3. Analízis modell kidolgozása 1

40 – *[scrum_that]*

Konzulens:
Szabó Ádám Imre

Csapattagok:

Kovács Levente Ákos	CM6UKU	vazul250@gmail.com
Lovász Attila Bence	INCMI7	attonet2@gmail.com
Graics Vince	HY9XQ6	wince17@gmail.com
Magyar Milán Bertalan	MCDNQL	milangfx@gmail.com
Tóth Krisztián Dávid	J38GIK	tht.krisztian@gmail.com

2015. március 2.

Tartalomjegyzék

Ábrák jegyzéke

3. Analízis modell kidolgozása 1

3.1. Objektum katalógus

3.1.1. Glue

A „Glue” objektum megvalósít egy adott tulajdonságú akadályt. Amely robot belemegy, annak a sebességét megfelezi.

3.1.2. GUI

A grafikus felületet megvalósító objektum. Ez az objektum maga a menü ami a játék indítása után ugrik fel, itt találhatóak a beállítások (mint például a gondolkodás idő és a maximális játék idő vagy a körök száma) és a játékmódok. Gombnyomásra fogja elindítani a játék működési szálát. Ez az objektum kezeli az ablak eseményeit és a játék bezárását.

3.1.3. HUD

Ez az objektum követi és nyilvántartja, hogy a robotok hány checkpoint-on mentek át, mennyi olaj és ragacs van náluk amit felhasználhatnak, illetve kiírja a képernyőre a hátramaradó időt és a megtett körök számát. Feladata, hogy minden körben megvizsgálja, hogy a robotok elérték-e a következő checkpointot.

3.1.4. MapBuilder

Fájlból beolvassa és létrehozza a memóriában a pályát, a kezdő pozíciókat és a checkpointokat reprezentáló objektumokat. Mivel a MapBuilder objektum tárolja a pályát így feladat, hogy vizsgálja a robotok azon belül tartózkodását.

3.1.5. Obstacle

Az Obstacle absztrakt osztály, mely a Unit osztályból származik le. Ez az osztály fogja összefogni a pályán található (lerakott) akadályokat és bevezet egy absztrakt függvényt, ami a leszármazottakban implementálva érvényesíti hatását (lassítás, csúszás) egy robotra.

3.1.6. Oil

Ez az objektum az Obstacle osztály leszármazottja. Hasonlóan a Glue objektumhoz, egy adott hatást valósít meg, ami letiltja a következő körben történő irányítását a robotnak, ami belelépett.

3.1.7. Phoebe

A játék logikát megvalósító objektum. Listában tárolja a pályán tartózkodó robotokat, akadályokat és figyeli, hogy mikor ér véget a játék. A „Phoebe” objektumrajzolja ki az objektumokat a pályán és szálként indítható osztályt, melyben maga a játék fut. Játékindításkor berakja a pályára a robotokat és az akadályokat a kezdő pozíciókba. Ebben az objektumbantörténnek az ellenőrzések (akadályba vagy robotba ütközések, pályáról leesés).

3.1.8. Robot

Olyan objektum, mely a pályán található robotokat valósítja meg. Leírja a viselkedésüket és a kezelésüket. A „Robot" osztály a Unit-ból származik le, ezáltal van pozíciója és az ütközés is le van kezelve. Felelős a mozgásért, megállapítja egy adott akadállyal vagy robottal ütközött-e és kezeli a felhasználó által leütött gombokat.

3.1.9. Timer

Az eltelt időt és a fennmaradt idő nyilvántartásáért felelős. Ilyen például a játék elején három másodperces visszaszámlálás vagy a időlimites játékmód esetén, amikor a maximális időtől számol visszafelé.

3.1.10. Unit

A Unit absztarkt osztály a pályán található objektumoknak a szülőosztálya. Tárolja a pozíciót, egy sokszöget, amivel vizsgálható az ütközés és a pályán megjelenő képüket. Továbbá ez az osztály valósítja meg a függvényt, ami ellenőrzi azt, ha két Unit ütközik (robotok egymással, robot akadállyal).

3.2. Osztályok leírása

3.2.1. Glue

- Felelősség
  A játékban szereplő Ragacs foltok viselkedését leíró osztály
- Ősosztályok
  Unit→Obstacle
- Metódusok
  - void **effect**(Robot r): Ütközéskor hívja meg az ütközést vizsgáló függvénye a Robot osztálynak. Módosítja a robot slowed értékét a 50%-ra a robot slowed attribútum setterének meghívásával.

3.2.2. GUI

- Felelősség
  A grafikus felületért felelős osztály, mely a menüt és a játékot megjeleníti.
- Attribútumok
  - **Phoebe** game: referencia a játékra
- Metódusok
  - **GUI**(): Konstruktor. Beállítja az ablak nevét, létrehozza az ablak elemeit, elrendezi őket és beállítja a figyelőket(ActionListener).

3.2.3. HUD

- Felelősség
  A robotok ragacs- és olajkészletét, illetve megtett köreit és checkpontjait számolja. Megvalósítja a checkpoint ellenőrzést.
- Attribútumok
  - **int[]** checkpointReached: Minden robothoz külön tárolja a legutoljára érintett checkpoint sorszámát.

  - **int[]** lap: Minden robothoz tárolja a megtett körök számát.
  - **int[]** numGlue: Minden robothoz tárolja a ragacsok számát.
  - **int[]** numOil: Minden robothoz tárolja az olajok számát.
  - **List<Shape>** checkpoints: Tárolja a checkpointokat reprezentáló objektumokat List adatszerkezetben. A checkpointSearch függvény kérdezi le ebből a következő checkpoint helyzetét.
  - **List<Robot>** robots: A robotokat tároló List adatszerkezet. A checkpointSearch függvény kérdezi le ebből a robotokat, majd azok helyzetét.

- Metódusok
  - **HUD**(List<Robot> robs): Konstruktor, inicializálja a köröket számláló változót, az érintett checkpointokat, a ragacs és olajkészleteket.
  - void **checkpointSearch**(): Ellenőrzi hogy a robotok teljesítették-e a következő checkpointot.
  - void **setCheckpoints**(List<Shape> checkObj): Checkpointokat reprezentáló adatszerkezet betöltése.

3.2.4. MapBuilder

- Felelősség
  A pálya felépítéséért, a checkpointok tárolásáért és a robot pályán tartózkodásának vizsgálatáért felelős osztály.

- Attribútumok
  - **Shape** map: A pályát reprezentáló objektum.
  - **List<Shape>** checkpoints: Tárolja a checkpointokat reprezentáló objektumokat List adatszerkezetben.
  - **int[]** startPosPlayerOne: Meghatároz egy (x,y) koordinátát, ahol az első játékos kezd.
  - **int[]** startPosPlayerTwo: Meghatároz egy (x,y) koordinátát, ahol az második játékos kezd.

- Metódusok
  - **MapBuilder**(): Konstruktor, a pálya beolvasása fájlból, majd létrehozása.
  - boolean **fallingDown**(Shape othershape): Igaz értéket ad vissza, ha a robot leesett a pályáról, hamisat ha még rajta van.

