**03. Analízis modell kidolgozása 1.**

12 - ch33kybr33ky\_labz

Konzulens:

Bodó Zsófia

Csapattagok

|  |  |  |
| --- | --- | --- |
| **Frontó András** | **G392CQ** | andrasfronto@gmail.com |
| **Czetter Gábor** | **CP6OG3** | cz3tt1@gmail.com |
| **Tomori Dániel** | **TDWSF5** | tomori.daniel@gmail.com |
| **Verasztó Bálint** | **VSDRBZ** | balintveraszto@gmail.com |

1. **Analízis modell kidolgozása**
   1. ***Objektum katalógus***
      1. **Player**

A Player-eket a felhasználók irányítják a pálya falakkal határolt mezőin. Képesek Box-okat tolni, de nem tudnak átmenni a Wall vagy Pillar Field-eken. A Button-ok nem aktiválódnak, ha rájuk lépnek viszont a TrapDoor-ok és a Hole-ok képesek megölni őket. Továbbá lépés hiányába, ha rájuk tolódik egy doboz akkor meghal.

* + 1. **Box**

A Box-okat a Playerek tologatják a Map-en. Nem lehet Wall-ba vagy Pillar-ba tolni őket, de cserébe képesek aktiválni a Button-al ellátott Field-eket. Ha Hole-al vagy TrapDoor-al rendelkező Field-re lépnek akkor meghalnak. Ha a Target mezőbe csúszik be egy Box, az pontot ad annak a Player-nek, aki utoljára hozzáért, majd eltűnik.

* + 1. **Map**

A játékban lévő összes Field-et összefoglalja, továbbá felelős azok elhelyezkedéséért és specialitásuk kiosztásáért.

* + 1. **Field**

A játék mezői, amik együtt a Map-ot alkotják. Field-eknek vannak speciális esetük, amik mind máshogy reagálnak arra, ha egy Player vagy egy Box kerül rájuk.

* + 1. **Wall**

A Filed speciális esete, ami határolja a Map mozgásterét. Se Player se Box nem képes átmenni rajta.

* + 1. **Pillar**

A Wall egy speciális esete, ami a Map azon részén helyezkedik el, ahol a Player-ek és a Box-ok mozognak. Lényegében ez egy egyedül álló falrész.

* + 1. **Hole**

A Field egy speciális esete, ha Player vagy Box kerül rá eltűnik.

* + 1. **Button**

A Field egy speciális esete. Csak akkor tud benyomódni, aktiválódni, ha egy Box kerül fölé. Ekkor a hozzá tartozó TrapDoor, valahol a Map-on kinyílik és ami rajta van eltűnik.

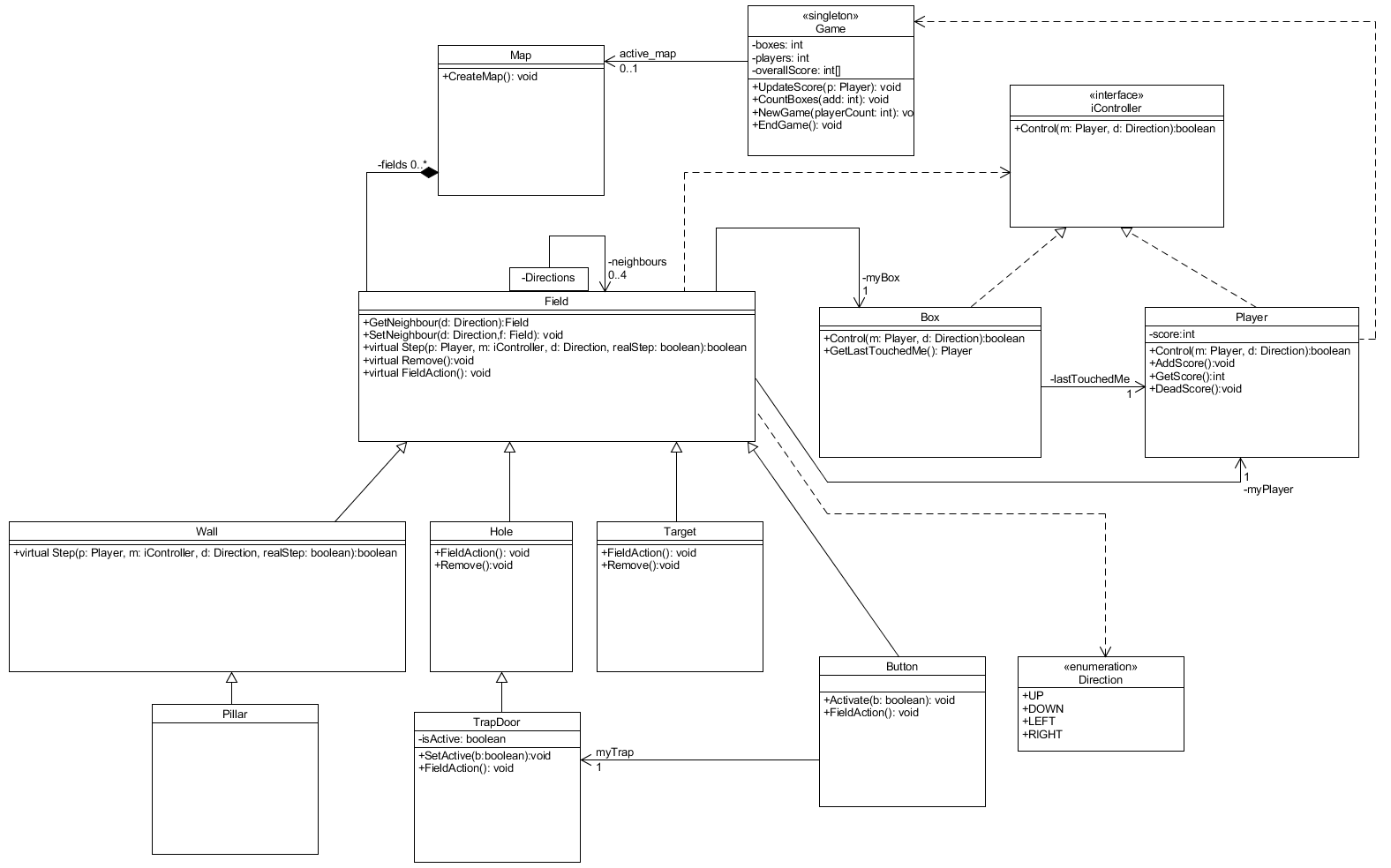
* + 1. **TrapDoor**

Ha a hozzá tartozó Button aktív, kinyílik és a rajta lévő Box, vagy Player beleesik. Ha a Button-ról eltűnik a Box, bezárul és újra egy egyszerű Field lesz.

