4. Analízis modell kidolgozása 2.

41 – Java Junk

Konzulens:
Szabó Ádám Imre

Csapattagok:

Dusik Máté	ALHW9D	dusikmate@gmail.com
Kachichian Lucienne	FLP3X6	anchoanhelo@gmail.com
Pohubi Zoltán László	EYJIX1	pohubi.zoltan@gmail.com
Szendi Tamás Pál	XEENOE	szendi.tam@gmail.com
Zelenák Gellért	ZDZ0U1	ze.gellert@gmail.com

2015. március 23.

4. Analízis modell kidolgozása

4.1 Objektum katalógus

4.1.1 Game

Ez az objektum felelős a létrehozási, fő kirajzolási és frissítési metódusok meghívásáért, vezeti a játék időkezelését és figyeli, hogy életben vannak-e még a játékosok által vezérelt robotok.

4.1.2 Glue

Egy ragacs kirajzolásáért és frissítéséért felelős objektum.

4.1.3 Map

Tárolja a pályát és megállapítja, hogy az adott pozíció a pálya területén található-e.

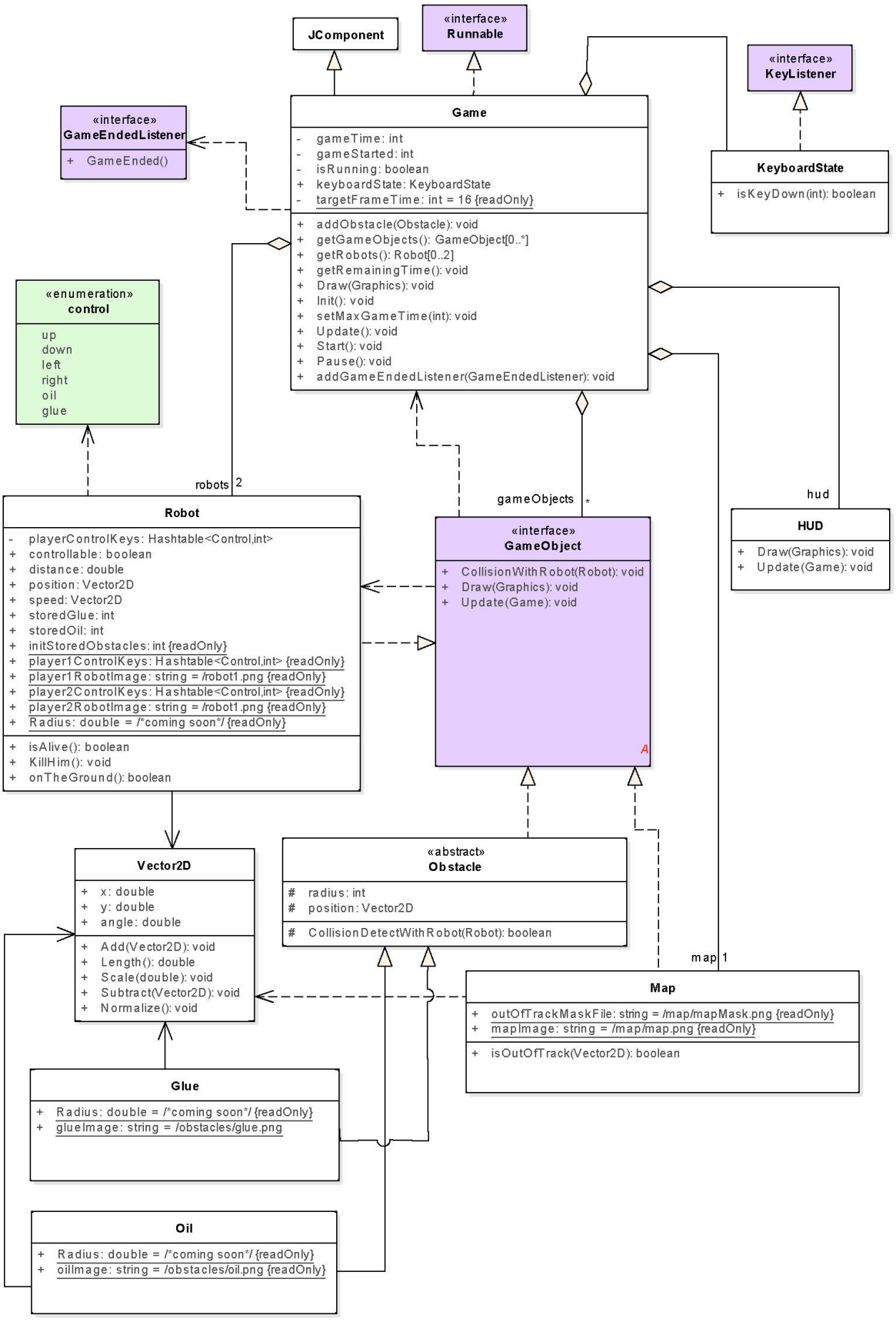
4.1.4 Oil

Egy olajfolt kirajzolásáért és frissítéséért felelős objektum.

4.1.5 Robot

A robot ugrál a pályán frissítés hatására, kirajzolja magát, tud ütközni másik robottal, tárolja a pozícióját, a sebességét, az aktuális olaj- és ragacskészletét, illetve a verseny kezdete óta megtett távolságot.

3.2 Statikus struktúra diagramok



4.3 Osztályok leírása

4.3.1 Game

· Felelősség

Ez az objektum felelős a létrehozási, fő kirajzolási és frissítési metódusok meghívásáért, vezeti a játék időkezelését és figyeli, hogy életben vannak-e még a játékosok által vezérelt robotok.

· Ősosztályok

- JComponent

· Interfészek

- Runnable

· Attribútumok

- **int gameTime**: egy verseny időtartamának hossza
- **int gameStarted**: a verseny elkezdésének időpontja
- **boolean isRunning**: megadja, hogy a verseny éppen folyamatban van-e, vagy nem
- **KeyBoardState keyBoardState**: a Robot osztály számára teszi elérhetővé a billentyűparancsokat, amik az irányításban és az akadályok (ragacs, olaj) lehelyezésében játszanak fontos szerepet.
- **static const int targetFrameTime**: a frissítések gyakoriságának felső határának megszabására használjuk, értéke 16 (kb 60fps lesz a maximum)

· Metódusok

- **addObstacle(GameObject go)**: kitesz a pályára egy újabb akadályt. Az akadály lehet ragacs vagy olajfolt.
- **getGameObjects()**: az összes GameObjectet tartalmazó listát adja vissza.
- **getRobots()**: visszaadja a Robotokat, listában tárolva.
- **getRemainingTime()**: a **gameTime** és **gameStarted** attribútumok felhasználásával kiszámolja a hátralévő időt egy versenyből.
- **Draw (Graphics g)**: ez a metódus felel a játékban megjelenő összes objektum kirajzolásáért.
- **Init()**: a játék során használt osztályok példányosítása, inicializálása.
- **setMaxGameTime(int t)**: a játékidő megadásáért felelős metódus. Ezzel állítjuk be a **gameTime** attribútumot.
- **Update()**: frissíti az összes játékban szereplő objektumot.
- **Start()**: ezzel a metódussal indítjuk el, valamint indítjuk újra a versenyt
- **Pause()**: a játék szüneteltetéséért felelős metódus. Például akkor hívódik meg, ha valamelyik játékos az ‘Esc’ gomb megnyomásával megszakítja a versenyt és belép a játék belső menüjébe.
- **addGameEndedListener(GameEndedListener gel)**: esemény, ami a játék befejezésekor hívódik meg. Az eseményre való feliratkozás a lehető leghamarabb történik meg, lehetőleg a Game osztály példányosítása után azonnal.