3.2.5. Obstacle

- Felelősség
  A pályán/játékosoknál lévő különböző akadályokat (ragacs,olaj) összefogó ősosztály.

- Ősosztályok
  Unit

- Attribútumok
  - **int** WIDTH: Az akadályokat jellemző szélesség. Szükség van rá, hogy létrehozzuk a leszármazottak hitbox-át(sokszög pályaelem).
  - **int** HEIGHT: Az akadályokat jellemző hosszúság. Szükség van rá, hogy létrehozzuk a leszármazottak hitbox-át(sokszög pályaelem).

- Metódusok

  - **Obstacle**(int x, int y, String imagelocation): meghívja a Unit konstruktorát a megadott adatokkal és létrehoz egy sokszög elemet ami reprezentálja a pályán majd.
  - void **effect**(Robot r): Meghatározza, milyen hatással van a robotra, ha érintkezik egy Obstacle-lel. Absztrakt.

3.2.6. Oil

- Felelősség
  A pályára lerakható olaj megvalósítása. Ha belelép egy játékos egy ilyen olajfoltba az effect függvény letiltja a mozgatást az adott roboton a következő ugrásig.
- Ősosztályok
  Unit → Obstacle
- Metódusok
  - **Oil**(int x, int y, String imagelocation): Egy Oil elem létrehozásáért felelős.
  - void **effect**(Robot r): Meghatározza, milyen hatással van a robotra, ha beleugrik egy olajfoltba. Ebben az esetben letiltja a játékost, hogy irányt váltson.

3.2.7. Phoebe

- Felelősség
  A játék motorját képviselő osztály. A robotok pozíciójáért, az akadályok elhelyezéséért és a játék végéért felel.
- Attribútumok
  - **boolean** ended: Állapot változó, ha vége a játéknak, akkor true. Ha beteljesül egy játék végét jelentő esemény, akkor ezen a változón keresztül leáll a játék és megállapítódik a nyertes.
  - **List<Robot>** robots: A játékban szereplő robotok listája.
  - **List<Obstacle>** obstacles: A játékban szereplő akadályok listája.
  - **HUD** hud: A játékosok előrehaladását, ragacs és olajkészleteit tartja számon
  - **MapBuilder** map:
- Metódusok
  - **Phoebe**(Setting set): A játék felépítése, a robotok lista, az akadályok lista létrehozása
  - void **run**(): Ez a metódus futtatja a főciklust, amelyben maga a játék működik.

3.2.8. Robot

- Felelősség
  A játékban résztvevő robotok viselkedését és kezelését leíró osztály.
- Ősosztályok
  Unit
- Interfészek
  Nincs interfésze
- Attribútumok
  - **int** staticID: Az osztályhoz tartozó statikus azonosító, a példány azonosítójának(id) meghatározásához szükséges.

  - **int** HEIGHT: A robot képének magassága, collision detektálásnál, továbbá az irányítást segítő nyíl kezdő koordinátájának meghatározásánál szükséges.
  - **int** WIDTH:A robot képének szélessége, funkcionalitásban hasonló a WIDTH-hez.
  - **int** ID: A robot példányának egyedi azonosítója, a keyconfig sorának indexelésére és a collison detektálásnál az önmagával való ütközés kivédésére szükséges.
  - **double** slowed: A sebesség modosításáért felel, default értéje 1.0, amennyiben ragacsba lép a robot ez 0.5-re módosul és minden ugrás végén visszaáll az eredeti értékére, ugrásnál ezzel szorozzuk be a végkordinátát kiszámító sugár hosszát.
  - **boolean** oiled: Azt jelzi, hogy olajba lépett-e, ennek hatására a mozgás iránya módosíthatatlanná válik egy kis időre.
  - **int** arrowendx: A robot irányítását segítő nyilnak az x koordinátája, a nyíl kirajzolásánál van szerepe.
  - **int** arrowendy: A robot irányítását segítő nyilnak az y koordinátája, a nyíl kirajzolásánál van szerepe.
  - **double** alpha: A robot irányítását segítő nyíl vízszintessel bezárt szöge. A nyil kirajzolásánál, az ugrás végpontjának meghatározánál van szerepe.
  - **boolean** moved: Azt jelöli, hogy lépett-e már a robot az aktuális körben. A megjelenítésnél(nyilat ugrás közben nem jelenítjük meg),illetve az irányítás letiltásánál van szerepe(olajba lépés esetén).
  - **int[][]** keyconfig: A játékosok írányítását tároló mátrix. A játékosok irányítását ennek segítségével határozzuk meg a keyPressed függvénybe. A sor meghatározza a felhasználóhoz tartozó gombokat, az oszlopok a funkciók(olaj/ragacs lerakás, a nyíl jobbra/balra mozgatása)

- Metódusok

  - **Robot**(int x,int y,String imagelocation,Phoebe p): Létrehoz egy robotot a megadott x,y kordinátákon, betölti a képét az imagelocation cím alapján, továbbá eltárolja a játékmotor referenciáját.
  - **void deathanimation**():A Robot halálának gafikus megjelenítéséért felelős függvény.
  - boolean **collisionWithObstacle**(Obstacle o): Ellenőrzi hogy a robot ütközött-e az akadállyal, igazzal tér vissza ha igen, hamissal ha nem.
  - boolean **collisionWithRobot**(Robot r): Ellenőrzi hogy a robot ütközött-e másik robottal , igazzal tér vissza ha igen, hamissal ha nem. ID alapján kiszüri ha önmagára hívják meg.
  - void **keyPressed**(Keyevent e):A robot irányítását megvalósító függvény, a játékmotor keylistener-e által hívódik meg, a lenyomott billentyű keyevent-jére. A következő ugrás beállítása, a ragacs/olaj lerakása történhet itt. A keyconfig változó felhasználásával.

