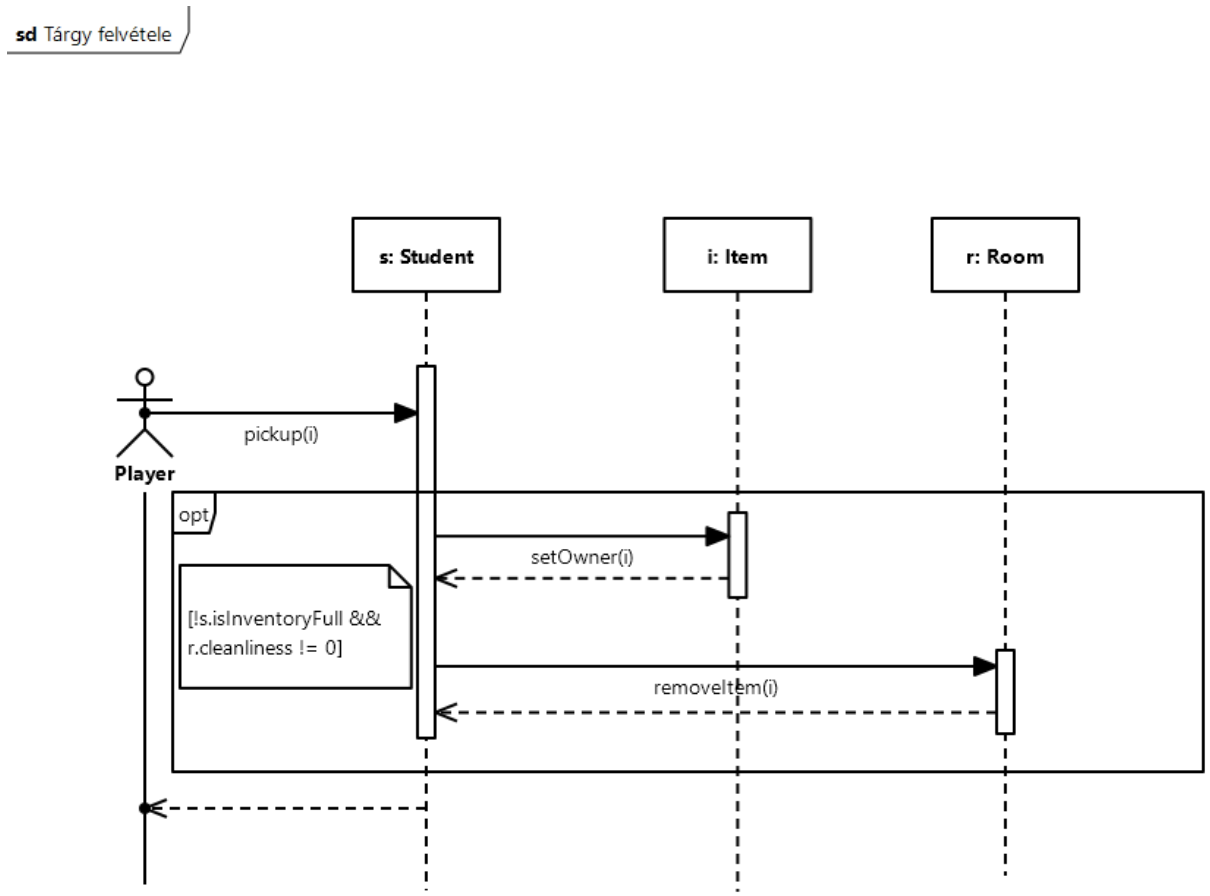
SlideRule Studenttel (Teacherrel nem történt változás)