* + 1. **Target**

Cél mező a Field egy speciális esete, ha Box kerül rá eltűnik és pontot kap a végső lökést adó játékos. Ha játékos lép rá, nem történik semmi.

* 1. ***Statikus struktúra diagramok***

**

* 1. ***Osztályok leírása***

Azért hogy leegyszerűsítsük az osztály leírásokat azokat a virtuális függvényeket amiket a leszármazott nem ír felül nem írtuk ki még egyszer (pl.: Button.Step,Button.Remove stb).

* + 1. **Box**
       - **Felelősség**

Az iController-t megvalósító osztály. A Player-ek Box-okat tolnak a Field-eken keresztül. Box-ok képesek gombokat benyomni, és a Target-be lökésükért pont jár a végős lökést adó Player-nek.

* + - * **Ősosztályok**

Nincs

* + - * **Interfészek**

iController

* + - * **Attribútumok**
* **Player lastTouchedMe**
  + - * **Metódusok**
* **Player GetLastTouchedMe:**.Visszaadja azt a player-t aki legutoljára ért hozzá.
* **boolean Controll(Player p, Direction d):** Mozdulási szándékot intéz egy Field felé, és visszaadja, hogy sikerült-e a mozdulás. Első paramétere megmondja, hogy ki tolta, a második pedig, hogy milyen irányba.
  + 1. **Button**
       - **Felelősség**

A Field-ből öröklődik. Felelős azért, hogy ha egy Box kerül rá, ami képes benyomni, akkor a hozzá tartozó TrapDoor kinyíljon, továbbá ha az lekerül róla, bezáródjon. Ha egy Player lép rá, nem történik semmi, a csapó ajtó csukva marad.

* + - * **Ősosztályok**

Field → Button

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **TrapDoor myTrap**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **void Activate(Boolean b)**: Box hatására aktiválja a gombot, Player hatására nem történik semmi.
* **void FieldAction():** A gomb aktiválódását segíti elő. Figyeli, hogy beszorul-e a láda.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.
  + 1. **Field**
       - **Felelősség**

Ezekből épül fel a Map. Egy Field mező felelős a rajta lévő Player-ért vagy Box-ért. Ha az a külső hatás miatt el szeretne mozdulni a Field felelős a lépéséért vagy az esetleges haláláért. A szomszédos mezőket is ő tartja számon.

* + - * **Ősosztályok**

Nincs

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.
* **virtual boolean Step(Player p, m iController, Direction d, boolean realStep)**: Tovább lépést indít egy megadott irányba. Visszatérési értéke azt jelzi, hogy a lépést sikeresen megtörtént-e vagy valami miatt meghiúsult. Az első paramétere azt mutatja, hogy ki tolta az objektumot, ennek később lesz majd hatása majd, amikor több Box-ot tolunk egyszerre. A második az a Box vagy Player aki a mezőre szeretne lépni. A harmadik pedig az irány, amerre az egész megindul. Az utolsó paraméter pedig ha igaz akkor tényleg lép,  ha hamis akkor csak egy ellenőrzés hogy léphetne e. Ez a beszorulás vizsgálatához kell.
* **virtual void Remove():** Az egyszerű Field Remove() függvénye csak és kizárólag Player-t tud eltűntetni, mivel csak az tud meghalni attól, ha rátolnak egy Box-ot. Box-okat nem lehet összetolni.
* **virtual void FieldAction():** Minden Field-hez az egyedi tulajdonságot hajtja végre ami kötelezően meghívódik, ha valaki rálép vagy rátolódik. Figyeli, hogy beszorul e a láda.
  + 1. **Direction**

Egy enumeráció, ami a lehetséges irányokat tartalmazza. (UP, DOWN, LEFT, RIGHT)

* + 1. **Game**
       - **Felelősség**

Egy singleton osztály. Feladata a játék elkezdése és lezárása, továbbá ezeknek a feltételeknek az ellenőrzése. Folyamatosan számon tartja hány Boksz mozgatható, így ha ez a szám nullára csökken a játék automatikusan véget ér, valamint a Player-ek pontjait és számát is ő tartja nyilván.

* + - * **Ősosztályok**

Nincs

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **int boxes**
* **int players**
* **int[] overallScore**
  + - * **Metódusok**
* **void UpdateScore(Player p):** Frissíti az adott Player pontszámát. Vagy növeli, vagy nulláza abban az esetben ha meghalt. Ha kiesik 3 játékos befejezi a játékot.
* **void CountBoxes(int add):** Számlálja a mozgatható Box-okat a Map-en. Amikor elfogynak befejezi a játékot.
* **void NewGame(int playerCount):** Új játékot indít a megadott számú játékossal.
* **void EndGame():** Véget ér a játék, ha vagy mozgatható Box, vagy 1-nél több irányítható Player nincs már a Map-en.
  + 1. **Hole**
       - **Felelősség**

A Field-ből öröklődik, ha bármi rákerül, eltűnik és törlődik a játékból. Ha Box kerül bele nem jár érte senkinek pont, ha Player, meghal és elveszíti az eddig összegyűjtött pontjait.

* + - * **Ősosztályok**

Field → Hole

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **void FieldAction():** Ebben az esetben, bármi ami rákerül, leesik, tehát törlődik a játékból.
* **void Remove():** A Hole Remove() függvénye már nem csak Player-t, de Box-ot is képes eltűntetni, hiszen tök mindegy mi lépett rá, az rögtön törlődik a játékból.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.
  + 1. **iController**
       - **Felelősség** Egy interface, amit a Player és a Box osztály valósít meg. Felelős az őt megvalósító osztályok lépéseiért.
       - **Ősosztályok**

Nincs

* + - * **Metódusok**
* **boolean Control(Player p, Direction d)**: Mozdulási szándékot intéz egy Field felé, és visszaadja, hogy sikerült-e a mozdulás. Első paramétere megmondja, hogy ki tolta, a második pedig, hogy milyen irányba. Dobozra hívva a beállítja a lastTouched paramétert is.
  + 1. **Map**
       - **Felelősség**

A pálya szabályos kialakításáért felelős, ezzel együtt raktározza a Field-eket.

* + - * **Ősosztályok**

Nincs

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **Field[] fields**
* **Game active\_map**
  + - * **Metódusok**
* **void CreateMap()**: Megadott algoritmusok szerint alkotja meg random a pályát az egyszerű Field-ek és a speciális Field-ek használatával. Lerakja a Boxokat és a Playereket is.
  + 1. **Pillar**
       - **Felelősség**

A Pillar-ok ugyan úgy viselkednek, mint a falak attól eltekintve, hogy a Map közepén helyezkednek el. Nem lehet rajtuk áthaladni, vagy Box-okat áttolni.