3.2.9. Timer

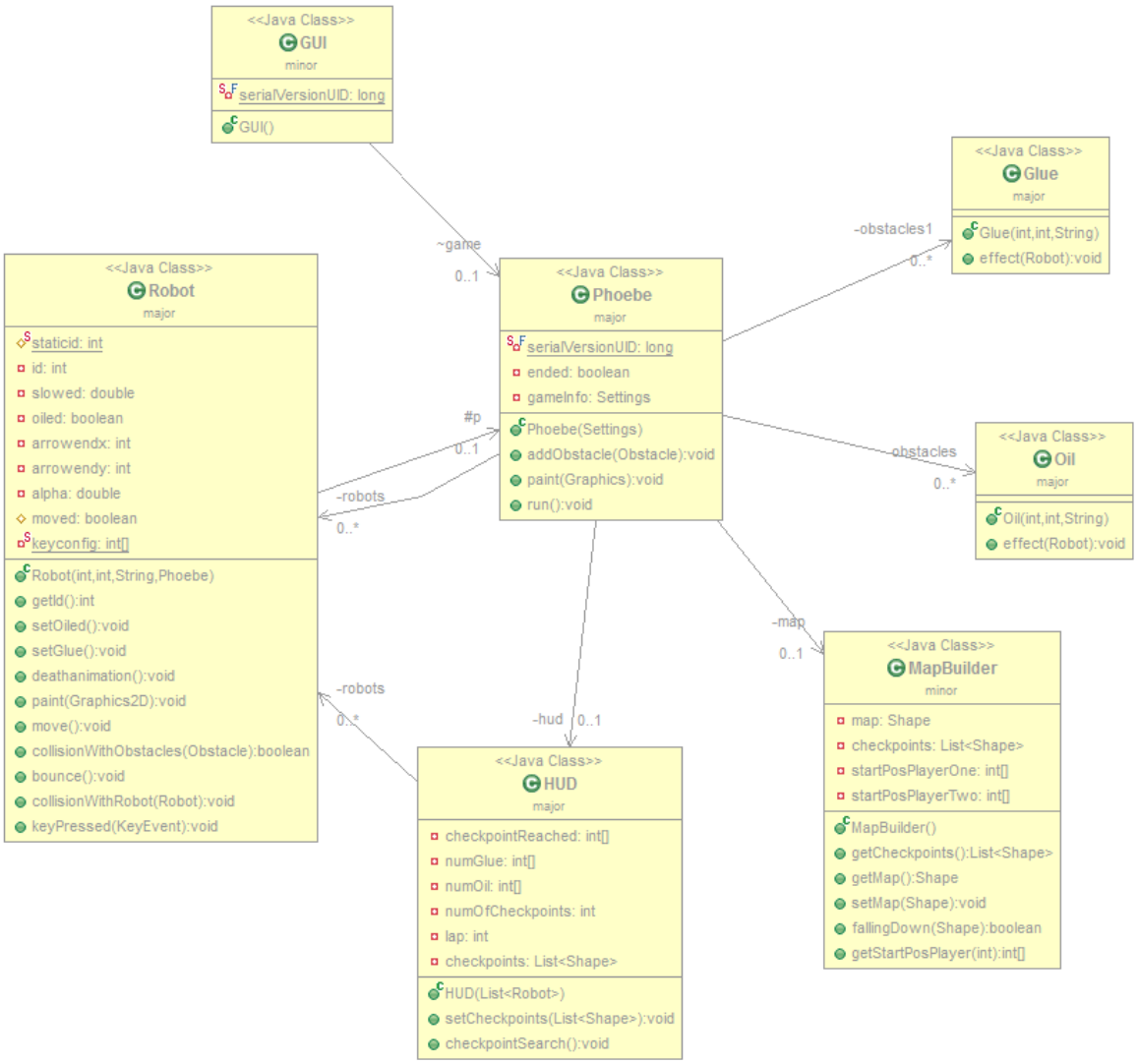
- Felelősség
  A visszaszámlálásért (idő, director time) felelős.

- Metódusok
  - **Timer**(int i): Konstruktor, inicializál egy viszaszámláló órát.
  - boolean **isZero**(): Igazzal tér vissza ha a megadott idő lejárt;

3.2.10. Unit

- Felelősség
  A pályán található objektumokért felel és azok viszonyáról (például ütközésükről).
- Attribútumok
  - **int** x: Az egység x koordinátája
  - **int** y: Az egység y koordinátája
  - **Rectangle** hitbox: Az egységet a pályán reprezentáló sokszög.
- Metódusok
  - void **move**() : Absztrakt függvény, mely a leszármazottakban fog megvalósulni. Az egységek mozgásáért felelős.
  - boolean **intersect**(Unit u): Két egység ütközését meghatározó függvény.