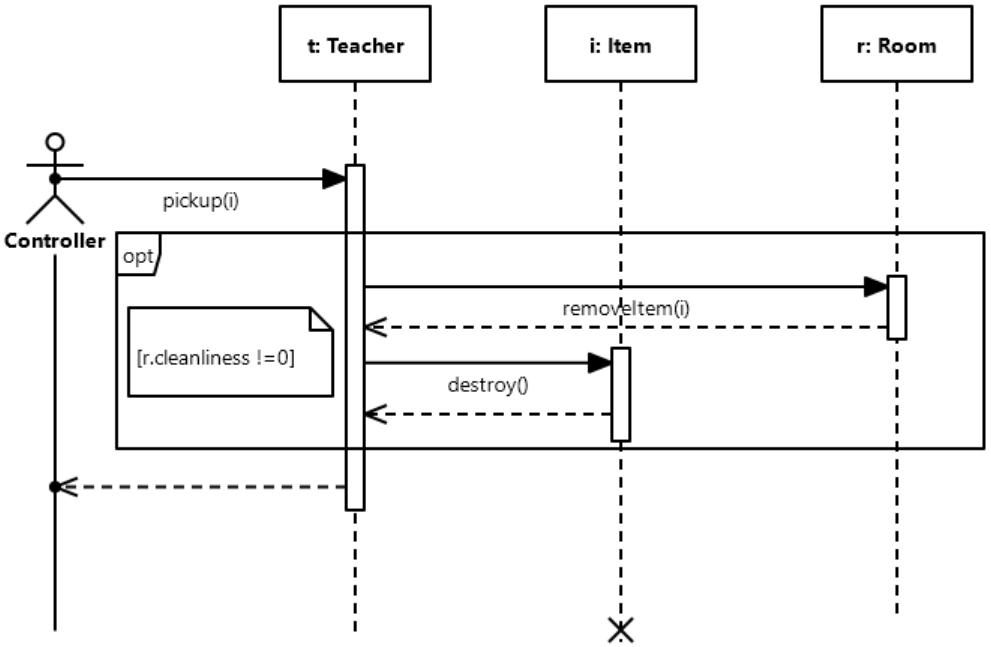
Maszk használata



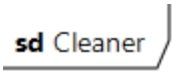
Tárgy felvétele hallgató által

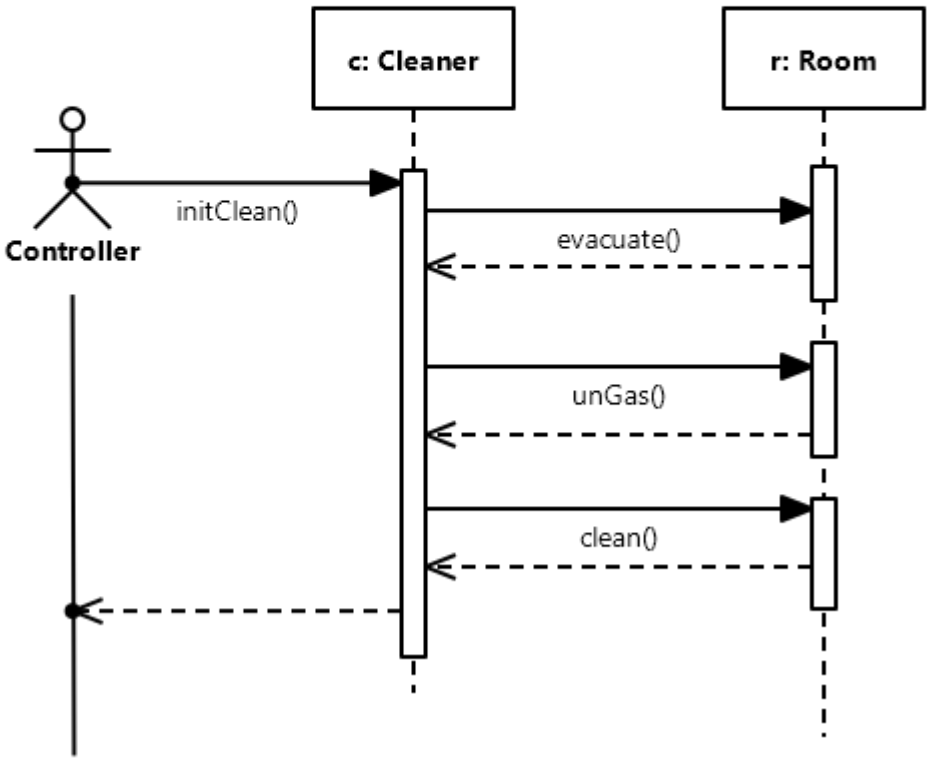


Tárgy felvétele oktató által

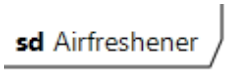


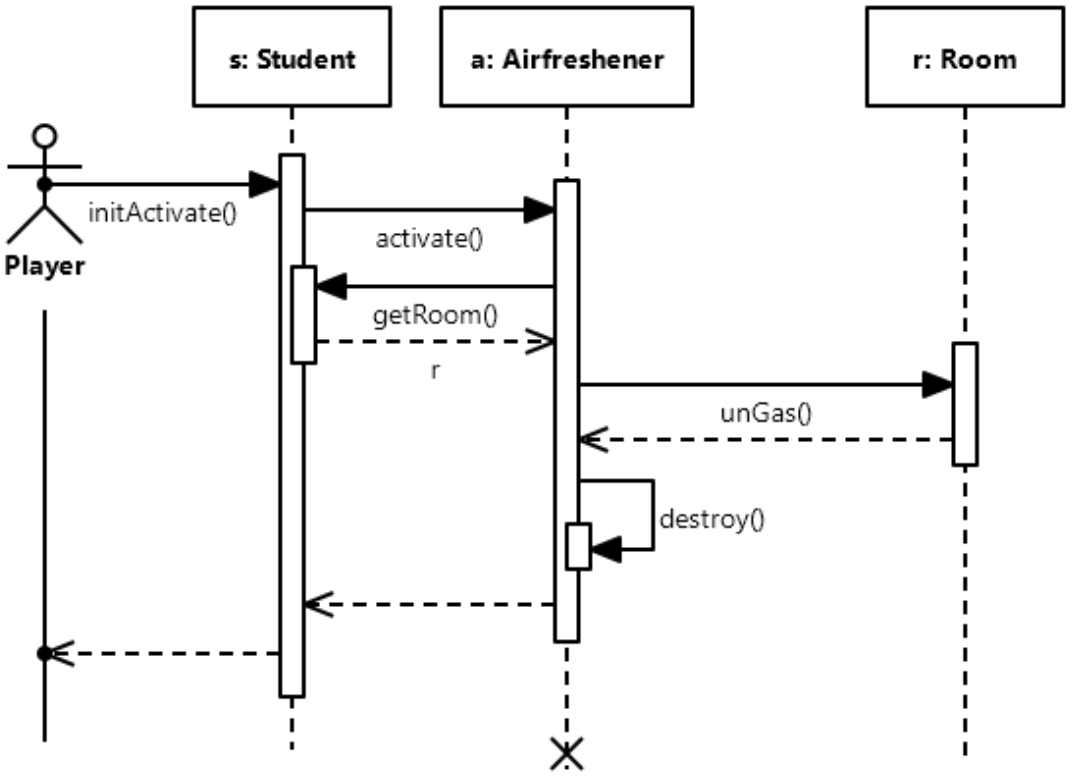
Takarító takarít





Légfrissítő használata





7.1 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.]

7.1.1 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.]