* + - * **Ősosztályok**

Field → Wall → Pillar

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **virtual boolean Step(Player p, m iController, Direction d, boolean realStep)**: Ebben az esetben a Step függvény nem engedi a lépést és rögtön elutasítja, ha valaki rá szeretne kerülni bármelyik irányból is érkezik a kérés a lépésre.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.
  + 1. **Player**
       - **Felelősség**

Az iController megvalósító osztály. Felelős a Playerek mozgatásáért és a pontjaiért.

* + - * **Ősosztályok**

Nincs

* + - * **Interfészek**

iController

* + - * **Attribútumok**
* **int score**
  + - * **Metódusok**
* **void AddScore():** Egyet hozzáad a Player pontjaihoz.
* **int GetScore():** Lekérdezi az adott Player pontját.
* **boolean Controll(Player m, Direction d):** Mozdulási szándékot intéz egy Field felé, és visszaadja, hogy sikerült-e a mozdulás. Első paramétere megmondja, hogy ki tolta, a második pedig, hogy milyen irányba.
* **void DeadScore():** -1-re állítja az adott Player pontjait, majd a Game UpdateScrore() függvénye kiveteti a játékból.
  + 1. **Target**
       - **Felelősség**

A Fieldből származó osztály. Csak a Box-okra aktiválódik a Player-eket figyelmen kívül hagyja. Ha egy Box rákerül az eltűnik és az utolsó lökést adó játékosnak pontot ad. Természetesen a Target mezőn is ugyan úgy összenyomódhat a Player mint bárhol máshol.

* + - * **Ősosztályok**

Field → Target

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **Directions[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **void FieldAction():** Rátoláskor a Box eltűnik és az utolsó Playernek aki hozzá ért pont írást kezdeményez.
* **void Remove():** A szokásos Remove() függvény. Ha meghívődik törli a Playert vagy a Boxot a Field-ről.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.

**3.3.10 TrapDoor**

* + - * **Felelősség**

A Field leszármazottja. Egy hozzá tartozó gomb utasítására nyílik ki és viselkedik úgy, mint egy Hole, vagy csukódik be és egy sima Field-ként létezik.

* + - * **Ősosztályok**

Field → Hole → TrapDoor

* + - * **Interfészek**

Nincs

* + - * **Attribútumok**
* **boolean isActive**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **void SetActive(boolean: b):** Átállítja a TrapDoor állását. Nyitottból csukott, csukottból nyitott.
* **void FieldAction():** Aktiválódáskor kinyílik, és ha éppen egy Player vagy egy Box volt rajta, akkor azt törli. Ha a hozzá tartozó Button-ról letolják a Box-ot akkor bezárul és sima Field lesz amire midnen következmény nélkül rá lehet lépni vagy tolni.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.

**3.3.11 Wall**

* + - * **Felelősség**

A Field-ből öröklődik. Sem a Player-ek sem a Box-ok nem képesek áthaladni rajta. Áthatolhatatlanok a játék kezdetétől a végéig és ezt már senki nem veheti eltőlük.

* + - * **Ősosztályok**

Field → Wall

* + - * **Interfészek**

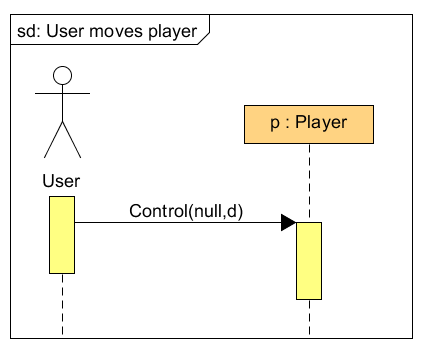
Nincs

* + - * **Attribútumok**
* **Field[] neighbours**
* **Box myBox**
* **Player myPlayer**
  + - * **Metódusok**
* **virtual boolean Step(Player p, m iController, Direction d)**: Ebben az esetben a Step függvény nem engedi a lépést és rögtön elutasítja, ha valaki rá szeretne kerülni.
* **Field GetNeighbour(Direction d)**: A megadott irányba lévő Field-et adja vissza.
* **void SetNeighbour(Direction d, Field f):** Beállítja az adott irányba a paraméterül kapott Field-et.

***Szekvencia diagramok***

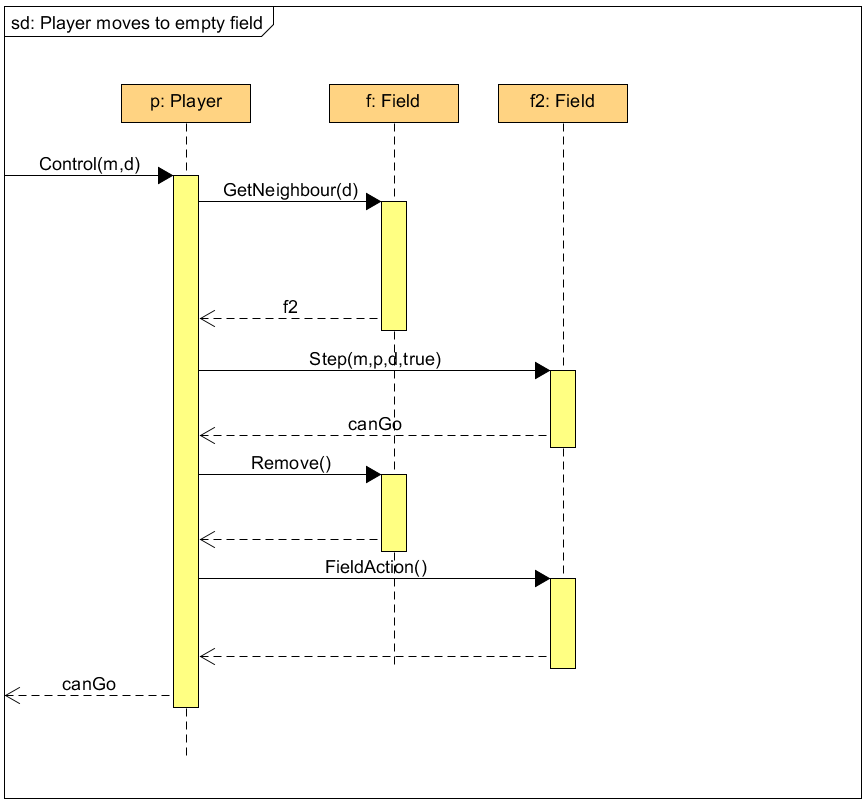
***3.4.1 User moves player***

A felhasználó mozgatja az egyik játékost. Fontos, hogy itt nem egy másik térképen lévő objektum “löki” meg a játékost - ezért a Controll első átvett értéke *null*.