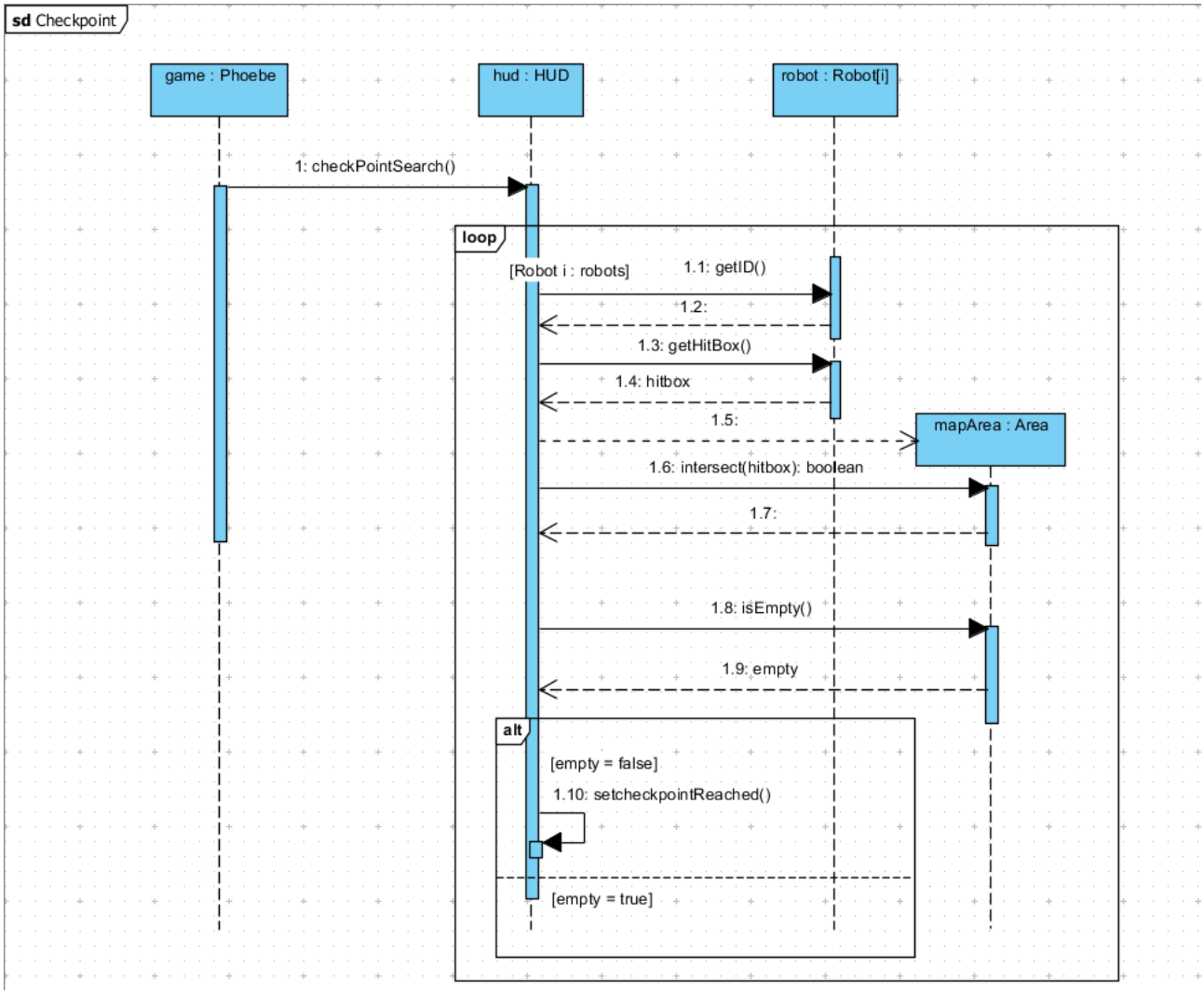
3.3. Statikus struktúra diagramok



3.1. ábra. Statikus struktúra diagramm

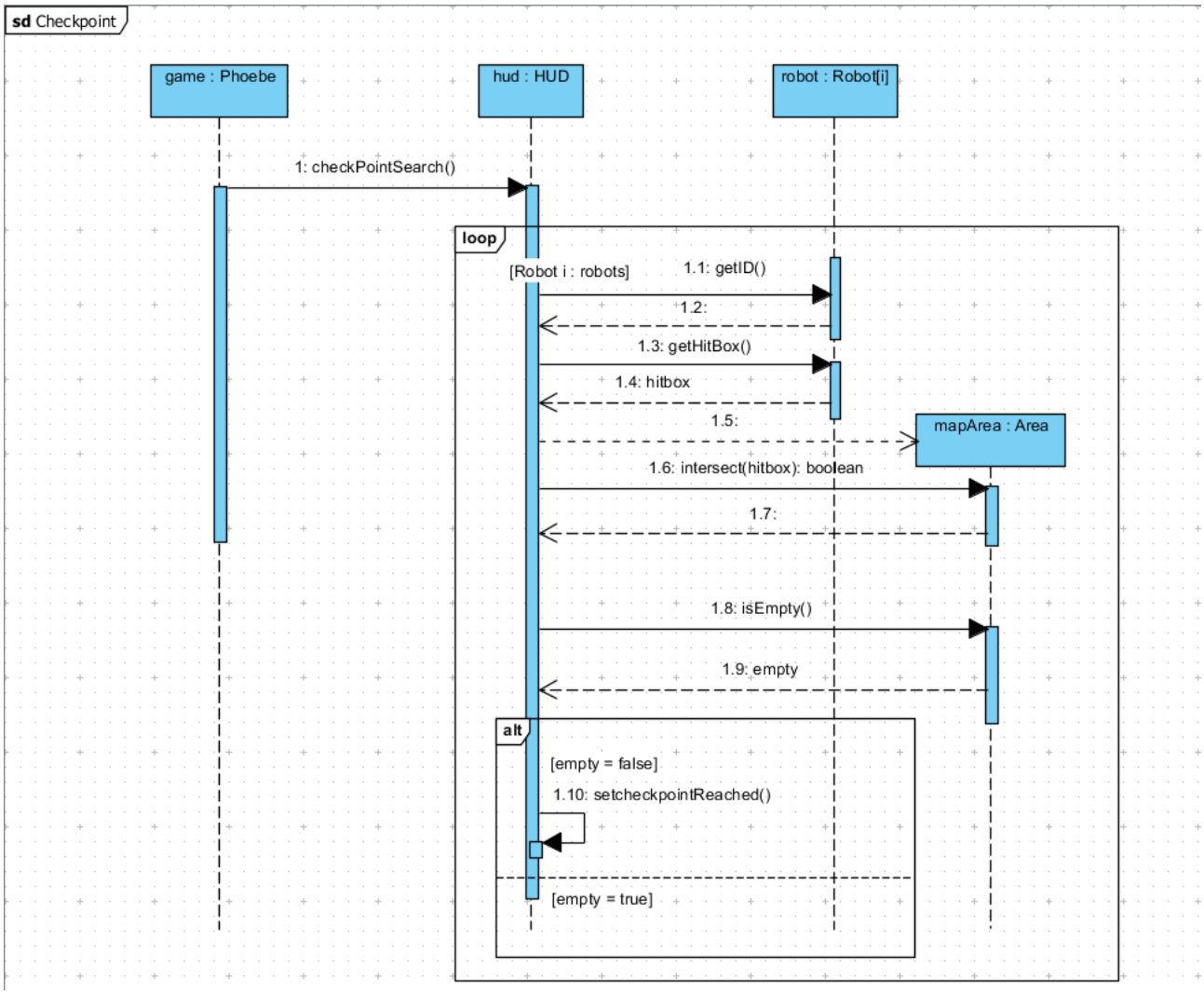
3.4. Szekvencia diagramok

3.4.1. Robot::CheckpointSearch



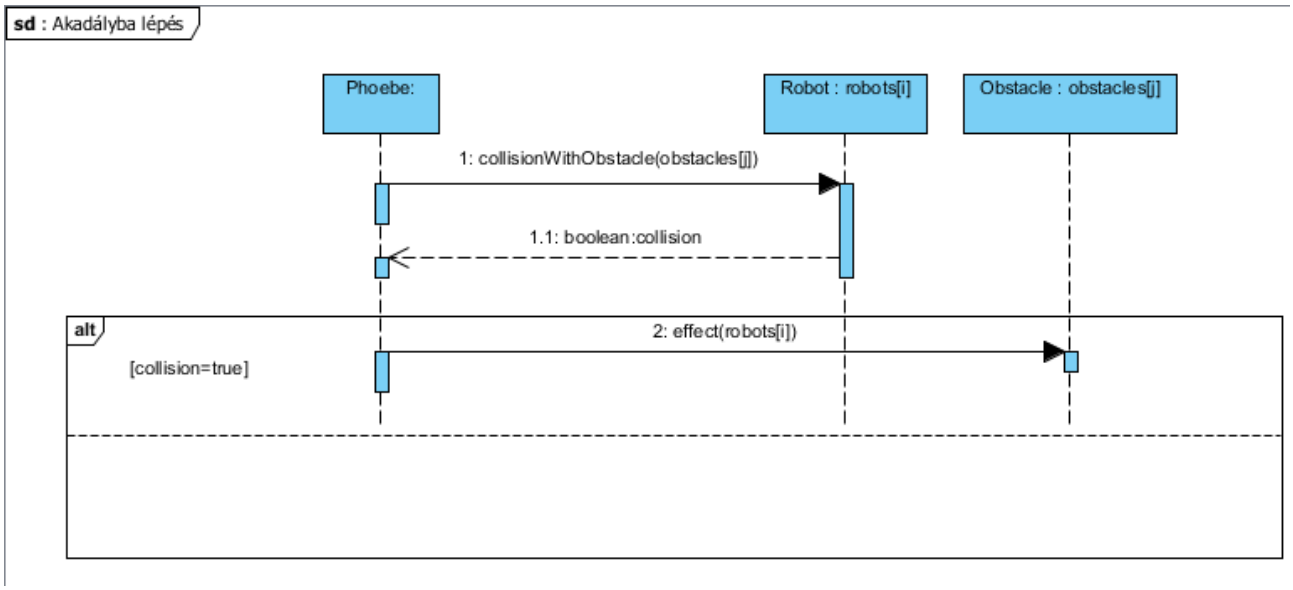
3.2. ábra. Következő checkpoint vizsgálata

