Összefoglalás

15 – just_another_team

Konzulens:
László Zoltán

Csapattagok

Ferencz Endre	PEBU1F	endre.ferencz@gmail.com
Orovecz Ferenc	B0HUPW	feri.orovecz@gmail.com
Brucker Amanda Mária	UHZMDX	mandi.brucker@gmail.com
Pocsai Csaba	BOYKMJ	pocsaycsaba@freemail.hu

2010.02.16

2. Követelmény, projekt, funkcionalitás

2.1 Követelmény definíció

2.1.1 A program célja, alapvető feladatai

A program egy játék, amelyben a felhasználó bankrablók gépkocsiját irányítja egy városban, azzal a céllal, hogy eljusson a kirabolt banktól a rejtekhelyre, anélkül, hogy közben valamivel ütközne, vagy elkapná egy rendőr.

A fejlesztőcsapatunk célja egy, a specifikáció szerint elkészített, jól működő program létrehozása, mely tökéletesen fut a „Rendszerkövetelmények” (2.1.3) pontban meghatározott környezetben.

Másodlagos célnak tekintjük azt, hogy a program valamilyen játékélményt is nyújtson a felhasználóknak.

2.1.2 A felhasználói felület

A program billentyűzettel és egérrel vezérelhető, valamint a lehetőségek szerint átlátható grafikus felülettel látjuk el.

2.1.3 Rendszerkövetelmények

A program futtatásához a felhasználónak rendelkeznie kell a Java Runtime Environment (JRE) 1.6.0 verziójú futtatási környezettel, és a JAVA JDK 1.6.0 fejlesztőeszközzel, továbbá a programot futtató gépnek meg kell felelnie a környezet rendszerigényének (Pentium 166MHz, 32MB RAM, 200 MB háttértár).

A grafikus megjelenítés bonyolultságának függvényében, a megfelelő játékélményhez szükség lehet magasabb szintű rendszerkövetelményekre is, de alapvető célunk, hogy a program fusson a labor számítógépein.

2.1.4 A szoftver fejlesztésével kapcsolatos alapkövetelmények, elvek, célok

Modellhűség: A fejlesztés teljes ideje alatt a csapatnak a lehető legjobban kell tartania magát a specifikációban meghatározottakhoz. A modell feladathoz való igazítása segít abban, hogy az esetleges specifikációváltozások, vagy új funkciók bevezetése minél kevesebb változást okozzon az addig megírt programban.

A modellezés legfőbb célja, hogy a követelményeket egy alacsonyabb absztrakciós szintre hozzuk, és növeljük a formalizáltságot. Azáltal, hogy mi, a fejlesztők, jobban átlátjuk a követelményeket, jobban tudunk igazodni a felhasználó igényeihez, valamint a megvalósítás szempontjából egyszerű, új funkciókat is javasolhatunk.

Továbbfejleszthetőség: A továbbfejlesztés többnyire elkerülhetetlen. Ahhoz, hogy a programunk versenyképes maradhasson, folyamatos tökéletesítéseket kell alkalmazni. Fontos szempont a fejlesztés közben, hogy a dokumentáció és a program felépítése lehetővé tegye az esetleges jövőbeli továbbfejlesztést.

Fontos továbbá, hogy a program kódja a többi fejlesztő számára is átlátható legyen, ezért elengedhetetlen a megfelelő, pontos és érthető programozói dokumentáció folyamatos írása a fejlesztés során.

Modularitás: A program egyes részeit, azaz moduljait, a lehető legnagyobb elkülönítéssel kell elkészíteni, hogy a tesztelés / hibakezelés könnyebbé váljon, és a kód rendszerezett, átlátható legyen.

Ugyanez következik a szoftver munka kiszámításának képletéből is, tehát a programot megfelelően részekre bontva optimalizálhatjuk a programozás folyamatát.

Számunkra a modularitást az objektum-orientáltság biztosítja.

Teljesítmény, optimalizálhatóság: Nagy hangsúlyt kell fektetni már tervezés közben is az optimalizálásra, a felesleges kódok kiiktatására, illetve egyszerűsítésére. Ezáltal a kész program a legalapvetőbb követelmények teljesítésével tökéletes futásra lesz képes.

A fejlesztés során a megfelelő funkciók implementálása a fő cél. A futás sebességét később szeretnénk optimalizálni megfelelő eszközökkel (code profiler), de ez nem jelenti azt, hogy a fejlesztés közben ezt a szempontot elhanyagolnánk.

Felhasználhatóság: A cél egy könnyen elsajátítható intuitív játék készítése, ahol az irányítás és a szabályok egyértelműek és könnyedén megtanulhatók.

Nem célunk, hogy a program a felhasználói kézikönyv elolvasása nélkül is könnyen használható legyen, de szeretnénk, ha az átlag felhasználó egyből átlátná a program kezelőfelületét.

Perzisztencia: A programnak fel kell jegyeznie futás, illetve tesztelés közben a fontos folyamatok tevékenységét, a változók értékeit és minden releváns adatot, mert egy esetleges meghibásodás esetén ezen feljegyzések alapján könnyen behatárolható a hiba helye és jellege.

Tervünk, hogy az összes osztály, ami a modellhez tartozik perzisztens legyen (itt szeretnénk felhasználni az XML által nyújtott előnyöket).

Modern technológiák használata: Két modern technológiára alapozzuk a fejlesztést: a RUP életciklus modell és az UML általános célú modellező nyelv. Ezek az eszközök megkönnyítik a fejlesztést, valamint a csoporton belüli kommunikációt.

Ezen két modern technológia mellett a fejlesztő környezet, a verziókezelő rendszer, valamint a kommunikációra felhasznált programok is nagyban segítik a közös munkát.