A prototípus program karakteres interfészének 2 fő célja van, az egyik a játékprogram működésének demonstrációja, a másik pedig a tesztelés támogatása. Előbbi célt az interfésznek a parancssori shell üzemmódja támogatja, mely a tényleges játék-vezérlővel együttműködve a játék szabályait betartva vezet le egy játékot, ahol a játékosok felhasználóbarát parancsokat kiosztva irányíthatják hallgatóikat és minden egyes parancsról visszajelző kimenetet kapnak, mely a játékos tájékozódását segíti. Utóbbi célra pedig a tesztelési üzemmód szolgál: ilyenkor a bemeneten az összes, a játékban történő eseményt le

kell írni (**teljesen determinisztikus működés**), a visszajelzés pedig a lefutást követő egyszeri output, mely a végállapotot részleteiben leírja az állapotleíró nyelven. Ezt a célt támogatják olyan parancsok, melyekkel a bemenetet fájlból olvashatjuk és a kimenetet fájlba írhatjuk. Egy játékállapot-definiáló nyelvet is definiálunk, mely mindkét üzemmódban egyaránt használható mind a shellből mind pedig egy külső fájlból használható az inicializációs állapot megadására. Ez a játékállapot-definíció kétirányban is működik: ugyanaz a szintakszisa a bemeneti inicializálásnak és az állapot-lekérdezés kimenetének. Ez teszi lehetővé a játékállás elmentését-visszaállítását. Dönthetünk úgy is, hogy nem definiálunk teljes inicializációt, ilyenkor csak a játékosok megadása szükséges. Ekkor életbe lép a játék saját pályageneráló algoritmusa. Teszt üzemmódban a teljes inicializáció megadása kötelező.

7.1.2 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).]

Bár a felhasználói interfész két üzemmóddal rendelkezik, az összes parancs kiadható mindkét üzemmódból, a különbség csak abban van, ahogyan azokra az értelmező reagál. A **startgame** parancs kiadása után a következő **endgame** parancsig a konfigurációs célú és inicializáló parancsok nem érvényesek.

A bemeneti nyelv nem kis-nagybetű érzékeny és nincs benne utasítás végét jelző speciális karakter (mint sok programozási nyelvben a ‘;’). Alapértelmezésben egy sor egy utasítás. Több sorból álló blokkok csak a kezdőállapot-definícióban vannak, külön definiált szintaktikával. A parancsok nevei mindig egybeírva, egy szóként írandók. Az opcionális paramétereket a kötelező paraméterek előtt (melyek <kötelező paraméter neve>-ként szerepelnek a specifikációban) kell megadni, mégpedig a következő módon:

parancs -opció érték -opció2 érték kötelező*_paraméter_*értéke

Mivel a külön szavak külön paraméternek számítanak, a sztring literálokat idézőjelek között szükséges megadni. **Az objektumok nevei nem sztring literálok, ezért csak egy szóból állhatnak!**

A következő parancsok értelmezettek:

setmode <mode>

Leírás: Beállítja a parancsértelmező üzemmódját. Lehet *shell* (alapértelmezés) vagy *test.* Konfigurációs célú utasítás.

Opciók: -

redirect <target>

Leírás: Átirányítja a kimenetet a **target** elérési úttal megadott fájlba. Ez addig érvényes, amíg felül nem írjuk. Ha újra a standard kimenetet szeretnénk használni(System.out), akkor a **target** értéke *stdout.* Konfigurációs célú utasítás.

Opciók: -

run <source>

Leírás: Lefuttatja a **source** elérési úttal megadott fájlt mint egy parancssori programot. Bármikor kiadható, nem áll vissza a kezdőállapotba.

Opciók: -

reset

Leírás: A program visszaáll a kezdőállapotba, minden előtte kiadott parancs érvénytelenné válik.
Opciók: -

exit

Leírás: Bezárja a programot.
Opciók:-

startgame

Leírás: Játék indítása, ha nem adunk opcióban presetet, akkor a játék állapota a korábban kiadott parancsokkal definiált kezdőállapot lesz, ha ilyen sincs akkor pedig az automatikusan generált kezdőállapot.
Opciók: ● **preset <target>:** a kezdőállapotot beolvashatjuk a **target** elérési úttal adott fájlból
● **players <number>:** megadhatjuk, hogy hány játékossal indítjuk el a játékot, ezután csak a játékosoknak a nevét kell felsorolni (a program ezt külön levezeti) és indul is a játék az **automatikus** presettel.

quitgame

Leírás: A játék megállítása. Ezután, amíg nem indítunk új játékot a **startgame-**el, a játék folytatható. Haszna shell módban az, hogy amíg a játék szünetel addig a játékszabályokat áthágva bármilyen parancsot adhatunk, amely módosítja a játék állapotát.
Opciók: ● **save <target>:** opcionálisan kérhetjük a játékállapot elmentését a **target** elérési úttal adott fájlba.

resumegame

Leírás: A játék folytatása **quitgame** után.
Opciók:-

endgame

Leírás: A játék lezárása és **befejezettnek** tekintése. Teszt módban lezárja a session-t és előállítja a kimeneti végállapotot.
Opciók:-

pickup <item>

Leírás: Felveszi a nevével megadott tárgyat, ha az éppen abban a szobában található ahol a játékos tartózkodik a karakterével. Csak a játékos körében használható.
Opciók:-

move <door>

Leírás: Átmegy a **door-**on keresztül a belőle nyíló szomszédos szobába, csak a játékszabályok szerint a játékos körében használható.
Opciók:-

activate <item>

Leírás: Aktiválja a nevével megadott tárgyat, ha az éppen a játékos inventoryjában található. Csak a játékos körében használható.

Opciók:-

drop <item>

Leírás: Eldobja a nevével megadott tárgyat, ha az éppen a játékos inventoryjában található. Csak a játékos körében használható.