3.4.2. Robot::CheckpointSearch



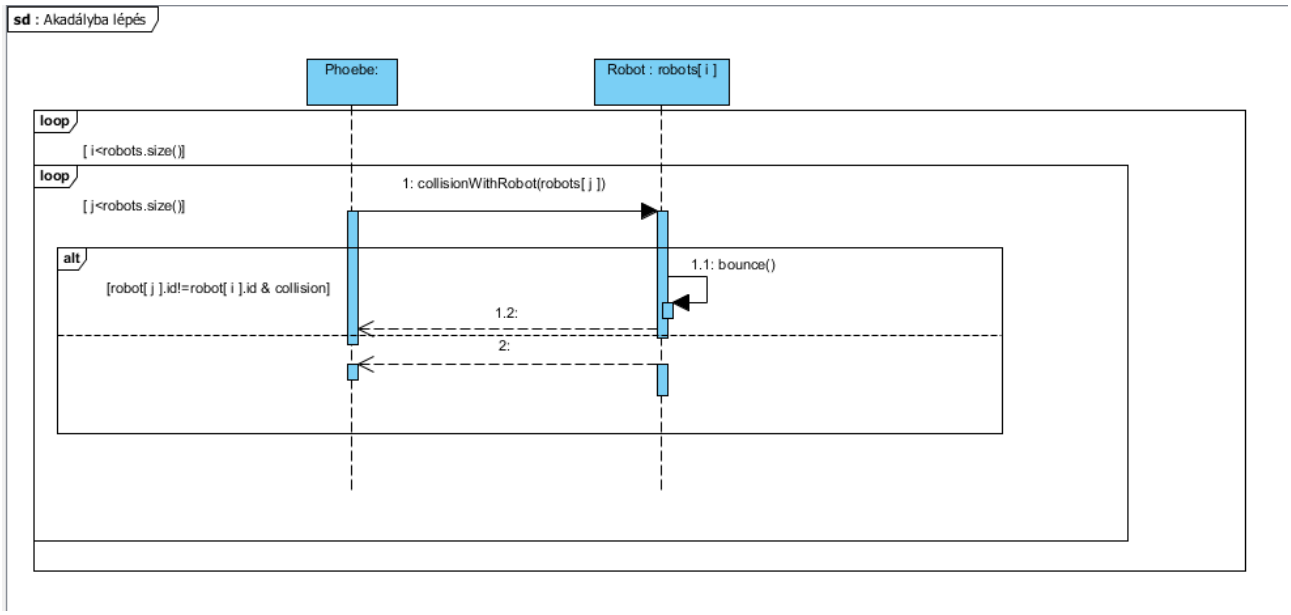
3.3. ábra. Következő checkpoint vizsgálata