******

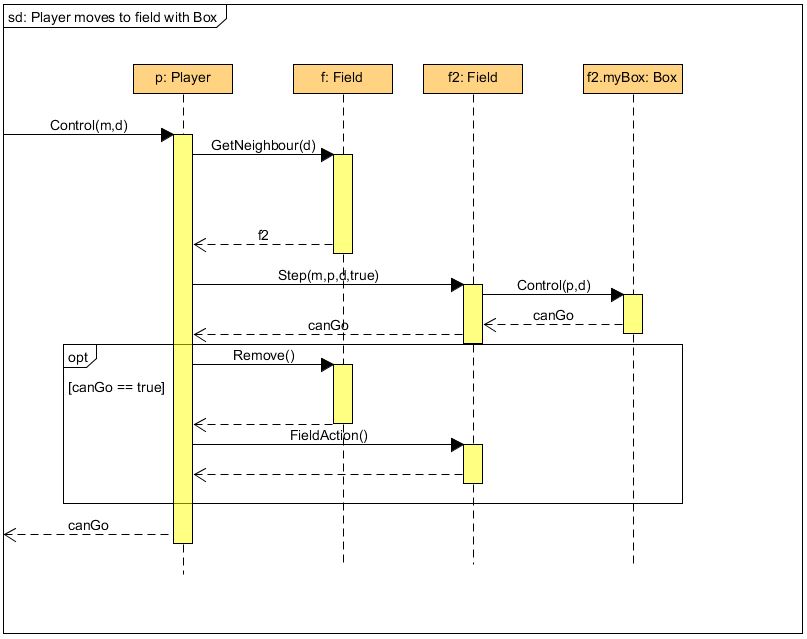
***3.4.2 Player moves to empty field***

A játékost valami (akár a user akár pl egy doboz) elmozgatja egy irányba, ahol üres mező taláható. Az ábrán nincs lekezelve az az eset, amikor van valami a szomszéd mezőn - ezt külön diagramon kezeljük. Az m értéke vagy a mozgást kezdeményező Player vagy null ha a játékos kezdte.

******

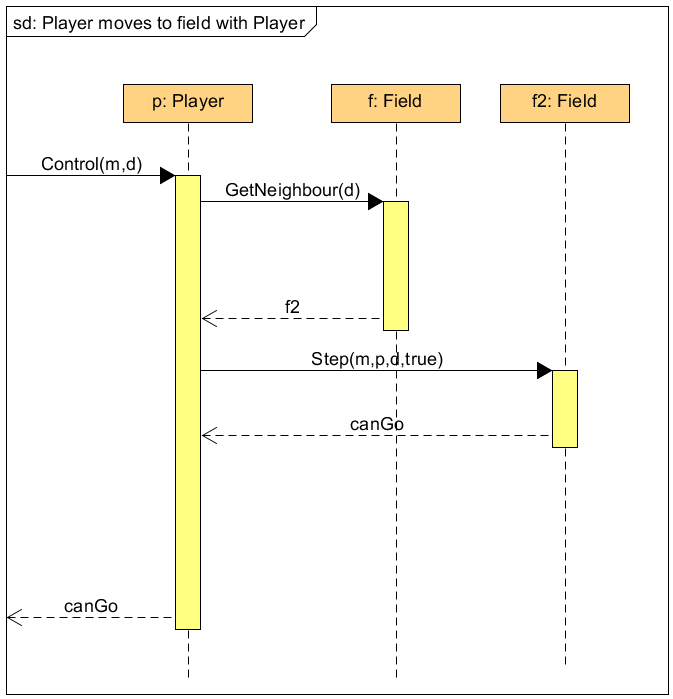
***3.4.3 Player moves to field with box***

A játékos olyan mezőre lép, amin van egy doboz. Ha a játékost a felhasználó mozgatta, akkor a Control első paramétere null, ha valamilyen másik objektum, akkor a mozgást kezdeményező Player. Ha a láda által visszaadott canGo érték igaz, akkor a játékos eltolhatja a dobozt és a lépés végbemegy. A doboznak saját magát adja a Controll-ban, így a dobozok mindig arról a Player-ről tudnak aki “utoljára ért bele” a mozgásba.

******

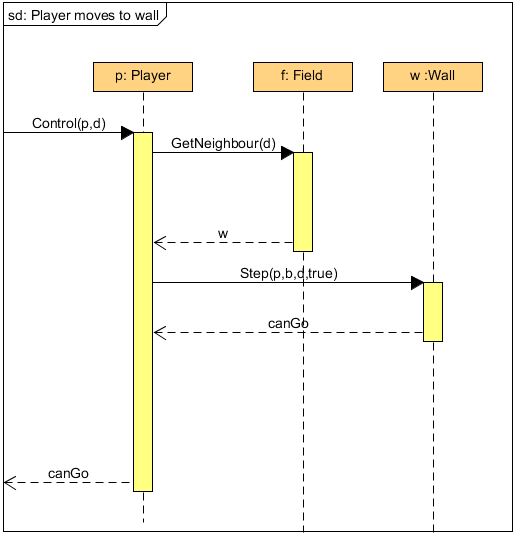
***3.4.4 Player moves to field with player***

A játékost a felhasználó vagy egy másik objektum egy olyan mezőre mozgatná, amin egy másik játékos áll. (Itt is a Controll függvény első paramétere ettől függően vagy null vagy a “megtoló” Player.)

******

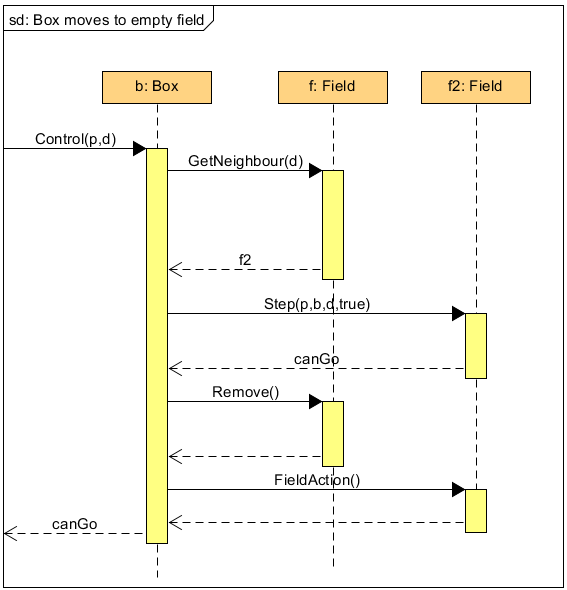
***3.4.5 Player moves to wall***

A Player egy falra akar lépni amit nem tehet, itt a Palyer lép Player-re esethez hasonlóannem fog oda lépni. (Itt is a Controll függvény első paramétere ettől függően vagy null vagy a “megtoló” Player.)