Opciók:-

list <property>

Leírás: Ez is egy, a játékos számára hasznos parancs. Segítségével kilistázhatók az aktuális szobában található Item-ek (**items**), a játékosnál található Item-ek (**inventory),** a játékossal egy szobában tartózkodó entitások(**people)** és a szobából nyíló ajtók (**doors).**

Opciók:-

roomdetails

Leírás: Alapértelmezetten megadja annak a szobának az adatait (gázos-e, tisztaság, benne lévő itemek és entitások), ahol a játékos éppen tartózkodik. Opciók nélkül csak a játékos körében használható, opcióval pedig csak azon kívül. (ez a korlátozás csak shell módban érvényes)

Opciók: ● room: a nevével tetszőleges szobát is hivatkozhatunk. Csak a játékos körén kívül használható (shell módban).

itemdetails <item>

Leírás: Megadja az **item** adatait, mint a típus, az élettartam,eredtiség és aktiváltság.
Shell módban a játékos körében csak olyan item hivatkozható meg, mely a játékos inventoryjában található. Tranzisztor esetén mutatja a párját is és azt is, hogy az melyik szobában található.

Opciók:-

doordetails <door>

Leírás: Shell módban a játékos körében nem használható. Megadja a **door** adatait, mint azt, hogy melyik két szoba között található és milyen irány(ok)ban járható.

Opciók:-

init

Leírás: A játékállapot-definíciós nyelven írt állapot definíció kódba ágyazásának kezdete.

Opciók:-

endinit

Leírás: A játékállapot-definíciós nyelven írt állapot definíció kódba ágyazásának vége.

Opciók:-

act <actor> <action> <opcionális action paraméter>

Leírás: Egy entitás (**actor)** elvégez egy cselekményt **(action).** Az NPC karakterek cselekedtetése felülírja az automatikus NPC mechanizmust, gyakori használata csak

teszt üzemmódban ajánlott. A cselekmények lehetősége függ az actor típusától. Mindhárom entitástípus tud mozogni (move <door>), a Student és a Teacher tárgyat felvenni (pickup <item>), a Student pedig tárgyat használni (activate <item>) és tárgyat eldobni/lehelyezni is tud (drop <item>).
Opciók: Az **action**-nek van egy opcionális paramétere, ez azonban nem az opció szintakszissal adható meg, hanem mint egy szimpla harmadik paraméter. Ez a harmadik paraméter a cselekvés tárgya, ha ennek megadására szükség van a cselekvés értelmezéséhez.

merge <room1> <room2>
Leírás: A **room2** beleolvad a **room1-**be.
Opciók:-

divide <existing room> <new room name>
Leírás: A meglévő szoba (**existing room)** kettéosztódik, és létrejön egy új szoba melynek nevét a második paraméterben adjuk meg.
Opciók:-

status
Leírás: Kiírja a teljes játékállapotot. Shell módban a játékos körében nem kiadható.
Opciók: ● file: megadható egy kimenete fájl az elérési útjával.

[Ha szükséges, meg kell adni a konfigurációs (pl. pályaképet megadó) fájlok nyelvtanát is.]
A játékállapot megadásának szintakszisa:

A játékállapot leírása is különböző részekből épül fel, ezeknek a **sorrendje az ellenőrizhetőség és a deklarációk elsőbbségének szükséglete miatt fix.**

A részek egymástól elhatárolva szerepelnek, a következő módon:

rész neve
    rész kifejtése (indentálás nem szükséges)
end

A definíció részei megadásuk szükségességének sorrendjében:

1. **Szoba deklarációk (rooms)**: egy deklaráció egy sorban.
   <szoba név> < tulajdonságok egymástól vesszővel elválasztva>
   Meg kell adni a szoba tulajdonságait is, először a befogadóképességet (ezt kötelező), aztán vesszővel elválasztva a többi tulajdonságot, ha van (gassed, cursed).
2. **Ajtó deklarációk(doors):** itt szükséges leírni azt, hogy melyik ajtó melyik két szoba között található és milyen irányokban járható a következő módon. Ezzel egy ajtót deklarálunk, egy deklaráció egy sorban, a következő módon:
   door: room1 <> room2: ha kétirányú,
   door: room1 < room2: ha csak room2-ből room1 felé járható,
   door: room1 > room2: ha csak room1-ből room2 felé járható.

3. **Entitás deklarációk(entities):** egy deklaráció egy sorban.
   <entitás típus> <entitás név> <entitás szoba>
   Azt is kötelező megadni, hogy az entitás éppen melyik szobában tartózkodik!
4. **Tárgy deklarációk (items):** egy deklaráció egy sorban.
   <tárgy típusa> <tárgy neve> <tárgy szobája/tulajdonosa> <egyéb paraméterek>
   A harmadik paraméter lehet egyaránt szoba vagy Student típusú entitás, előbbi azt jelenti, hogy a tárgy az adott szobában található, utóbbi pedig azt, hogy az adott Student a tárgy tulajdonosa. Az egyéb paraméterek a tárgy tulajdonságait jelentik és megadásuk szintakszisa a következő: (példák)
   Transistor t1 room2 isActive: false pair: t2
   Beer b1 student1 lifetime: 2 isActive: true
   Minden tárgyhoz az adott tárgy számára lényeges paramétereket kell felsorolni! (isActive minden tárgyhoz, Transistorhoz pair is kell, Beer, TVSZ, Mask esetén lifetime)

7.1.3 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.]

A kimeneti nyelv nagyrészt a bemeneti nyelv definíciójára épít. **Ha a játékállapot leírás kimenetként áll elő, akkor garantált, hogy az egyes deklarációk a nevek szerint betűrendben szerepelnek. Ez a tesztelhetőség szempontjából egy fontos követelmény.**

Shell módban ha épp egy játékos köre következik, a játékos név szerint felszólításra kerül.

A parancsok által kiváltott válaszüzenetek a parancs sikerességétől függenek. Ha valahol kivétel keletkezett azt a program elkapja, 4-el kezdődő kódú hibaüzenet keletkezik és a parancsértelmező visszaáll a kezdeti állapotába (reset)!

A parancsok által kiváltott válaszüzenetek ismertetése:

- **200 Operation successful:** Shell módban a parancs sikerességét jelenti. **Parancsfüggően kiíródnak a megfelelő dolgok.**

  Ha a **drop** teleportálást váltott ki, arról értesítést látunk.