3.4.3. Robot::CollisonWithObstacle



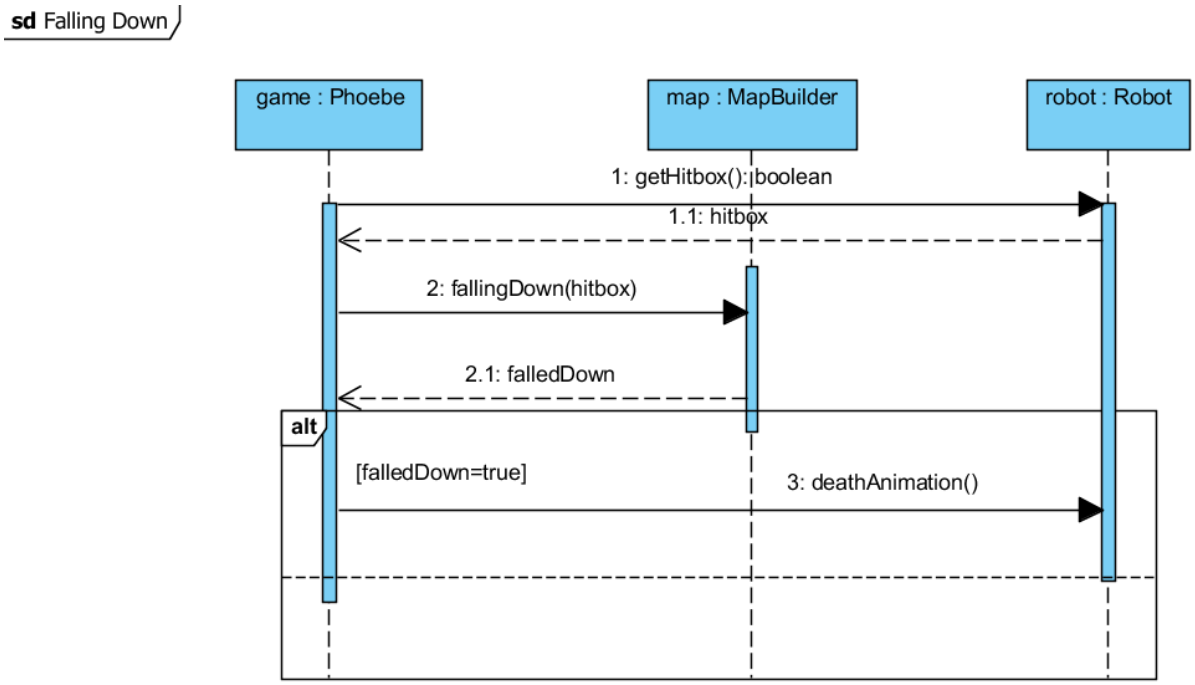
3.4. ábra. Robot ütközése akadállyal

3.4.4. Robot::CollisionWithRobot



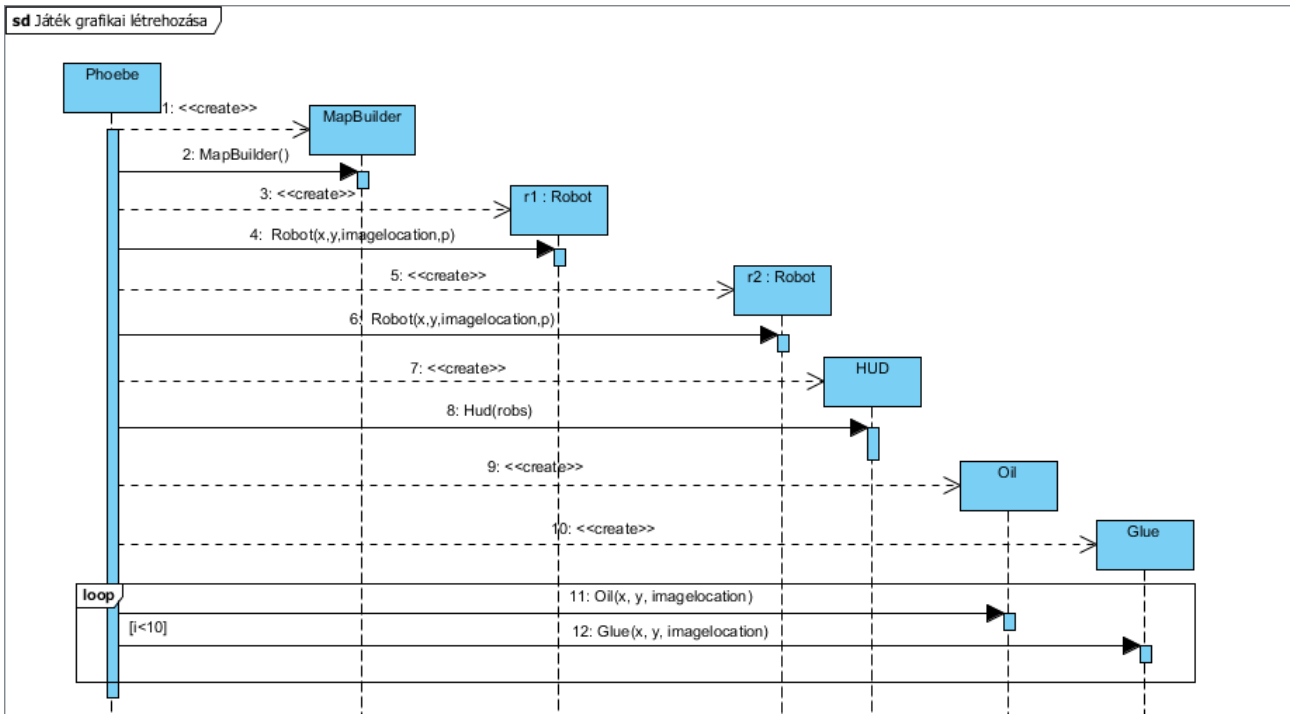
3.5. ábra. Robot ütközése akadállyal

3.4.5. Robot::FallDown



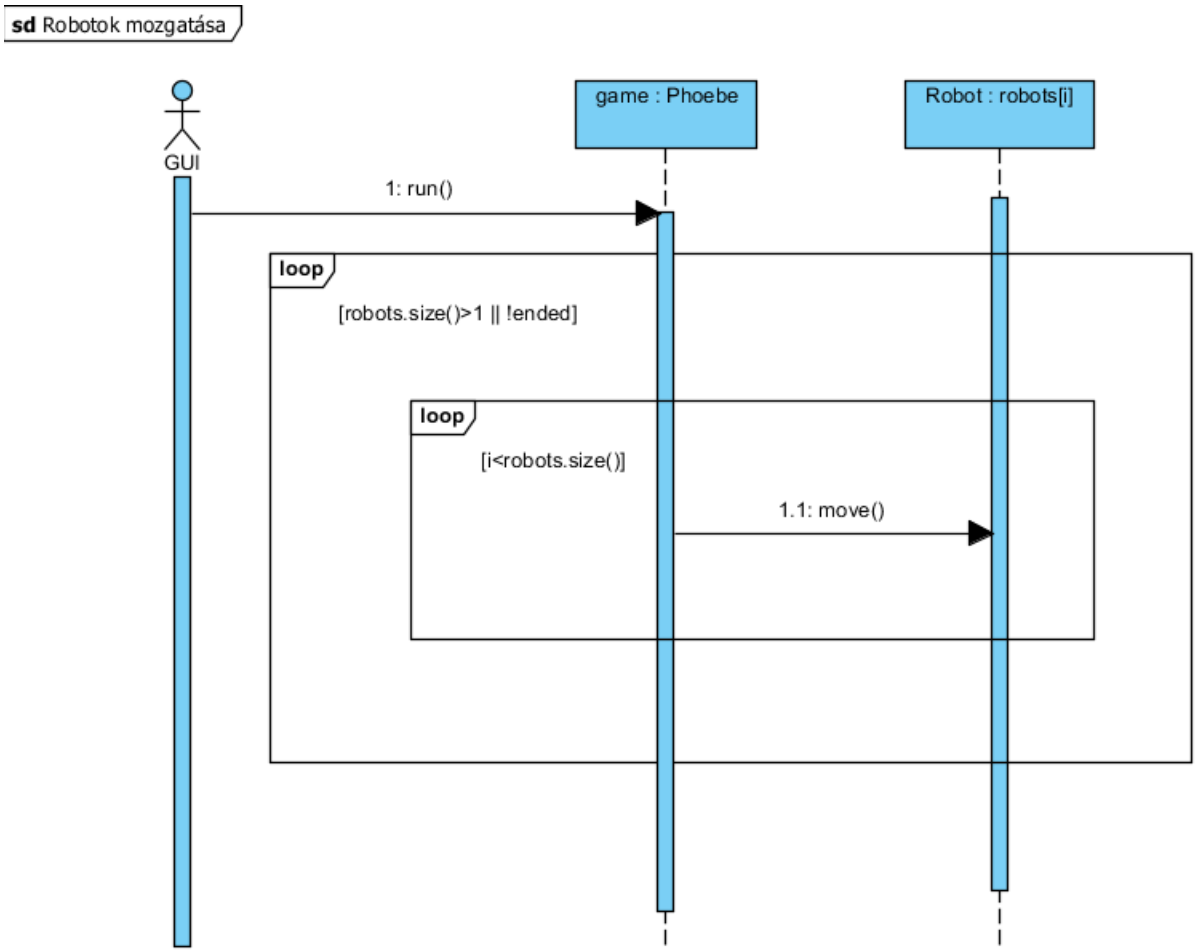
3.6. ábra. Robot leesése a pályáról

3.4.6. Robot::InitGame



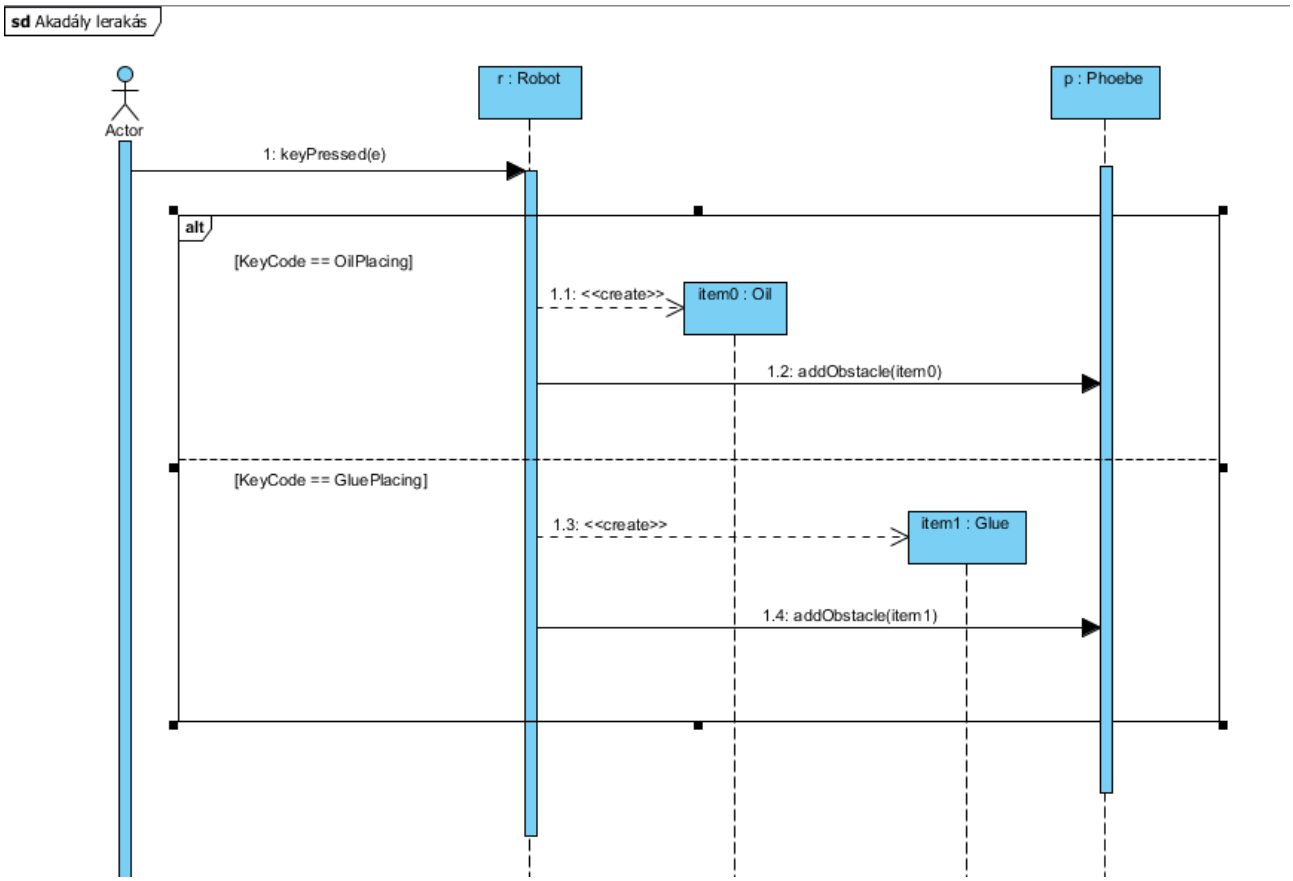
3.7. ábra. A játék inicializálása

3.4.7. Robot::Move



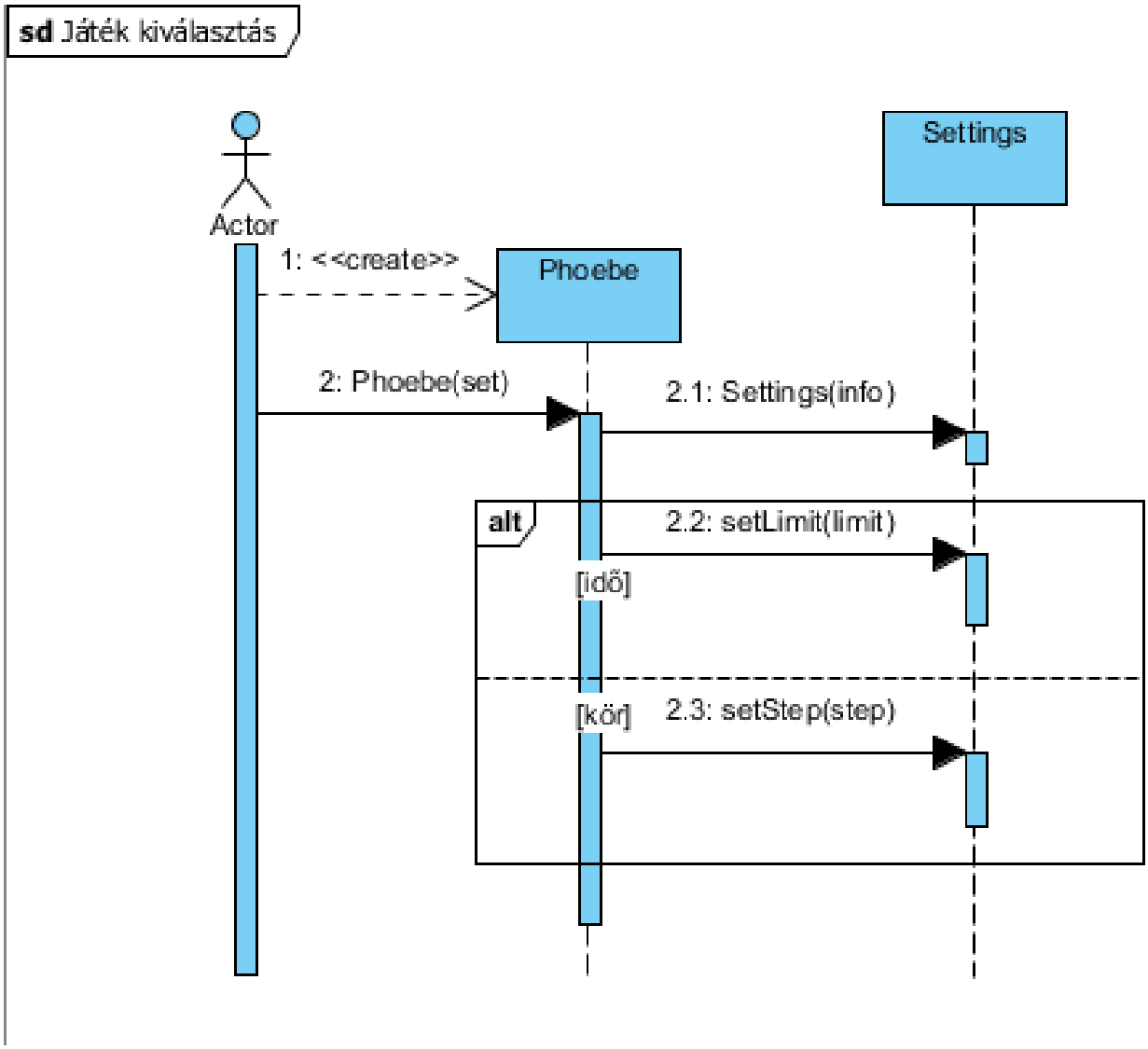
3.8. ábra. A robot mozgatása