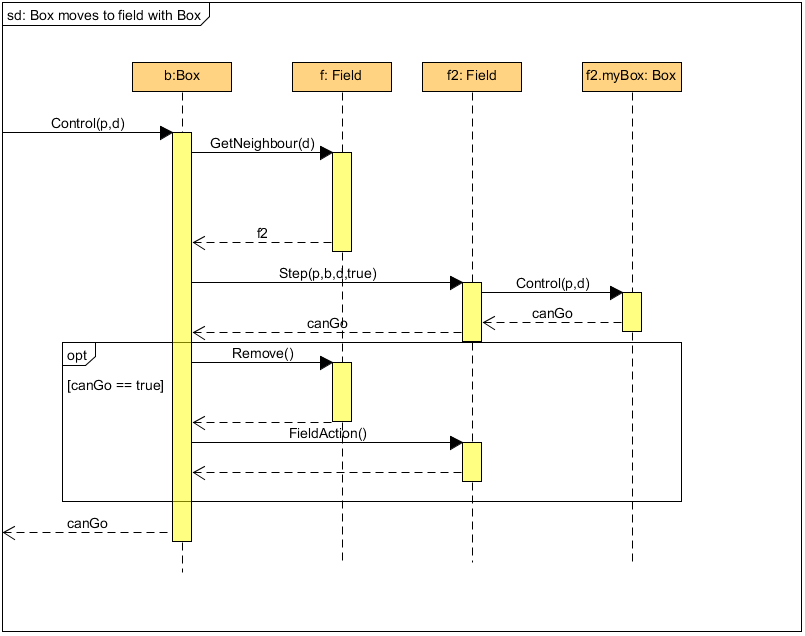
***3.4.6 Box moves to empty field***

Egy dobozt egy szomszédos üres mezőre tolják. Ebben a diagramban csak azt az esetet kezeljük, amikor a szomszéd mező üres.

******

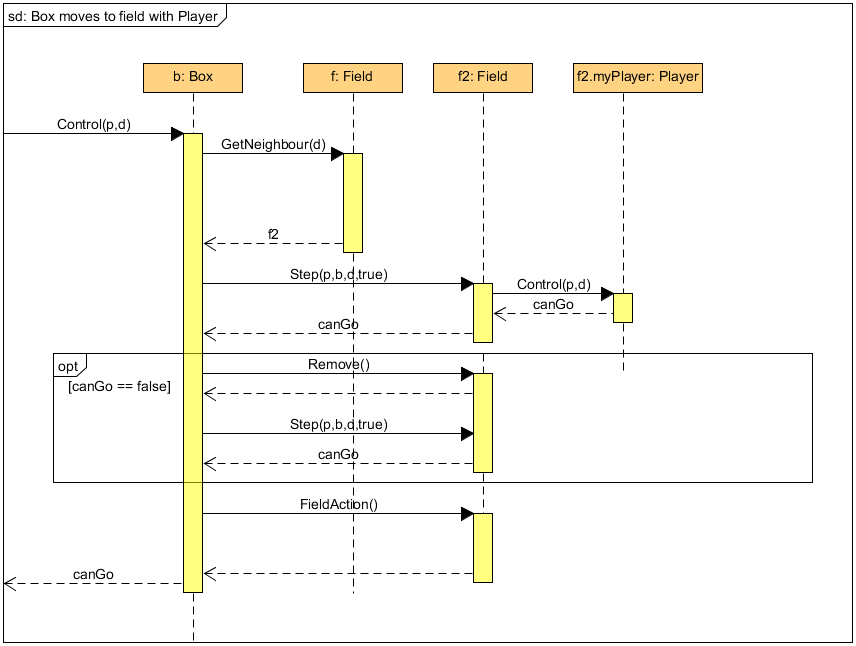
***3.4.7 Box moves to field with box***

A dobozt egy olyan mezőre tolják, amin szintén egy doboz van. Rekurzíve megnézi, hogy tud-e lépni az adott irányba. Ha igen, akkor lép.

******

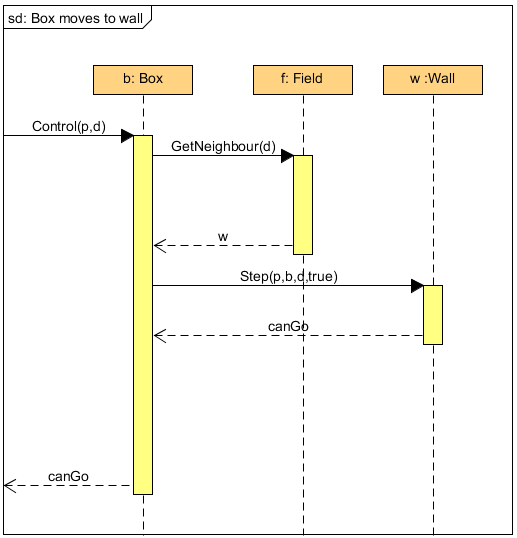
***3.4.8 Box moves to field with player***

A dobozt egy olyan mezőre tolják, ahol egy másik játékos van. Ha nem tud menni a játékostól “összenyomja” azt.