  A **list** parancs a listákat a következő módon írja ki:

  - lista elem 1
  - lista elem 2
  - ..
  …
  - lista elem n

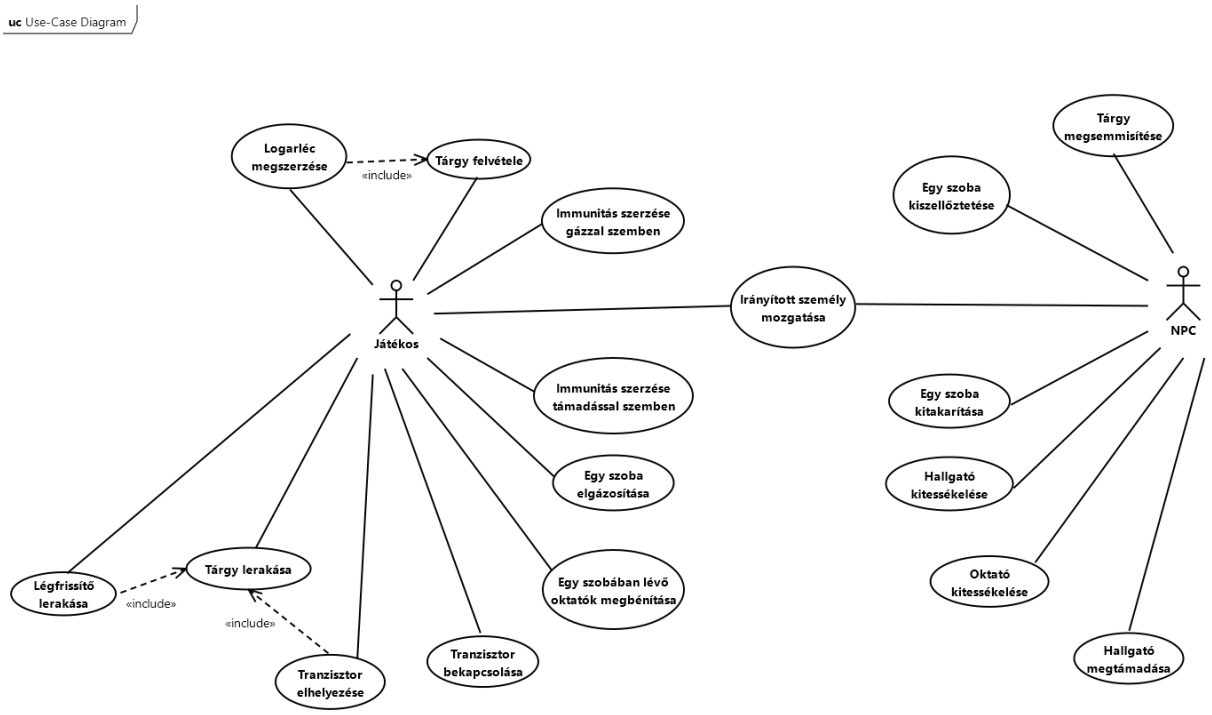
  A **doordetails, itemdetails, roomdeatils** parancsok esetén a kimenet a következő

formát ölti:

Objektum neve
Tulajdonság1: tulajdonság értéke/értékei
Tulajdonság2: tulajdonság értéke/értékei
….
TulajdonságN: tulajdonság értéke/értékei

- **350 Operation prohibited:** A parancs az adott programállapotban (pl: init block belseje, játékos köre) nem kiadható. Mindkét módban előáll.
- **351 Illegal operation:** A parancs nem végrehajtható, mivel áthágja a játékszabályokat. (pl: inventory kapacitáson túl tárgy felvétele) Mindkét módban előáll.
- **352 Nonexistent object:** A megadott névvel nem létezik élő objektum, ezért a parancs végrehajtása sikertelen. Mindkét módban előáll.
- **353 Nonexistent operation:** A megadott parancs nem létezik. Mindkét módban előáll.
- **354 Necessary parameter(s) missing:** Kötelező paraméter hiánya miatt a parancs végrehajtása sikertelen. Mindkét módban előáll.
- **400 File not found:** Kivétel keletkezett, mivel a megadott fájl nem található. Mindkét módban előáll.
- **401 Unexpected error:** Programhiba történt, ilyenkor a stack-trace megjelenik. Mindkét módban előáll.

7.2 Összes részletes use-case



Use-case neve	A Logarléc megszerzése
Rövid leírás	Győzelmi feltétel teljesülése a hallgatók számára, a játéknak ilyenkor vége.
Aktorok	Játékos
Forgatókönyv	A játékos felveszi a Logarlécet (pickup), mint egy tárgyat, ezáltal megnyerve a játékot a hallgatók csapata számára.

Use-case neve	Tárgy felvétele
Rövid leírás	Ha egy hallgató felvesz egy tárgyat, akkor azt képes az eszköztárában tárolni.
Aktorok	Játékos
Forgatókönyv	A hallgató olyan szobában tartózkodik, ahol egy vagy több tárgy található. Ilyenkor a játékos felveszi a felvenni kívánt tárgyat (pickup) és innentől kezdve azt a hallgatója eszköztárában tárolja. Ezt csak akkor tudja megtenni, ha az eszköztárban ehhez elég hely áll rendelkezésére, illetve a szoba padlója nem ragad.

Use-case neve	Tárgy megsemmisítése
Rövid leírás	Az oktatók, ha felvesznek egy tárgyat akkor az a tárgy kikerül a játékból.
Aktorok	NPC
Forgatókönyv	Az oktató egy olyan szobában tartózkodik, ahol egy vagy több tárgy található. Ilyenkor az oktató véletlenszerűen felveszi a tárgyakat (act NPC pickup).