3.4.8. Robot::NewObstacle



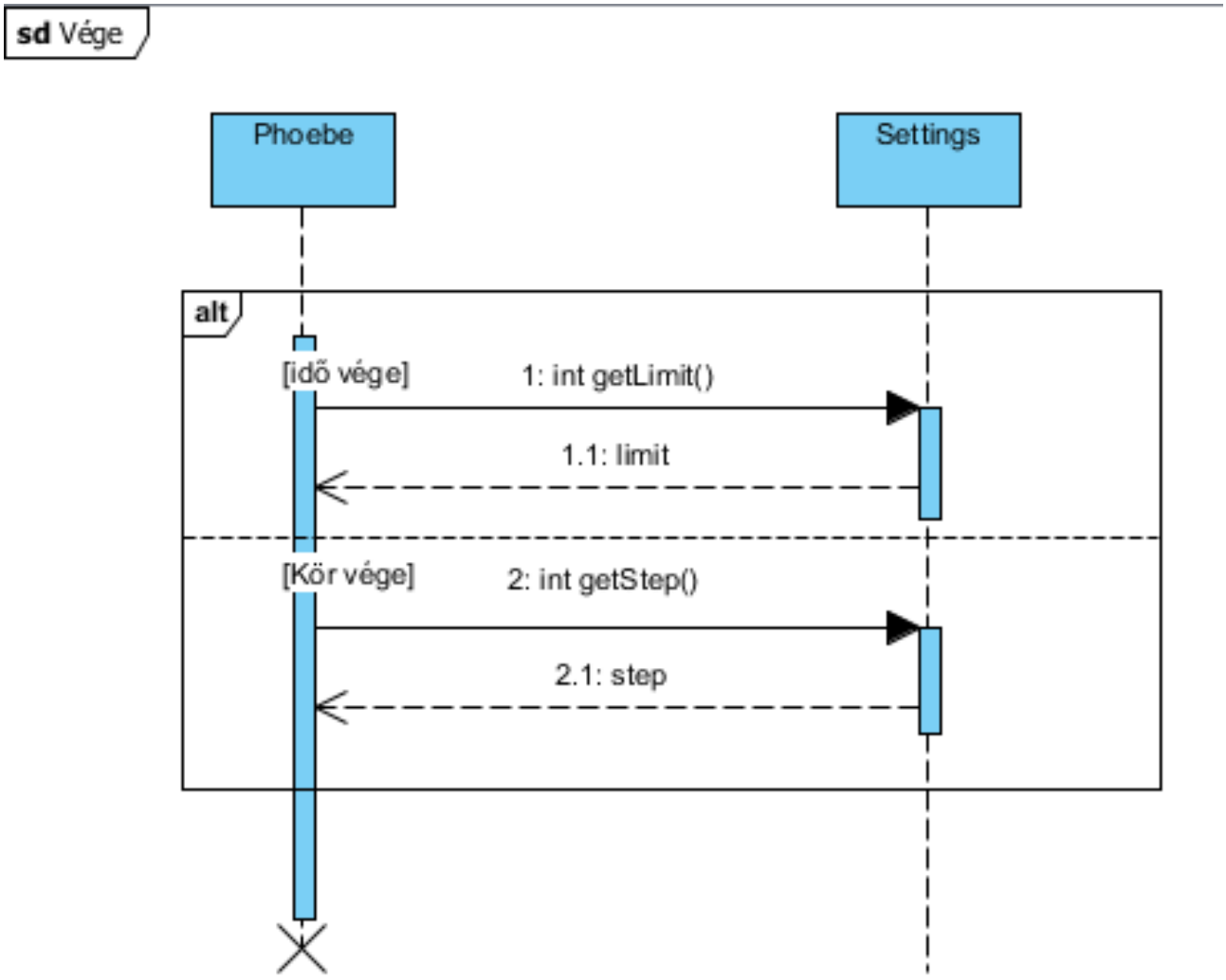
3.9. ábra. Akadály lerakása

3.4.9. Robot::Settings



3.10. ábra. A játé beállításainak kiválasztása

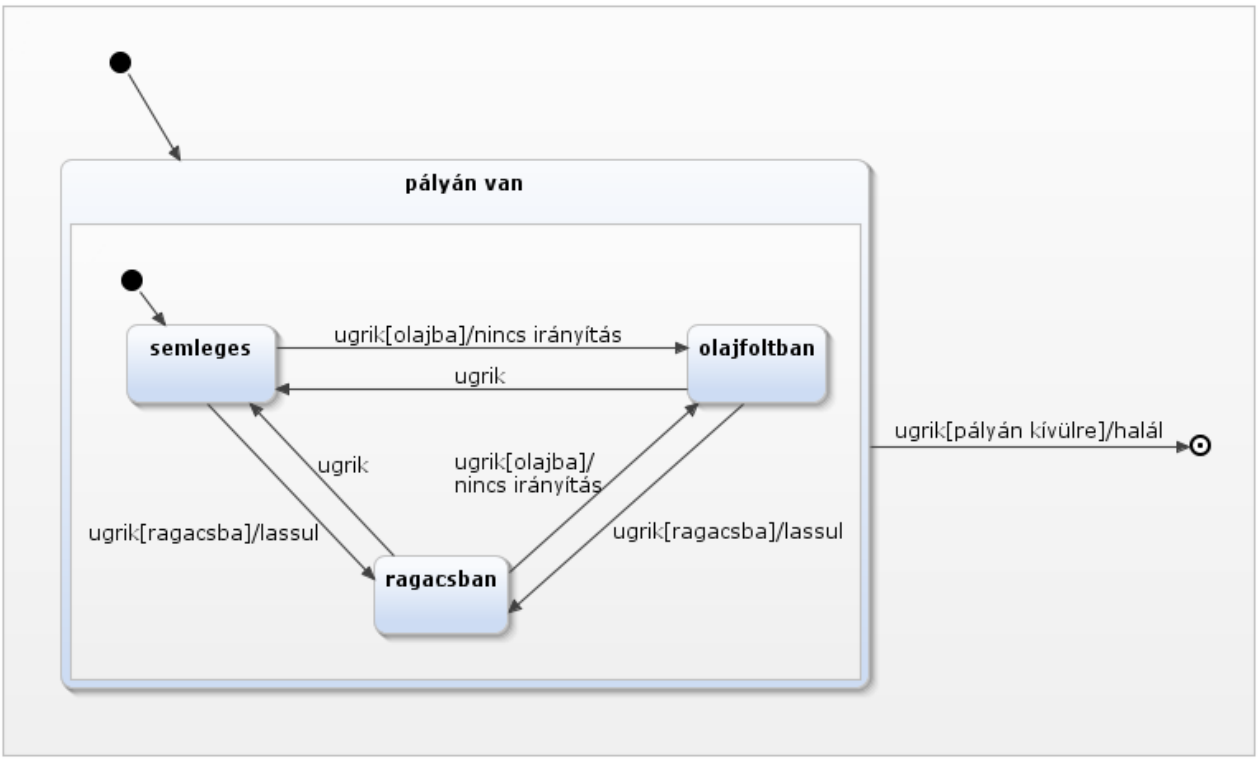
3.4.10. Robot::End



3.11. ábra. Játék vége

3.5. State-chartok

3.5.1. Robot::States



3.12. ábra. A robot állapotai

3.6. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2015.02.25. 12:30	1,5 óra	**Kovács** **Lovász** **Tóth** **Graics** **Magyar**	Analízis modell előkészítése
2015.02.25. 18:15	4 óra	**Kovács**	Analízis modell kidolgozása
2015.02.26. 19:45	2 óra	**Kovács** **Lovász** **Tóth** **Graics** **Magyar**	Analízis modell előkészítése, feladatok kiosztása
2015.02.28. 18:15	3 óra	**Graics**	Objektum katalógus készítése
2015.02.28. 20:00	2 óra	**Magyar**	State-chartok készítése
2015.02.28. 20:00	4 óra	**Lovász**	Osztályok leírása
2015.02.28. 21:00	1,5 óra	**Kovács**	Osztályok leírása
2015.03.01. 18:00	3 óra	**Lovász** **Magyar** **Graics**	Szekvencia diagrammok készítése
2015.03.01. 17:00	7 óra	**Tóth**	Analízis modell kidolgozása, Dokumentálás