******

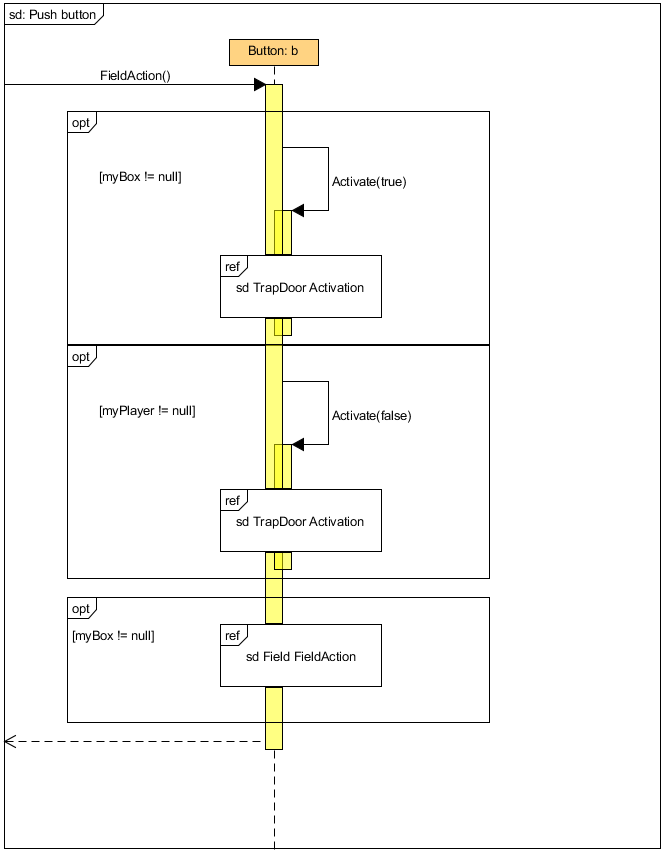
***3.4.9 Boksz moves to wall***

A doboz egy falra akar lépni amit nem tehet tehát itt a canGo false-al jön vissza és nem lép oda.



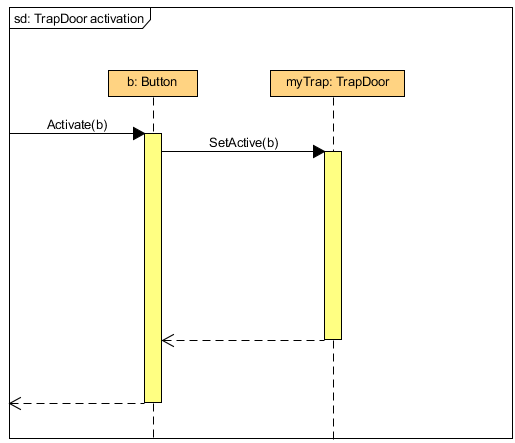
***3.4.10 Box pushes button***

A doboz egy olyan mezőre kerül, amin egy gomb van. Meghívódik a Button Activate függvénye a rajta lévő obijektumnak megfelelő boolean értékkel. (a Box megnyomja a Player nem)



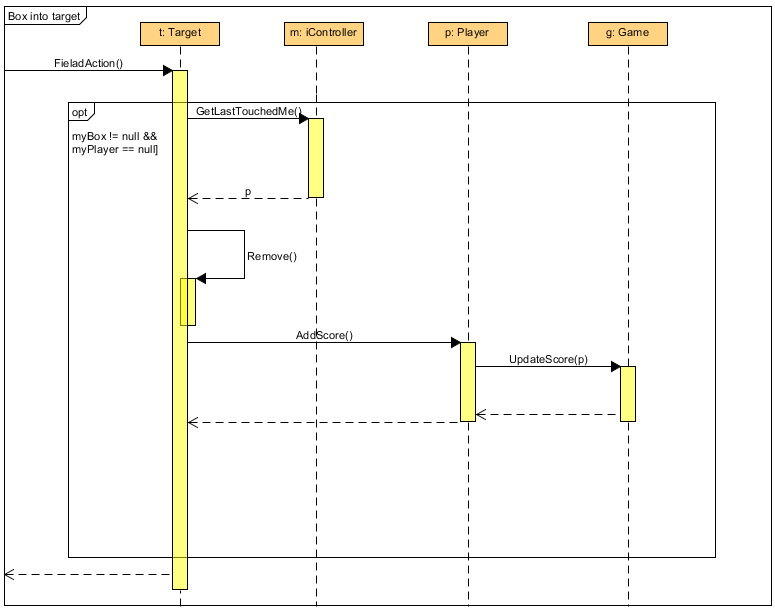
***3.4.11 TrapDoor activation***

Ha egy gombra rátolnak egy dobozt, az aktivál egy TrapDoort. Beállítja a gombon az aktivitást, ami meghívja az általa tartalmazott TrapDoor aktiváló függvényét, ami ez után kinyílik és Hole-ként viselkedik.

******

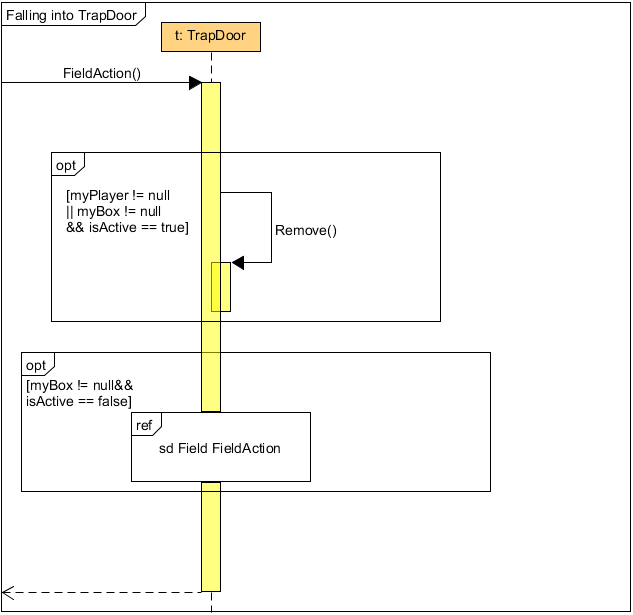
***3.4.12 Move box into target***

Egy doboz célba tolása. Megnézi az adott Field, hogy Box került-e rá. Ha igen, akkor lekérdezi, hogy ki volt az a player aki az adott mozgást indította. Ezután a Target Fieldről eltávolítja a Boxot, és az adott Playernek jóváírja a pontot.

******

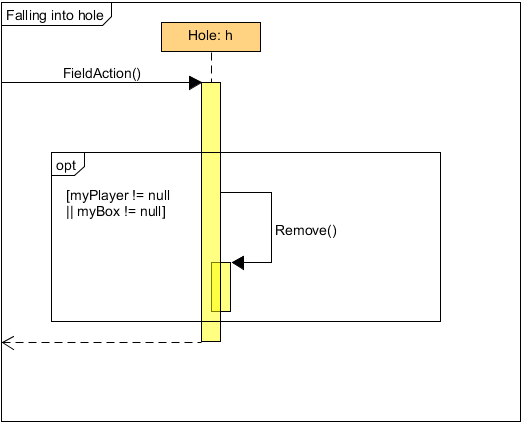
***3.4.13 Falling into TrapDoor***

Vagy egy Player vagy egy Box lép egy nyitott TrapDoor-ra. Ekkor az adott objektum leesik és eltűnik.