4.3.2 GameObject

- **Felelősség**

A GameObject interfész deklarálja a játékobjektumok közös függvényeit, amit majd az implementáló osztályok definiálnak. Lehetővé teszi, hogy a játékobjektumokat egy közös listában tudjuk tárolni.

- **Ősosztályok**
  - Nincs

- **Metódusok**
  - **void CollisionWithRobot(Robot r):** akkor hívódik meg, amikor a robot ütközik egy objektummal. Az objektumtól függően más-más történik: pl. ragacs esetén a robot sebessége megfeleződik.
  - **Draw (Graphics g):** az adott objektum kirajzolását végző metódus
  - **Update (Game g):** az adott objektum frissítését végző metódus

4.3.3 Glue

- **Felelősség**

Az osztály feladata a ragacsfoltok képének, helyzetének és kiterjedésének tárolása, valamint a ragacsfoltok frissítése és kirajzolása is.

- **Ősosztályok**
  - Obstacle

- **Interfészek**
  - Nincs

- **Attribútumok**
  - **static const double Radius:** a ragacs kiterjedése, amit az ütközések vizsgálatakor használunk
  - **static const string glueImage:** a ragacsfolthoz tartozó kép

- **Metódusok**
  - Nincs, az absztrakt osztály metódusait definiálja felül.

4.3.4 Map

- **Felelősség**

Tárolja a pálya képét és a pályát meghatározó maskot. Utóbbi segítségével állapítja meg, hogy az adott pozíció a pálya területén található-e.

- **Ősosztályok**

- Nincs

- **Interfészek**

- GameObejct

- **Attribútumok**

- **static const string outOfTrackMaskFile:** a pályához tartozó maszk képe
- **static const string mapImage:** a pálya képe

- **Metódusok**

- **boolean isOutOfTrack(Vector2D pos)**: megállapítja, hogy az adott koordinátával rendelkező objektum (pl.: robot, vagy akadály) a pályán kívül van-e.

4.3.5 Obstacle

- **Felelősség**

Az Obstacle absztrakt osztály tartalmazza az akadályok közös attribútumait és metódusát. Az akadályok (Glue és Oil) őse. Az osztály fő feladata az ütközések detektálása egy robot és egy akadály között.

- **Ősosztályok**

- Nincs

- **Interfészek**

- GameObject

- **Attribútumok**

- **int radius**: az akadály kiterjedése, amit az ütközések vizsgálatakor használunk
- **Vector2D position**: az akadály helyzete, pozíciója.

- **Metódusok**

- **boolean CollisionDetectWithRobot(Robot r):** a metódus feladata megállapítani, hogy az adott akadály ütközik-e a paraméterül kapott robottal.

4.3.6 Oil

- **Felelősség**

Az osztály feladata az olajfoltok képének, helyzetének és kiterjedésének tárolása, valamint az olajfoltok frissítése és kirajzolása is.

- **Ősosztályok**

- Obstacle

- **Interfészek**

- Nincs

- **Attribútumok**

- **static const double Radius:** az olajfolt kiterjedése, amit az ütközések vizsgálatakor használunk
- **static const oilImage**: az olajfolthoz tartozó kép

- **Metódusok**

- Nincs, az absztrakt osztály metódusait definiálja felül.

4.3.7 Robot

- **Felelősség**

Az osztály feladata a játékos által irányított robot legfontosabb paramétereinek a tárolása. Ilyen például a robot pozíciója, sebessége, vagy a megtett távolság. Ezen kívül az is a feladata, hogy frissítse és kirajzolja a robotot.

- **Ősosztályok**

- Nincs.

- **Interfészek**

- GameObject

- **Attribútumok**

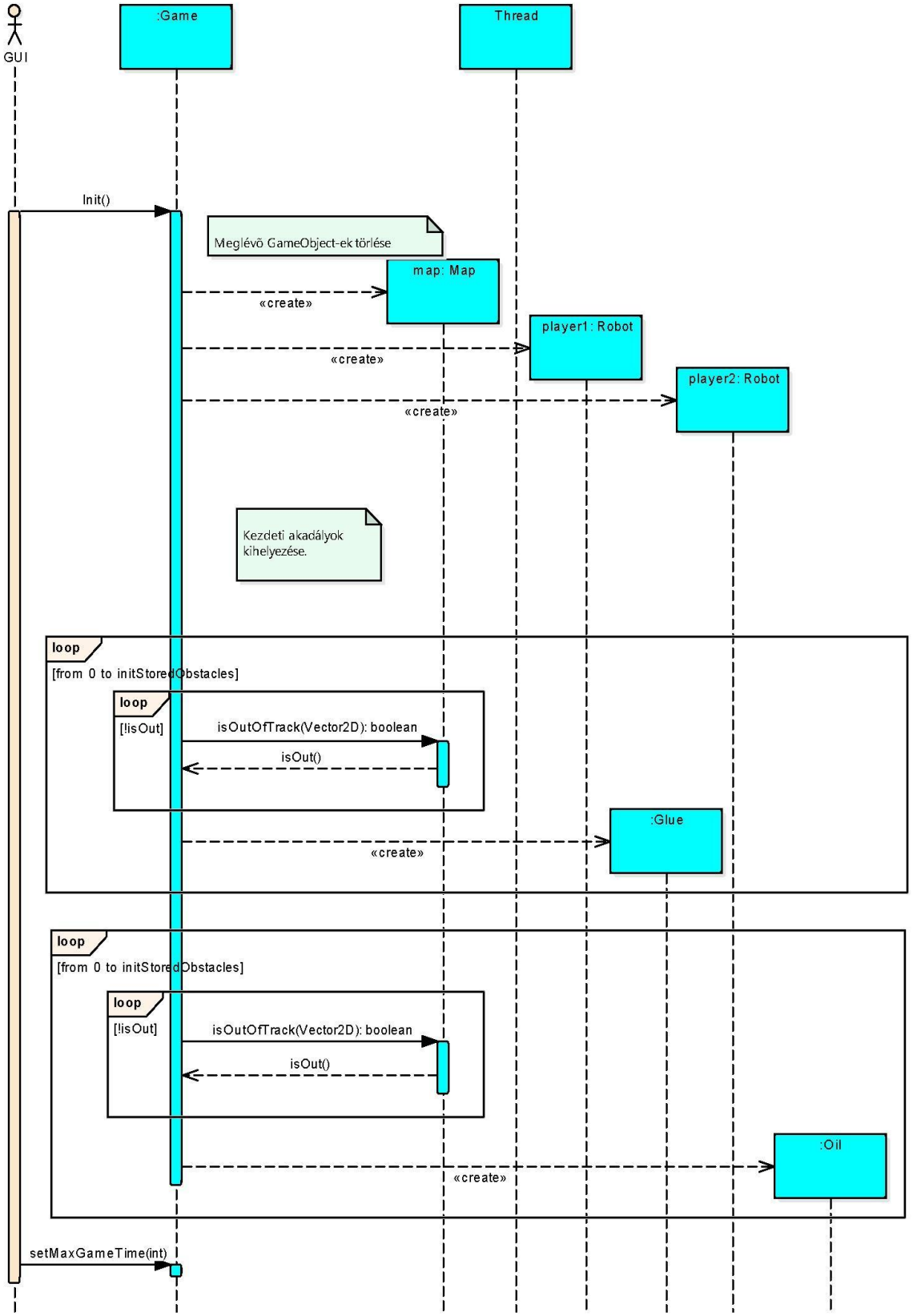
- **Hashtable<Control,int> playerControlKeys:** a játékos által használt gombok, amikkel a robotját tudja irányítani.
- **boolean controllable:** megadja, hogy a robot éppen irányítható-e, vagy sem. Egy robot akkor válik irányíthatatlanná, ha olajfoltra ugrik.
- **Vector2D distance:** a játékos által megtett távolság a verseny megkezdésétől számítva.
- **Vector2D position:** a robot helyzete, pozíciója.
- **Vector2D speed:** a robot pillanatnyi sebessége.
- **int storedOil()**: a még lehelyezhető olajfoltok száma
- **int storedGlue()**: a még lehelyezhető ragacsfoltok száma
- **static const int initStoredObstacles()**: a játékos vezérelte robot által lehelyezhető akadályok (előre beállított) száma
- **static const Hashtable<Control,int> player1ControlKeys:** az 1-es játékos által használt gombok, amikkel a robotját tudja irányítani
- **static const Hashtable<Control,int> player2ControlKeys:** a 2-es játékos által használt gombok, amikkel a robotját tudja irányítani
- **static const string player1RobotImage:** az 1-es játékos robotjának a képe
- **static const string player2RobotImage:** a 2-es játékos robotjának a képe
- **static const double Radius:** a robot kiterjedése, amit az ütközések vizsgálatakor használunk