Use-case neve	Irányított személy mozgatása
Rövid leírás	A játékos vagy az NPC tud mozogni, tehát ajtón keresztül átléphet egy szomszédos szobába.
Aktorok	Játékos, NPC
Forgatókönyv	A játékos vagy az NPC mechanizmus kiválasztja, hogy melyik ajtón keresztül szeretne átmenni és ezt a döntését véglegesíti (move / act NPC move).

Use-case neve	Immunitás szerzése gázzal szemben
Rövid leírás	A gázzal szembeni immunitást a nem elhasználódott FFP2-es maszk birtoklása jelenti, ilyenkor a játékos által irányított személy annyi körön át tartózkodhat elgázosított szobában mérgezés nélkül, amennyit a maszkja bír.
Aktorok	Játékos
Forgatókönyv	A játékos felveszi a maszkot az eszköztárába (pickup), onnantól kezdve a maszk a passzív tárgyhasználat szerint működik.

Use-case neve	Hallgató megtámadása
Rövid leírás	Az oktatók a velük egy szobában tartózkodó hallgatóknak “kiszívják a lelkét”, azaz megtámadják őket, ez védekezés hiányában a hallgató halálát jelenti, azaz az illető játékos számára ilyenkor a játék véget ért.
Aktorok	NPC

Forgatókönyv	A szobába ahol egy vagy több oktató tartózkodik belép egy vagy több hallgató. Ilyenkor az oktató(k) támadást indít/indítanak a hallgató(k) ellen. Ha ilyenkor egy hallgató nem tud védekezni a kör folyamán, akkor ő a kör végén életét veszti. Ha az utolsó hallgató is életét vesztette, akkor a játék az oktatók győzelmével zárul.

Use-case neve	Immunitás szerzése támadással szemben
Rövid leírás	Hallgató immunissá válhat az oktatók támadására aktív vagy passzív tárgyhasználat útján is.
Aktorok	Játékos
Forgatókönyv	Egy oktató támadást indít a hallgató ellen, ekkor a hallgatót megvédi egy nála lévő sör vagy TVSZ példány.

Use-case neve	Hallgató kitessékelése
Rövid leírás	A takarítók a velük egy szobában tartózkodó (nem bénult / ájult) hallgatókat kitessékelik a szobából, azaz átkerülnek az egyik szomszédos szobába.
Aktorok	NPC
Forgatókönyv	A szobába ahol egy vagy több hallgató tartózkodik belép egy vagy több takarító (act NPC move). Ilyenkor a takarítók kitessékelik a mozogni képes hallgatókat, akik átkerülnek az egyik szomszédos szobába.

Use-case neve	Oktató kitessékelése
Rövid leírás	A takarítók a velük egy szobában tartózkodó (nem bénult / ájult) oktatókat kitessékelik a szobából, azaz átkerülnek az egyik szomszédos szobába.
Aktorok	NPC
Forgatókönyv	A szobába ahol egy vagy több oktató tartózkodik belép egy vagy több takarító (act NPC move). Ilyenkor a takarítók kitessékelik a mozogni képes oktatókat, akik átkerülnek az egyik szomszédos szobába.

Use-case neve	Egy szoba elgázosítása

Rövid leírás	A játékosnak lehetősége van azon szoba elgázosítására ahol a hallgatója éppen tartózkodik. Ez azt jelenti, hogy innentől kezdve az adott szoba mérgezőnek, elgázosítottnak számít mindaddig, míg meg nem szüntetik az elgázosítást.
Aktorok	Játékos
Forgatókönyv	A játékos aktiválja a nála lévő dobozolt káposztás camembert-t (activate).

Use-case neve	Egy szoba kiszellőztetése
Rövid leírás	A takarító kiszellőzteti az elgázosított szobát, így kiszáll a gáz a szobából.
Aktorok	NPC
Forgatókönyv	Takarító belép egy elgázosított szobába (act NPC move), kiszellőztet, azaz megszűnik a szoba elgázosítása.

Use-case neve	Egy szoba kitakarítása
Rövid leírás	Ha egy takarító kitakarítja egy szobát, az a szoba takarítottá válik.
Aktorok	NPC
Forgatókönyv	Takarító belép egy szobába (act NPC move) és kitakarít, azaz a szoba takarított állapotba kerül.

Use-case neve	Egy szobában lévő oktatók megbénítása
Rövid leírás	A hallgató egy oktatóval egy szobába kerül és a birtokában van egy ilyen tárgy. Ha a hallgató aktiválja, akkor megmenekül a haláltól, és lebénítja az oktatókat.
Aktorok	Játékos
Forgatókönyv	A játékos hallgatója egy olyan szobában van, ahol vele együtt egy vagy több oktató is tartózkodik. A játékos a táblatörlő rongy aktiválásával (activate) a szobában tartózkodó oktatókat adott ideig megbéníthatja, ezalatt azok nem tudnak támadni és mozogni sem.

Use-case neve	Tárgy lerakása

Rövid leírás	A játékos dönthet úgy, hogy egy nála lévő tárgyat lerak abban a szobában, ahol éppen tartózkodik a hallgatójával.
Aktorok	Játékos
Forgatókönyv	A játékos lerakja a lerakni kívánt tárgyat az eszköztárából (drop). Ezután a tárgyat a szoba tartja nyilván.

Use-case neve	Légfrissítő lerakása
Rövid leírás	A játékos dönthet úgy, hogy egy nála lévő légfrissítőt lerak abban a szobában, ahol éppen tartózkodik a hallgatójával.
Aktorok	Játékos
Forgatókönyv	A játékos lerakja a nála lévő légfrissítő tárgyat (drop). Ekkor megszűnik a szoba elgázosítása, amennyiben ez korábban fennállt.

Use-case neve	Tranzisztor bekapcsolása
Rövid leírás	A játékos a nála lévő tranzisztorokat bekapcsolhatja. A tranzisztorok az első bekapcsoláskor kapcsolódnak össze. A tranzisztor bekapcsolása szükséges feltétele a teleportálásnak.
Aktorok	Játékos
Forgatókönyv	A játékos az eszköztárárában a tranzisztort kiválasztva aktiválja azt (activate). A tranzisztorok összekapcsolása az adott játékos általi bekapcsolások sorrenjében történik. Az első bekapcsolás után a párosítások megmaradnak, mindaddig amíg a pár mindkét tagja megvan.