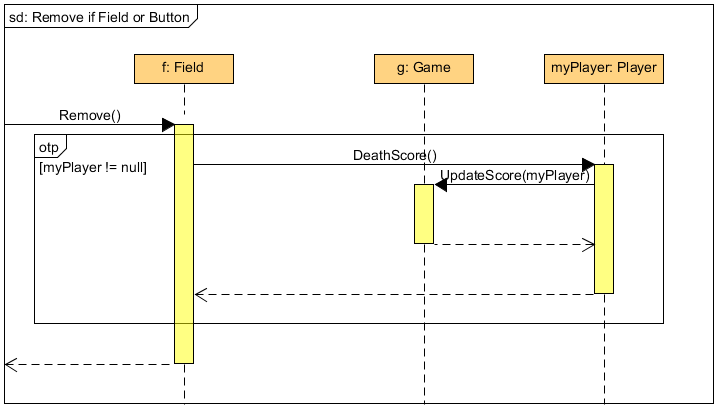
******

***3.4.14 Falling into hole***

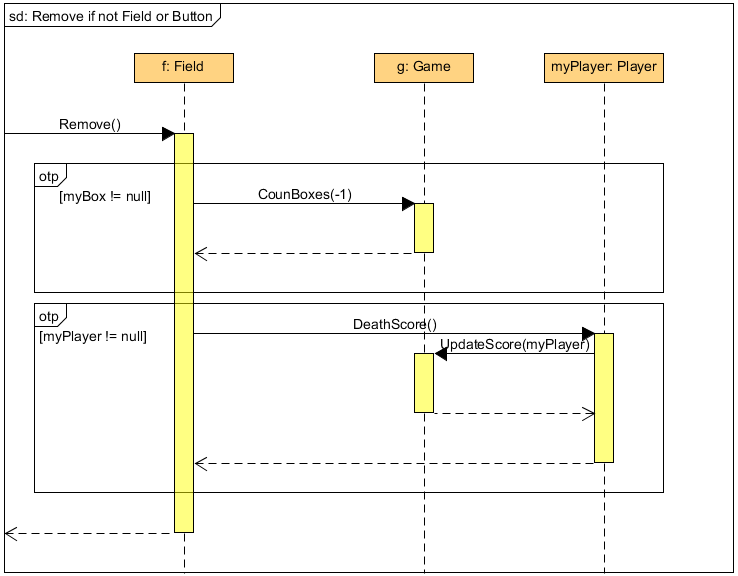
Egy Player vagy egy Box ha Hole-ra lép, leesik és eltűnik.

******

***3.4.15 Field és Button mező Remove***

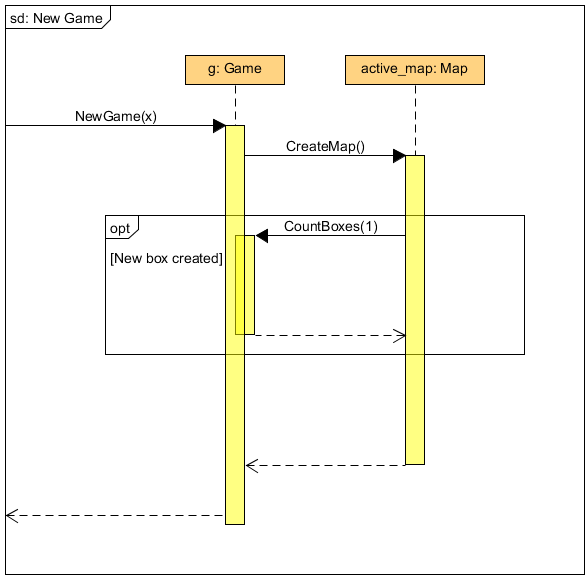
******

***3.4.16 Minden más Remove***

******

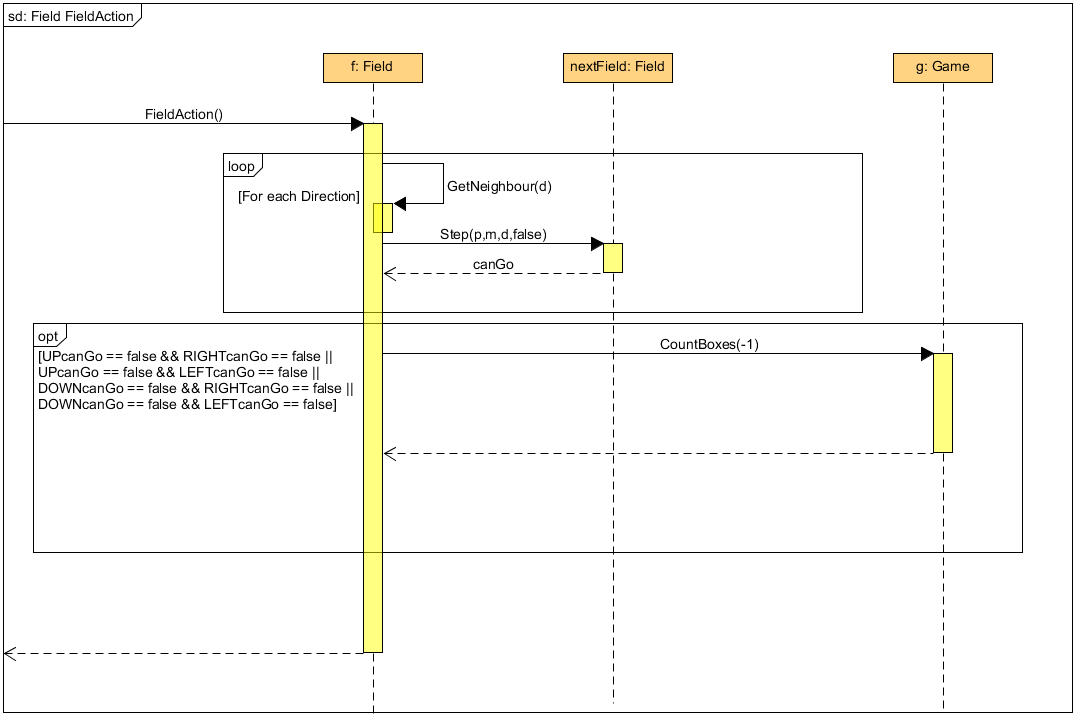
***3.4.17 New Game***

Új játék indításakor léterjön egy Map, ami feltöltődik Fieldekkel és Boxokkal.