- **Metódusok**

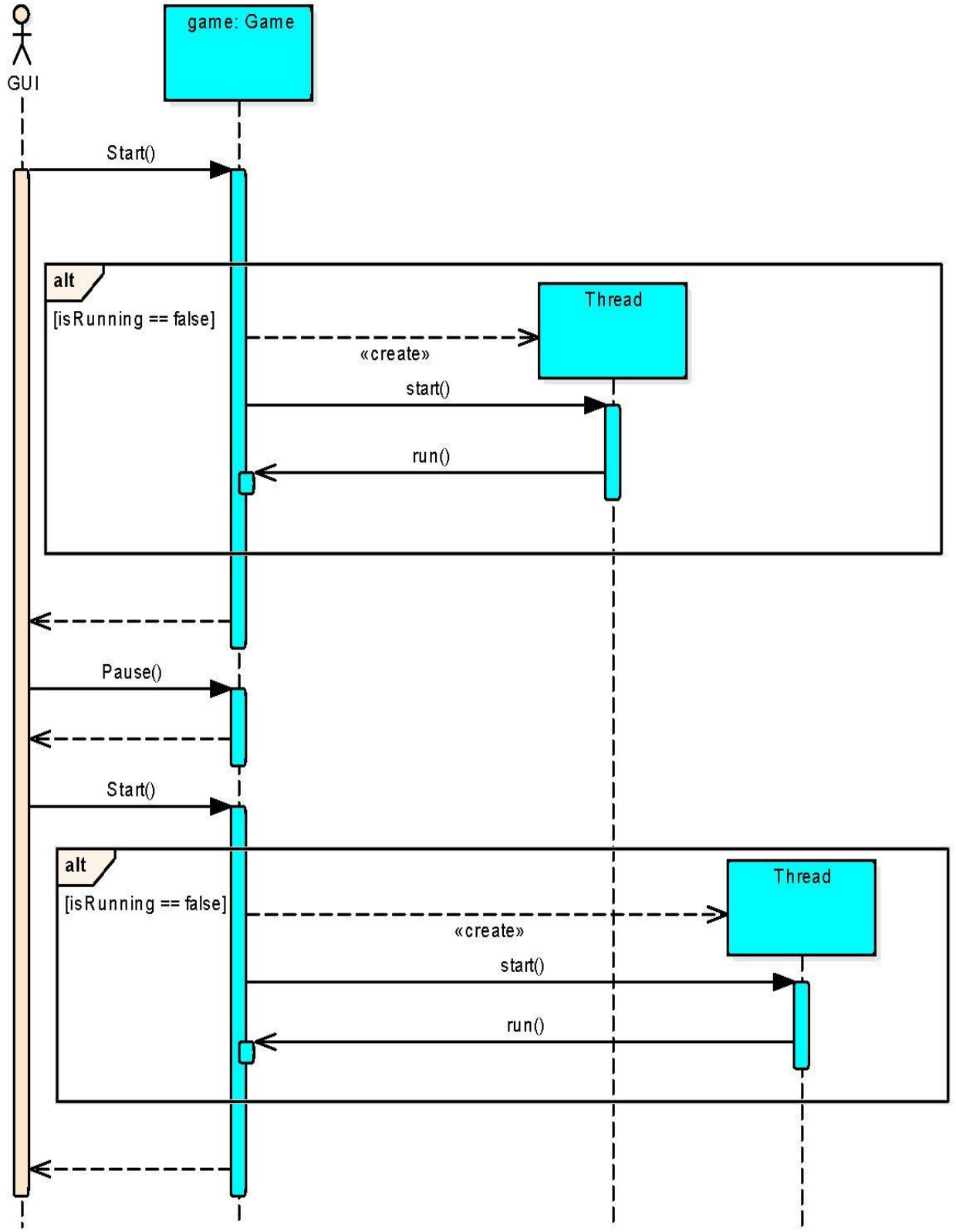
- **boolean isAlive():** azt adja vissza, hogy a robot játékban van-e még, vagy ütközés/pálya elhagyás miatt kiesett-e már.
- **void killHim():** a robot elpusztításáért felelős metódus. Akkor kerül meghívásra, ha összeütközik a másik robottal, vagy elhagyja a pályát.
- **boolean onTheGround():** megadja, hogy a robot éppen a földön van-e, vagy egy ugrás során a levegőben tartózkodik.