Use-case neve	Tranzisztor elhelyezése
Rövid leírás	A tárgyak lerakásának speciális esete. Egy tranzisztorpár második bekapcsolt tagjának lerakása azonnali teleportálással jár abba a szobába, ahol a pár másik tagja található.
Aktorok	Játékos
Forgatókönyv	A játékos leteszi a nála lévő bekapcsolt tranzisztort (drop). Ha a pár második tagját helyezte le és mindkét tranzisztor be van kapcsolva akkor megtörténik a teleportálás.

7.3 Tesztelési terv

Teszt-eset neve	Tárgy felvétele hallgató által (söröspohár)
Rövid leírás	A hallgató felvesz egy tárgyat, ami bekerül az inventoryjába, és eltűnik a szobából. (söröspoharat használunk itt, a lefutás azonos a többi tárgynál, kivéve a logarlécnél)
Teszt célja	A Student osztály pickup(Item) funkcionalitását teszteli.

Teszt-eset neve	Tárgy felvétele oktató által (söröspohár)
Rövid leírás	Az oktató, ha felvesz egy tárgyat megsemmisíti. (söröspoharat használunk itt, a lefutás azonos a többi tárgynál, kivéve a logarlécnél)
Teszt célja	A Teacher osztály pickup(Item) és az Item osztály destroy() funkcionalitásait teszteli.

Teszt-eset neve	Oktató megöl egy hallgatót
Rövid leírás	Az oktatók a velük egy szobában tartózkodó hallgatóknak “kiszívják a lelkét”, azaz megtámadják őket, ez védekezés hiányában a hallgató halálát jelenti, azaz az illető játékos számára ilyenkor a játék véget ért.
Teszt célja	A Student osztály teacherAttack() funkcionalitását teszteli, abban az esetben, ha a megtámadott Student protectiveItems tömbje üres, valamint a Room osztály removePerson() függvényét is.

Teszt-eset neve	Oktató megtámad oktatót
Rövid leírás	Az oktatók a velük egy szobában tartózkodó oktatókat is megtámadják, de ez rájuk nincs hatással.
Teszt célja	A Teacher osztály initAttack() és teacherAttack() funkcionalitásait teszteli.

Teszt-eset neve	TVSZ használata
Rövid leírás	Egy oktató megtámad egy Hallgatót, mivel egy szobában vannak, de mivel a Hallgató birtokában van egy TVSZ, az megvédi őt az Oktatóval szemben.
Teszt célja	ATeacher osztály initAttack(), a Student osztály teacherAttack() funkcionalitásait teszteli, abban az esetben, amikor a Student protectiveItems tömbjében egy TVSZ található. Továbbá a TVSZ osztály teacherThreat(), acceptProtection(), valamint a Student osztály teacherProtection() függvényei is tesztelésre kerülnek.

Teszt-eset neve	Gázos szoba hatása oktatóra

Rövid leírás	Egy Oktatót megtámad egy gázos szoba. Az Oktató elgázosodik.
Teszt célja	A Teacher osztály gasStun() funkcionalitását teszteli.

Teszt-eset neve	Rongy használata
Rövid leírás	Egy Hallgató aktiválja a nála levő rongyot, ami megbénítja a vele egy szobában levő Oktatókat.
Teszt célja	A Student osztály initActivate(), a Cloth osztály activate(), valamint a Student és Teacher osztályok clothStun() funkcionalitásait teszteli.

Teszt-eset neve	Camembert használata
Rövid leírás	Egy Hallgató egy Oktatóval van egy szobában és felnyitja a Camembert-t, mindketten lebénulnak.
Teszt célja	A Student osztály initActivate(), a Camembert osztály activate(), valamint a Room osztály gas() és Student, Teacher osztályok gasStun() funkcionalitásait teszteli.

Teszt-eset neve	Gázos szoba hatása hallgatóra
Rövid leírás	Egy Hallgató elgázosított szobába lép, elejti tárgyait és lebénul.
Teszt célja	A Student osztály gasStun() fukncionalitását teszteli, abban az esetben, ha nincs a Student-nél védelem, továbbá emiatt a Person osztály dropAll() függvénye is tesztelésre kerül.

Teszt-eset neve	Mask használata
Rövid leírás	Egy Hallgató elgázosított szobába lép, de a nála lévő maszk megvédi a lebénulástól.
Teszt célja	A Student osztály gasStun() funkcionalitását teszteli, abban az esetben, amikor a Student-nél van Mask. Továbbá a Mask osztály gasThreat(), acceptProtection(), valamint a Student osztály gasProtection() függvényei is tesztelésre kerülnek.

Teszt-eset neve	Beer és TVSZ használata (Hallgatót megtámad egy Oktató, de TVSZ és söröspohár is van nála)
Rövid leírás	Egy oktató megtámad egy Hallgatót, mivel egy szobában vannak. A Hallgató birtokában van egy söröspohár és egy TVSZ is. Ha a söröspohár nincs aktiválva akkor a TVSZ, ha aktiválva van akkor pedig a söröspohár fog védelmi nyújtani mivel utóbbinak mindig nagyobb a prioritása.
Teszt célja	A Teacher osztály initActivate(), a Student osztály teacherAttack() funkcionalitását teszteli, abban az esetben, amikor a Student protectiveItems tömbjében egy Beer és egy TVSZ található. Ha a Beer aktiválva van, akkor teszteli a Beer osztály teacherThreat(), acceptProtection() függvényeit, továbbá a Student

	osztály teacherProtection és dropRandomItem() függvényeit. Ha nincsen, akkor pedig a TVSZ osztály teacherThreat(), acceptProtection(), valamint a Student osztály teacherProtection() függvényei kerülnek tesztelésre.