******

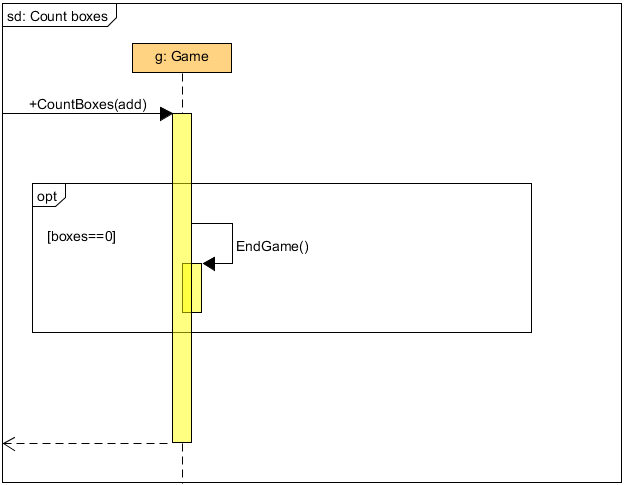
***3.4.18 Field Field Action***

Minden irányba megnézzük hogy tolható e. A Step ha false az utolsó paramétere akkor nem lép csak rekurzívan lekérdezi hogy arra lehet e menni. Ha a doboz “beszorult” akkor csökkenti a pályán lévő dobozok számát. Amikor a dobozok elfogynak a játéknak vége.

******

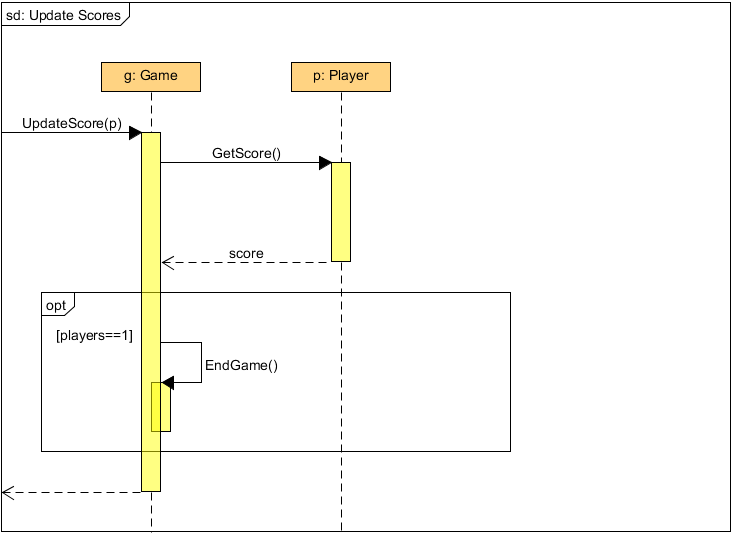
***3.4.19 Count Boxes***

A dobozok számát mindig a megadott értékkel “növeli” ha a dobozok elfogynak, tehát nincs több mozgatható, akkor a játék véget ér.

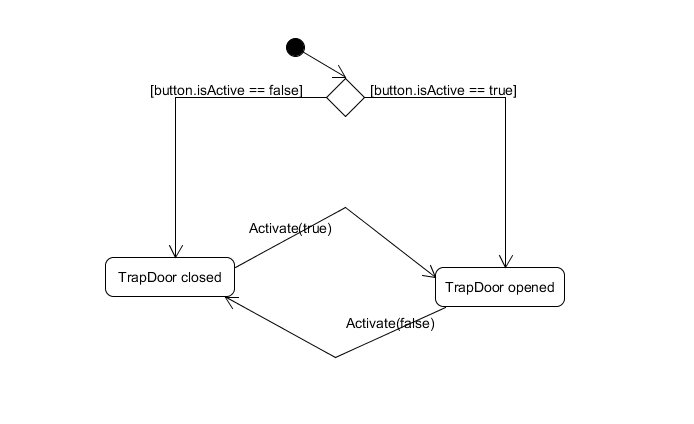
******

***3.4.20 Update Scores***

A játékosok pontját a DeathScore() -1-re állítja, ha ezt érzékeli az UpdateScore akkor csökkenti a Playerek számát. Ha a Playerek száma 1 akkor az utolsó játékos nyer és a játék véget ér.

******

* 1. ***State-chartok***



* 1. ***Napló***

|  |  |  |  |
| --- | --- | --- | --- |
| **Kezdet** | **Időtartam** | **Résztvevők** | **Leírás** |
| 2018.02.20 11:00 | 40 perc | Czetter  Frontó  Tomori  Verasztó | Megbeszélés:  Objektumkatalógus megvitatása. Feladatok kiosztása |
| 2018.02.20 12:20 | 80 perc | Frontó | 3.2 Statikus struktúra diagramok |
| 2018.02.20 12:20 | 80 perc | Tomori | 3.2 Statikus struktúra diagramok |
| 2018.02.20 12:20 | 80 perc | Czetter | 3.2 Statikus struktúra diagramok |
| 2018.02.20 12:20 | 80 perc | Verasztó | 3.2 Statikus struktúra diagramok |
| 2018.02.20 14:00 | 2 óra | Frontó | 3.3 Szekvencia diagramok |
| 2018.02.20 14:00 | 2 óra | Tomori | 3.3 Szekvencia diagramok |
| 2018.02.20 14:00 | 2 óra | Czetter | 3.3 Szekvencia diagramok |
| 2018.02.20 14:00 | 2 óra | Verasztó | 3.3 Szekvencia diagramok |
| 2018.02.21 10:00 | 2 óra | Frontó | 3.3 Szekvencia diagramok |
| 2018.02.21 10:00 | 2 óra | Czetter | 3.3 Szekvencia diagramok |
| 2018.02.21 10:00 | 2 óra | Verrasztó | 3.3 Szekvencia diagramok |
| 2018.02.21 10:00 | 2 óra | Tomori | 3.3 Szekvencia diagramok |
| 2018.02.21 13:00 | 1 óra | Czetter | 3.3 Szekvencia diagramok |
| 2018.02.21 13:00 | 1 óra | Frontó | 3.3 Szekvencia diagramok |
| 2018.02.21 13:00 | 1 óra | Tomori | 3.3 Szekvencia diagramok |
| 2018.02.21 13:00 | 1 óra | Verrasztó | 3.3 Szekvencia diagramok |
| 2018.02.22 10:00 | 3,5 óra | Czetter  Frontó | 3.2 Osztály diagramok |

|  |  |  |  |
| --- | --- | --- | --- |
| 2018.02.22 10:00 | 3,5 óra | Verrasztó  Tomori | 3.3 Szekvencia diagramok |
| 2018.02.23 | 3 óra | Czetter  Frontó  Verrasztó | 3.3 Szekvencia diagramok |
| 2018.02.25 15:00 | 6 óra | Frontó  Czetter  Verrasztó  Tomori | Források integrálása, dokumentum formázása. |