4.4 Szekvencia diagramok

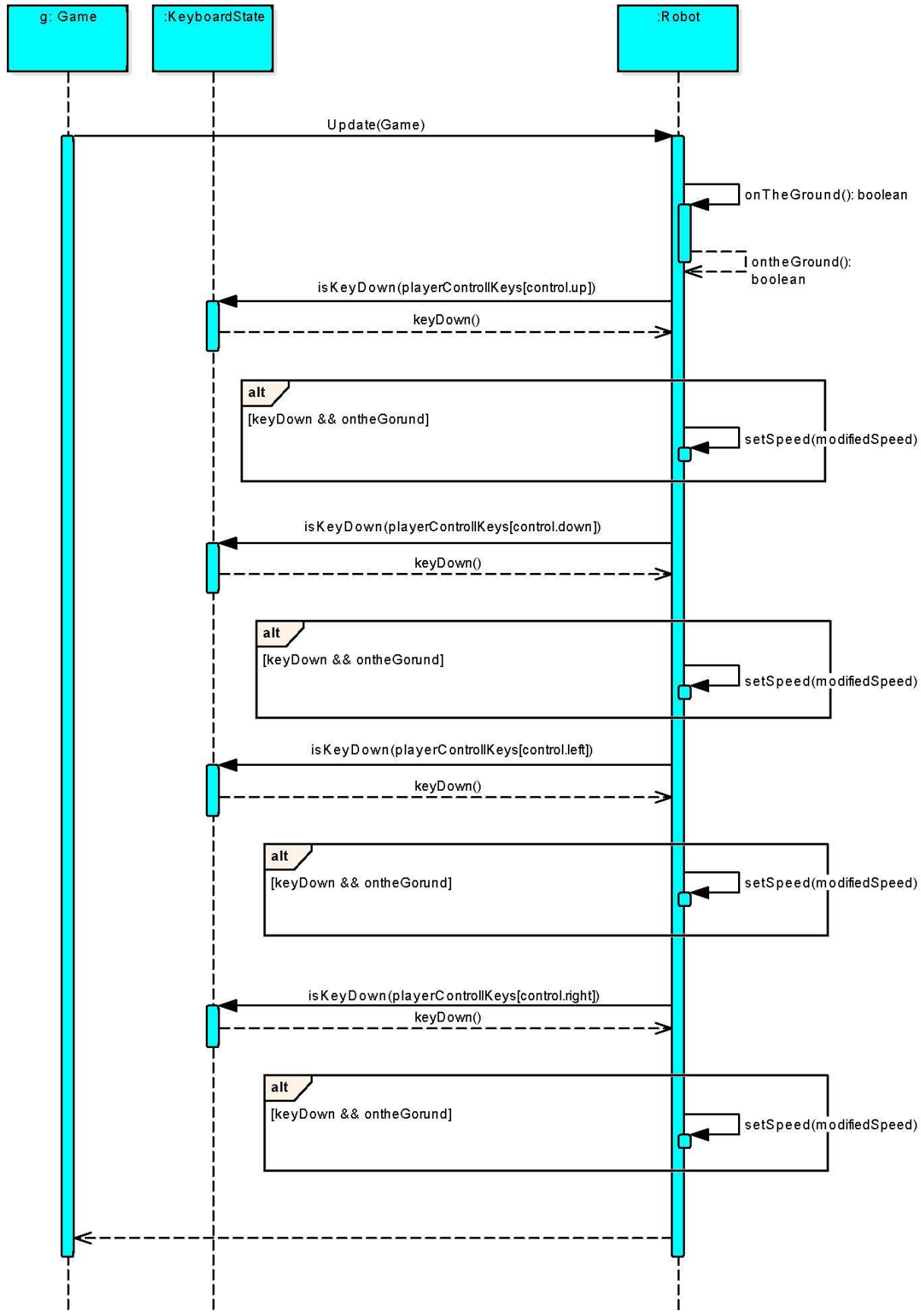
A játék inicializálását és indítását bemutató szekvencia diagram:



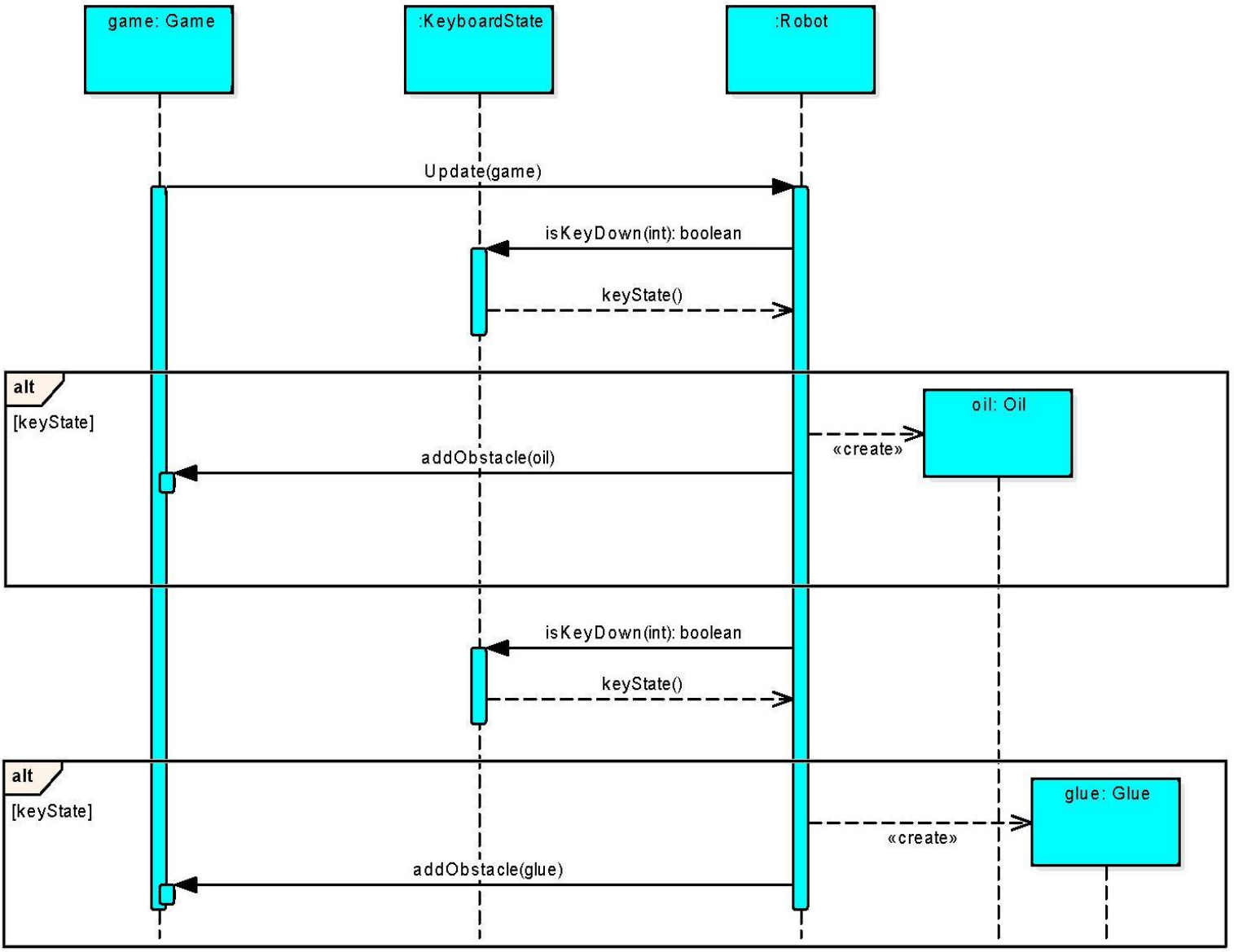
A játék indítását, szüneteltetését, majd folytatását bemutató szekvencia diagram:



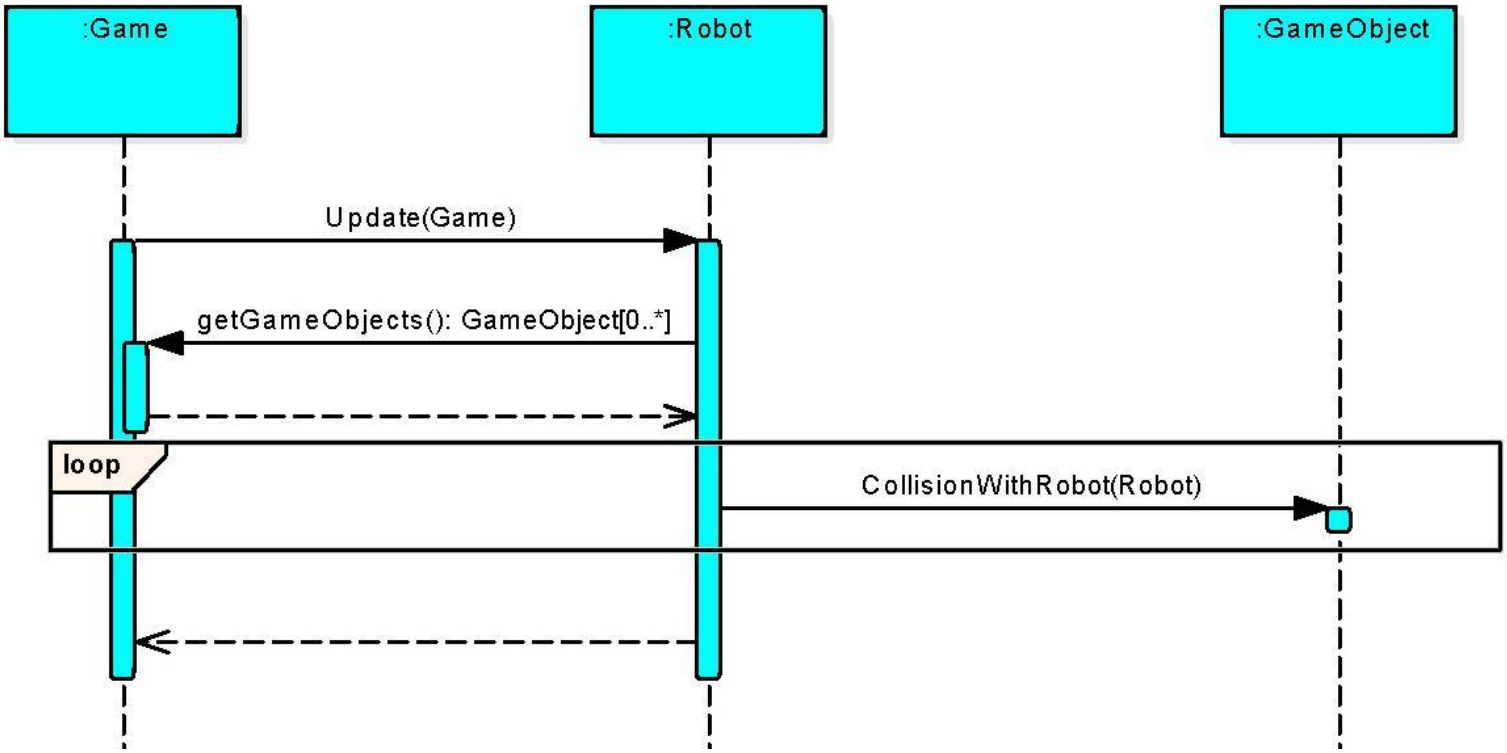
Az alábbi szekvencia diagramon látható, hogyan történik a robotok irányítása:



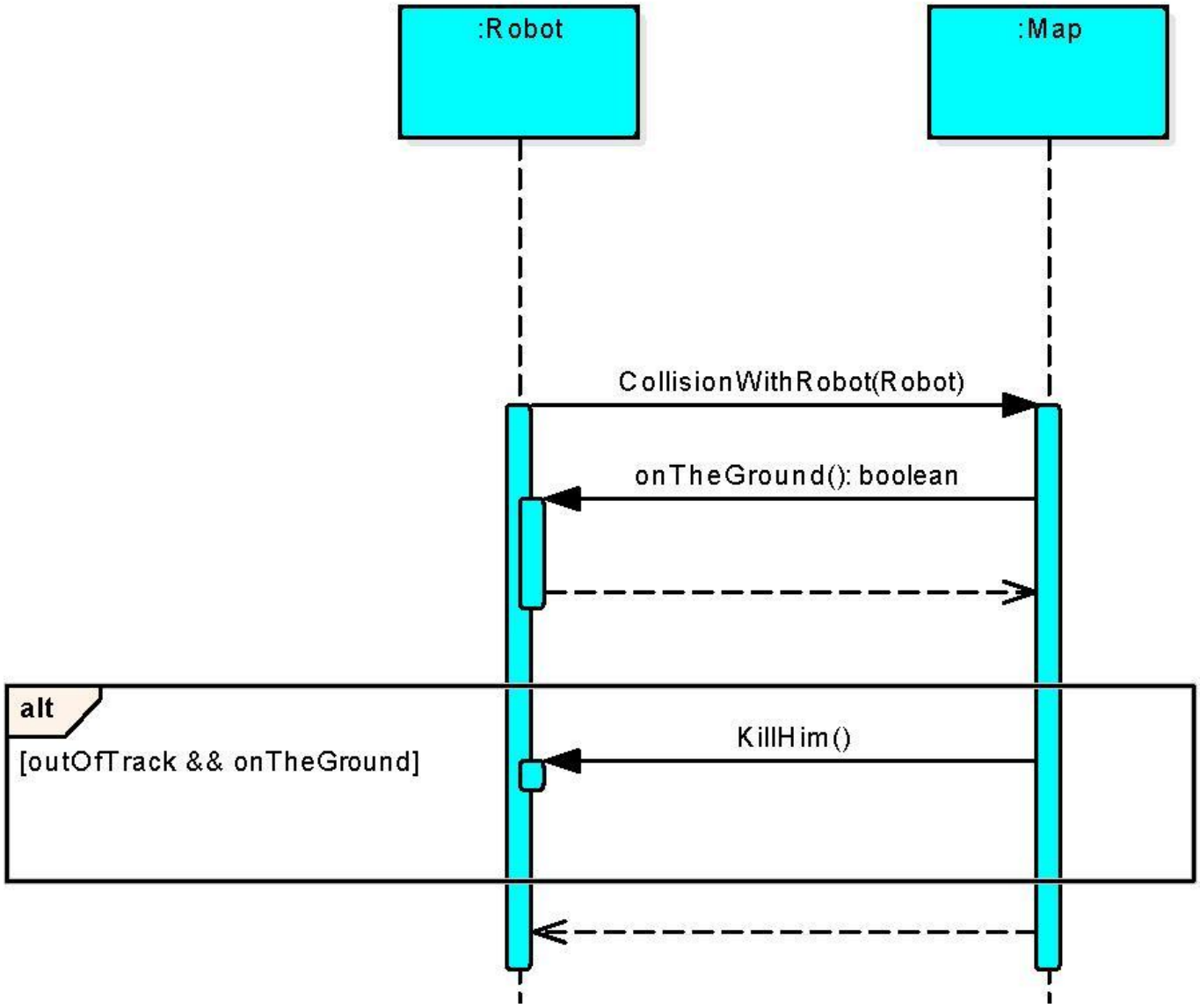
A következő szekvencia diagram mutatja be, hogyan tudják a játékosok lerakni az olaj- és ragacsfoltokat a pályára:



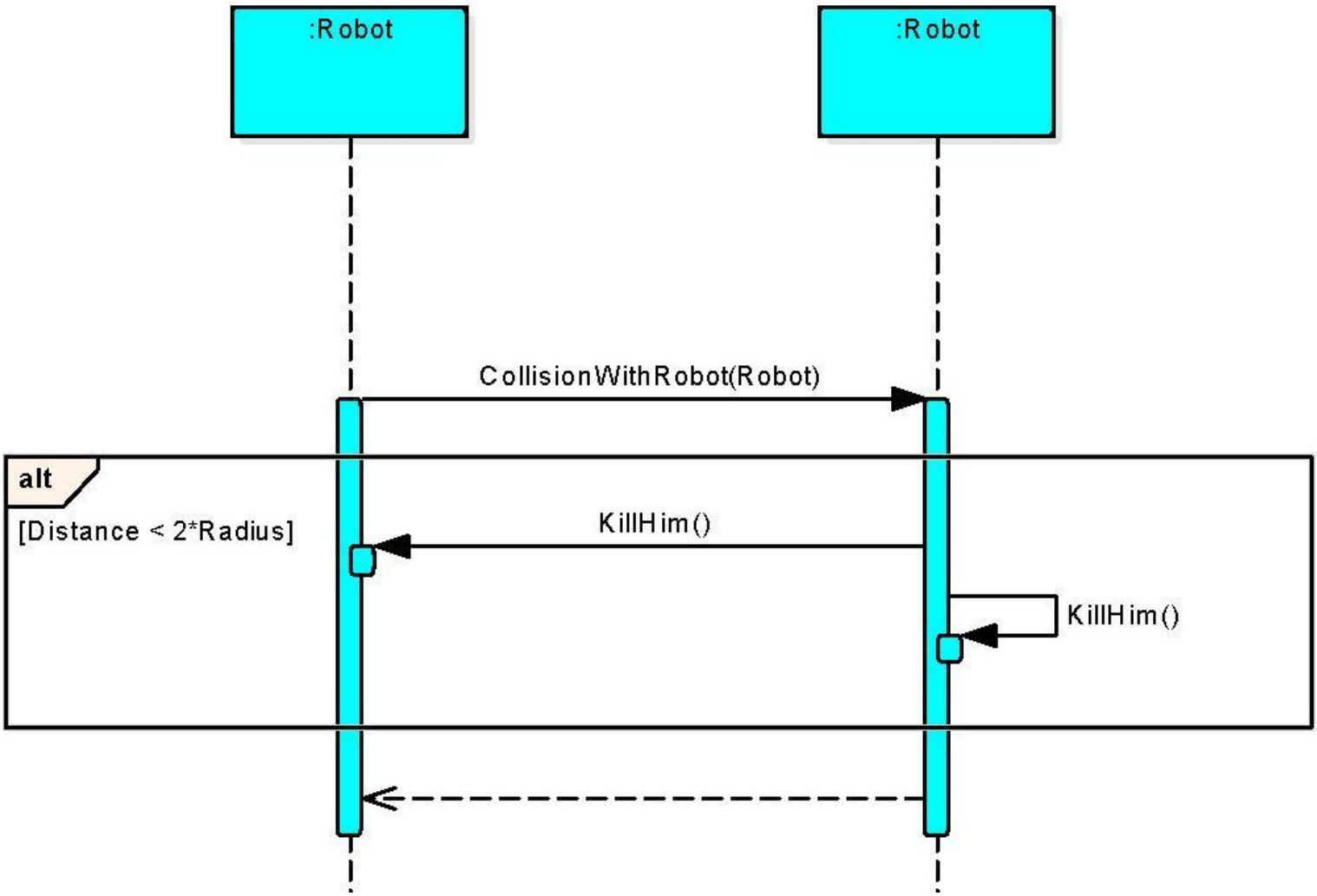
A játékban az ütközéseket bemutató általános szekvencia diagram:



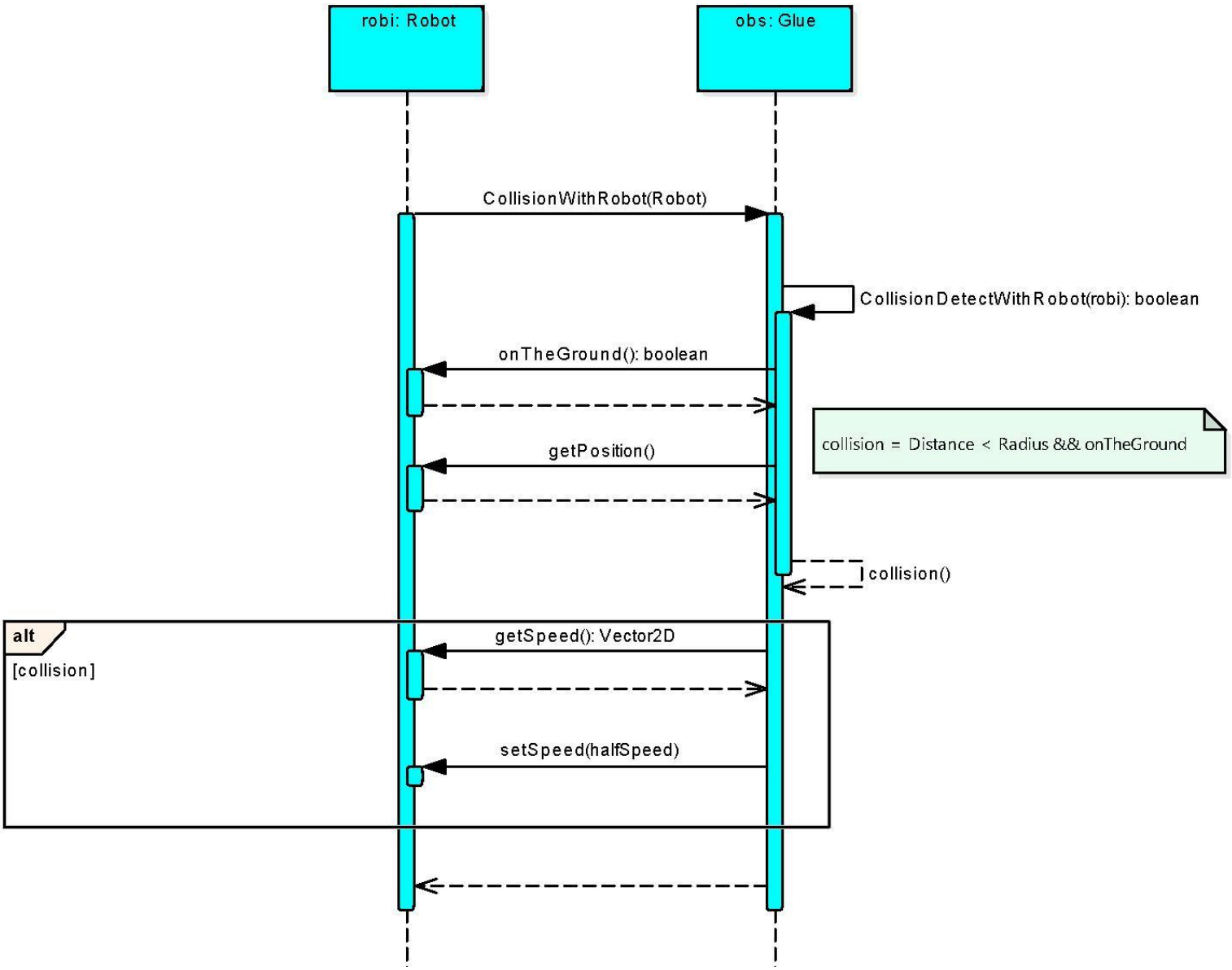
A játékban a robot helyzetét a pályához képest a következőképpen vizsgáljuk:



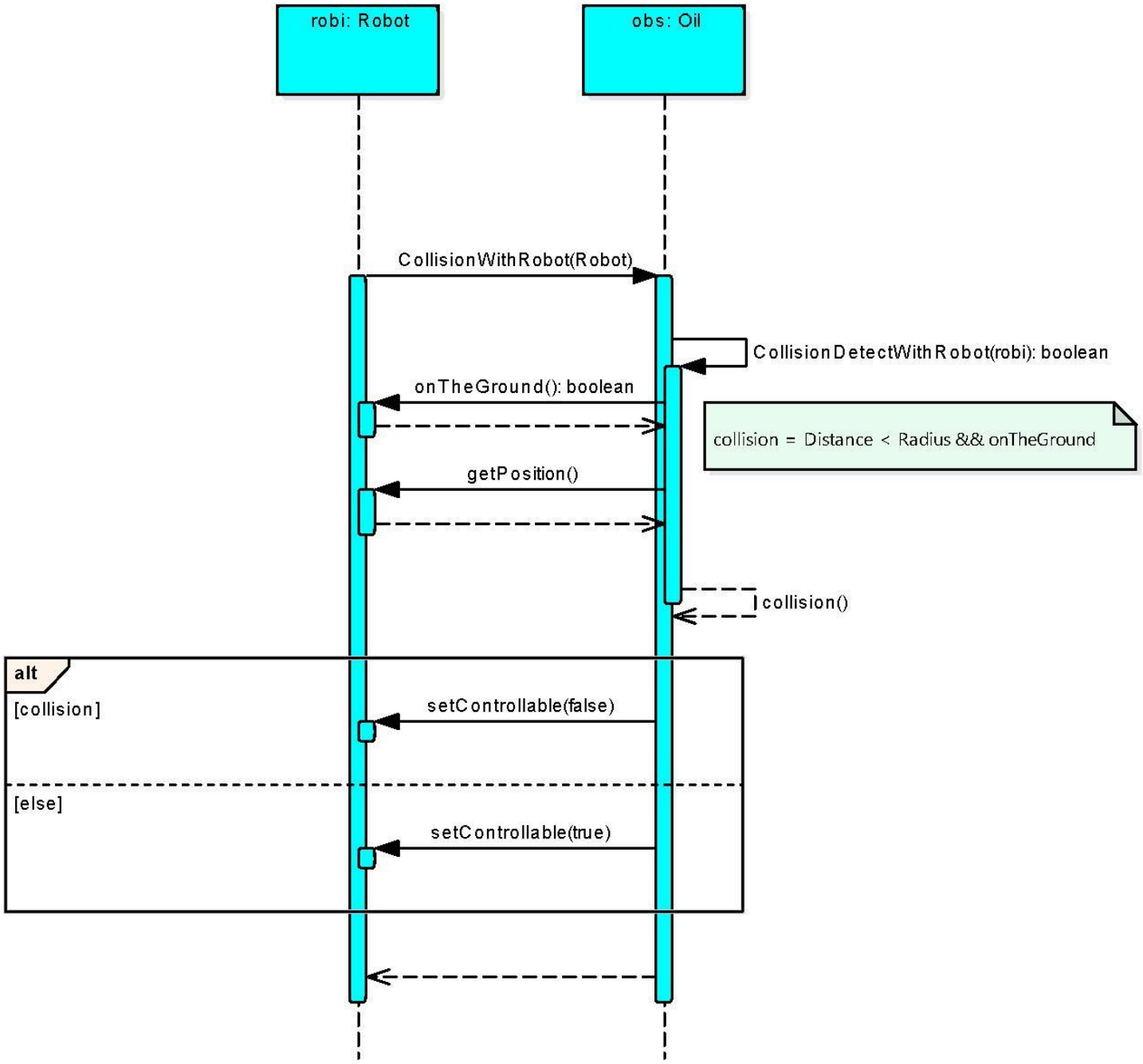
Az alábbi szekvencia diagram mutatja be, mi történik akkor, amikor a két robot összeütközik egymással:



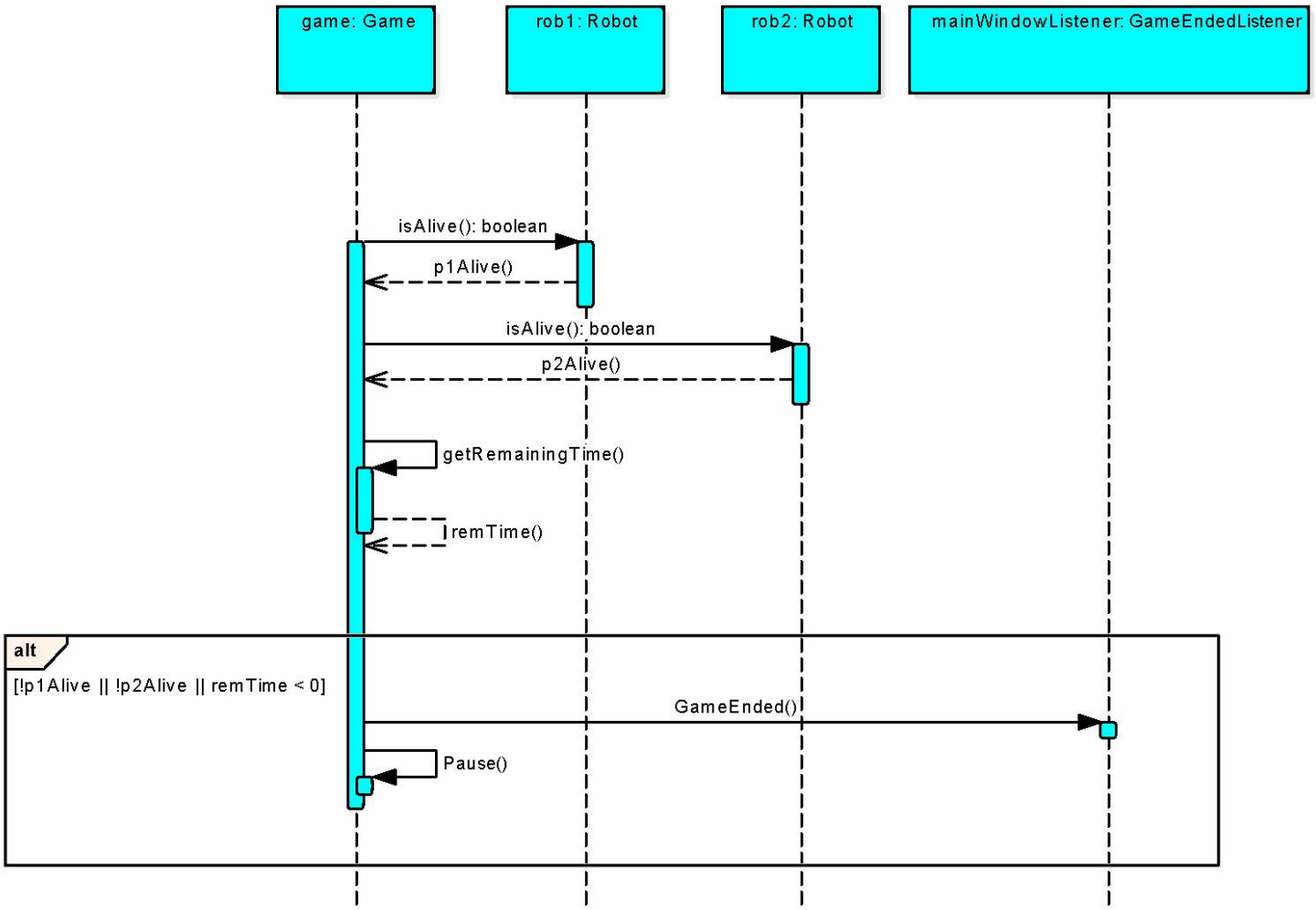
A következő szekvencia diagram mutatja be, hogy mi történik, amikor az egyik robot ráugrik egy ragacsfoltra:



A következő szekvencia diagram mutatja be, hogy mi történik, amikor az egyik robot ráugrik egy olajfoltra:

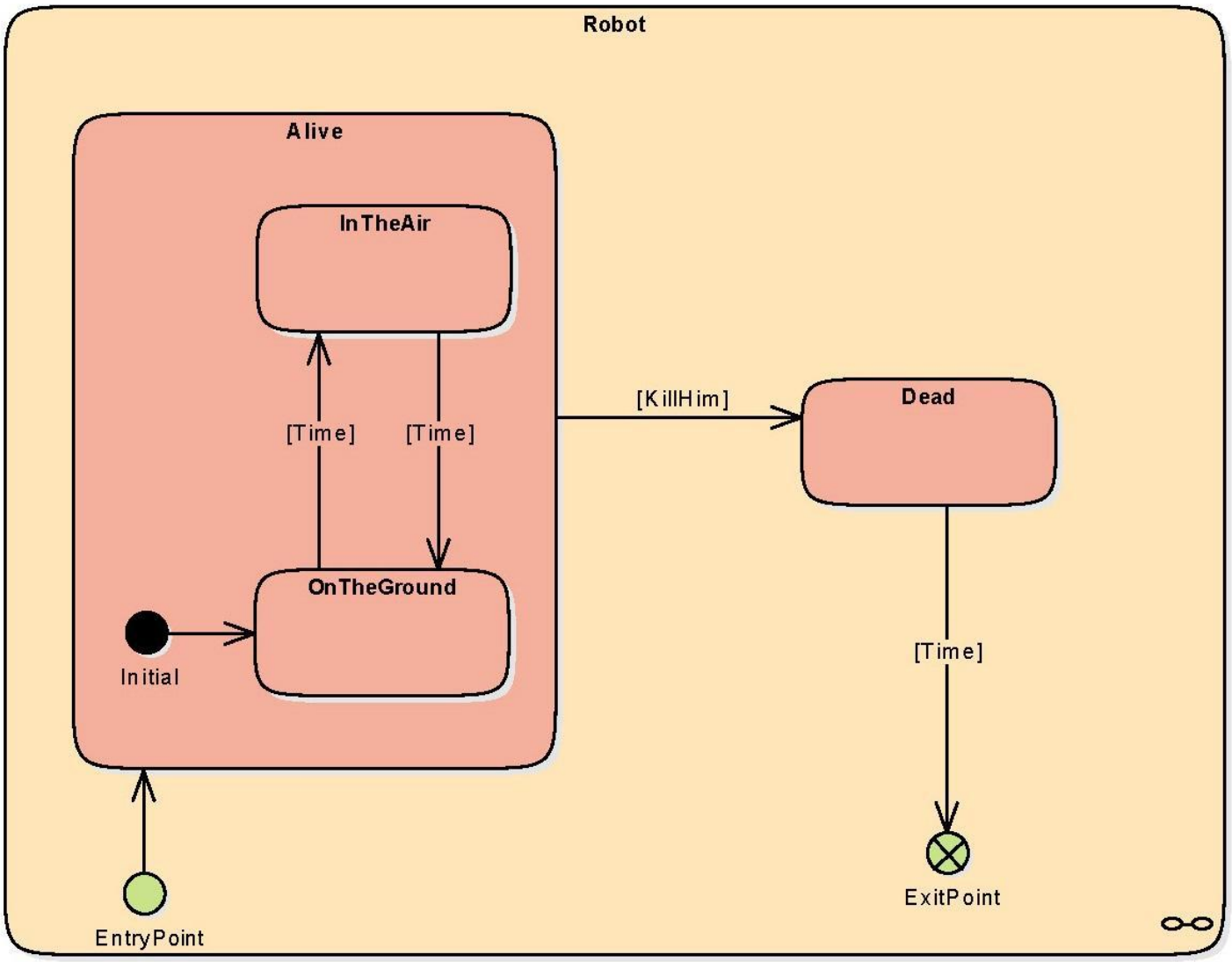


A játék befejezését szemléltető szekvencia diagram:



4.5 State-chartok

A robot életciklusát bemutató state-chart:



4.6 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2015.03.04. 8:00	2 óra	Dusik Kachichian Pohubi Szendi Zelenák	Konzultáció az R-ben
2015.03.05. 10:00	2 óra	Dusik Kachichian Pohubi Szendi Zelenák	Megbeszélés az E-ben. A jelzett hibák kijavításának megtervezése.
2015.03.05. 20:00	2 óra	Dusik Kachichian Pohubi Szendi Zelenák	A hibák tényleges kijavítása, egyeztetés Hangouts-on.
2015.03.06. 14:00	1,5 óra	Dusik Kachichian Pohubi Szendi Zelenák	Dokumentáció véglegesítése