Teszt-eset neve	Szoba osztódás
Rövid leírás	A szobák körönként véletlenszerűen “dönthetnek” úgy, hogy osztódnak. Az osztódó szoba két olyan szobára válik szét, amelyek egymás szomszédai lesznek, és megosztoznak a korábbi szoba tulajdonságain és szomszédain. Az osztódás után létrejövő két szoba között kétirányú ajtó jön létre. A két “új” szoba befogadóképessége az eredeti szobának a befogadóképességével fog megegyezni.
Teszt célja	A Room osztály divide() funkcionalitását teszteli.

Teszt-eset neve	Szoba összeolvadás
Rövid leírás	A szobák körönként véletlenszerűen “dönthetnek” úgy, hogy egyesülnek egy szomszédukkal. Két szoba egyesülésével létrejövő új szoba a korábbi két szoba tulajdonságaival, szomszédaival, tárgyaival és entitásaival rendelkezik, de a befogadóképessége a nagyobb szoba befogadóképességével lesz azonos. Két szoba csak akkor egyesülhet, ha a két szobában lévő entitások összege nem haladja meg a nagyobb szoba befogadóképességét.
Teszt célja	A Room osztály merge() funkcionalitását teszteli.

Teszt-eset neve	Logarléc hallgatóval
Rövid leírás	Egy hallgató felvesz egy logarlécet egy szobában. Ezzel meg is nyeri a játékot
Teszt célja	A Student osztály slideRuleNotification() funkcionalitását teszteli.

Teszt-eset neve	Logarléc oktatóval
Rövid leírás	Egy oktató felvesz egy logarlécet a szobában. Rögtön el is dobja ennek hatására
Teszt célja	A Teacher osztály slideRuleNotification() funkcionalitását teszteli.

Teszt-eset neve	Tranzisztor használata
Rövid leírás	Egy hallgató egy szobában van és van nála két tranzisztor. Ezeket aktiválja majd lerakja az egyiket. Ezután átmegy egy másik szobába, ahol lerakja a másikat és ezzel elteleportál abba szobába ahonnan jött. Deaktiválódnak a tranzisztorok

Teszt célja	A Student osztály initActivate(), teleport(),a Transistor osztály activate(), getDestination(), setPair(), setRoom() funkcionalitásait teszteli.

Teszt-eset neve	Tranzisztor felvétele oktató által
Rövid leírás	Egy oktató felvesz egy lerakott tranzisztort, ami össze van kapcsolva egy másikkal. Megsemmisíti azt, amelyiket felvett és a párja pedig resetelődik.
Teszt célja	A Transistor osztály getDestination(), reset(), deactivate(), destroy() funkcionalitásait teszteli.

Teszt-eset neve	Légfrissítő használata
Rövid leírás	Egy hallgató gázos szobába lép, a Légfrissítő aktiválásával megszűnik a szoba gázossága. A légfrissítő egy használat után megsemmisül.
Teszt célja	A Student osztály initActivate(), az AirFreshener osztály activate() és a Room osztály ungas() funkcionalitásait teszteli.

Teszt-eset neve	Takarító kitessékel embereket és kiszellőztet
Rövid leírás	Egy takarító olyan szobába lép amiben személyek vannak, akiket “kitessékel”, azaz szomszédos szobákba fog kerülni minden mozogni képes (nem bénult / ájult / van hova mozognia) személy rajta kívül.
Teszt célja	A Cleaner osztály initClean(), továbbá a Room osztály evacuate(), unGas() és clean() funkcionalitásait teszteli.

7.4 Tesztelést támogató segéd- és fordítóprogramok specifikálása

Saját parancssorban futtatható programot írunk a tesztesetek validálására. A tesztprogram futásának elején meg kell adnunk a teszteseteket tartalmazó mappa elérési útján, aztán pedig azt, hogy debug módban futtatjuk-e a teszteket. Ha igen, a konzol kimeneten megjelenik a skeletonból már ismert hívási lánc is, a tesztek validációja azonban továbbra is az állapot alapján történik. Egy tesztesetet az ezeket tartalmazó mappában két fájllal tudunk megadni: egy testN.txt-vel és egy outputN.txt-vel. Az N a teszteset száma 0-tól indexelve, a testN.txt a bemenetet adja meg (inicializáló állapot és végrehajtandó parancsok egyaránt), az outputN.txt pedig az elvárt végállapot definícióját tartalmazza, ez utóbbi alapján történik a validáció. A tesztprogram visszajelzést küld a tesztek sikerességéről. Ha egy teszt sikertelen volt akkor a tesztprogram megjeleníti a tényleges kimenet esetleges hibaüzeneteit és/vagy az elvárt kimenetettel nem egyező első sor után következő sorokat. Az elvárt kimenet tartalmazhat hibaüzenetet is, ez a 3-al kezdődő hibaüzenetek esetén ajánlott mert ezek előfordulhatnak egy test-case helyes futása alatt is! Ezeket az outputN.txt-ben a végállapot definíciója elé kell írni.

Napló

Kezdet	Időtartam	Résztvevők	Leírás
2024.04.03. 16:00	1 óra	Cardinael Görömbey Király Szakos	Feladatok kiosztása
2024.04.04. 11:00	1 óra	Riba	Use-case-ek leírása
2024.04.05. 11:00	1 óra	Cardinael	Szekvenciák
2024.04.05. 10:00	2 óra	Görömbey	Tesztesetek megírása.
2024.04.05. 12:00	1 óra	Cardinael Görömbey Király Riba	Egyeztetés.
2024.04.07. 22:00	1 óra	Riba	Use-case diagram és Use-case leírások véglegesítése
2024.04.07. 21:00	3 óra	Szakos	A bementi és kimeneti nyelvek specifikálása.