Minőségbiztosítás: A folyamatos teszteléssel és ellenőrzéssel, valamint az egymás munkájának nyomon követésével megpróbáljuk elérni azt, hogy minimálisra csökkentsük a hibák számát.

2.2 Projekt terv

2.2.1 A felhasznált segédeszközök

Csapatunk minden tagja a közismert Eclipse fejlesztőkörnyezetet részesítette előnyben, részben az eddigi tapasztalatok, részben a sok bővítmény miatt, melyek közül kiemelném a verziókezelés lehetőségét.

A modell megtervezésében nagy segítség a Visual Paradigm Community Edition UML diagramszerkesztő, mely hasznosságát nem csak az ábrák rajzolásában bizonyítja, hanem nagy segítség abban is, hogy az UML –lel megalkotott modellnek megfelelő kódot generáljunk.

A verziókezelést az Eclipse és a Subversion rendszerek párosításával értük el.

2.2.2 A projekt végrehajtásának lépései

Legfőbb mérföldkövek:

A fejlesztést az alapoknál kell kezdeni. A **szkeleton** egy fontos mérföldkő a fejlesztés során, mivel befolyásolhatja a végtermék minőségét, illetve annak kifejlesztéséhez szükséges időt. A szkeleton változat célja annak bizonyítása, hogy az objektum és dinamikus modellek a definiált feladat egy modelljét alkotják.

A **prototípus** program célja annak demonstrálása, hogy a program elkészült, helyesen működik, valamennyi feladatát teljesíti. A prototípuson már az osztályok közötti interakció ellenőrzése is magasabb szinten lehetséges.

A teljes (**grafikus**) változat a prototípustól csak a kezelői felület minőségében különbözik.

Ezen mérföldkövek teljesítése között is több iterációra bomlanak a lépések. Ezeket az iterációkat remekül tükrözik a beadási határidők:

#	*Határidő*	*Feladatok*
1	febr.10.	14 h - csapatok regisztrációja
2	febr. 18.	Követelmény, projekt, funkcionalitás
3	febr. 25.	Analízis modell kidolgozása 1.
4	márc. 4.	Analízis modell kidolgozása 2.
5	márc. 11.	Szkeleton tervezése
6	márc. 18.	Szkeleton
7	márc. 25	Prototípus koncepciója
8	ápr. 1.	Részletes tervek
9	ápr. 8.	
10	ápr. 15.	Prototípus
11	ápr. 22.	Grafikus felület specifikációja
12	ápr. 29.	
13	máj. 6.	Grafikus változat
14	máj. 13.	Összefoglalás

1. Táblázat: Lépések

2.2.3 Beadások

A dokumentációt minden héten nyomtatott formában adjuk be. A fontosabb mérföldkövek megfelelőségét a konzultációs időpontban mutatjuk be, valamint határidőig mindig feltöltjük a megadott helyre.

2.2.4 Feladatkörök

Az alábbi táblázatban összefoglaltuk az egyes csapattagok főbb feladatait. A táblázat csak általános elveket fogalmaz meg, mivel munkánk fontos része az, hogy egymást pozitív irányú kritikákkal és minden más módon segítsük. A felsoroltakon kívül mindenkinek feladata az, hogy a beadandó anyagot átnézze és a hibákat / észrevételeket jelezze.

Továbbá fontos szempont az is, hogy mindenki egyformán vegye ki a részét a munkából.

Név	*Feladatok*
Brucker Amanda Mária	dokumentáció, tesztelés, munka összehangolása, grafikus felület
Ferencz Endre	csapatvezetés, kódírás, (business) modell megtervezése
Orovecz Ferenc	dokumentáció, tesztelés, beadandó anyagok véglegesítése
Pocsai Csaba	dokumentáció, kódírás, diagramok szerkesztése

2. Táblázat: Feladatkörök

2.2.5 Csapaton belüli kommunikáció

Egy ilyen összetett projekt sikerességének szempontjából fontos a csapattagok közötti kommunikáció. Elsődleges kommunikációs formánk az elő szó. Lényeges, hogy minden héten összeüljünk és megbeszéljük, elosszuk a szükséges teendőket, még akkor is, ha erre csak kevés szabad idő áll rendelkezésre.

Elektronikus üzeneteken keresztül hangoljuk össze a további teendőket. Itt még megemlíteném a verziókezelést, ami a kódírásban és a dokumentáció készítésében is nélkülözhetetlen.

2.2.6 Kockázatelemzés

A projekt kezdetekor érdemes megvizsgálni a kockázatokat, és ennek megfelelően lépéseket hozni. (Például azért választottunk megfelelő verziókezelést, hogy egy esetleges meghibásodás során enyhítsük a következményeket.)

A valószínűségeket öt kategóriába soroljuk: nagyon alacsony, alacsony, közepes, nagy, nagyon nagy. Jelentésük értelemszerű.

A hatásokat négy csoportba oszthatjuk: végzetes, komoly, tolerálható, jelentéktelen.

Kockázat neve	*Valószínűsége*	*Hatása*	*Ellenlépések*
Specifikáció változása	nagy	tolerálható	Jó (business) modell tervezése.
Javítandó heti beadandó	közepes	közepes	Minden beadandót ellenőrizünk.
Sikertelen heti beadandó	alacsony	komoly	Minden beadandót ellenőrizünk.
Csapattag kiválása	nagyon alacsony	komoly	Megfelelő belső kommunikáció.
Csapattag személyes problémája	közepes	közepes	A fontos feladatoknak több felelőse van.
Szoftver hiba	közepes	jelentéktelen - végzetes	Biztonsági mentés. (verziókezelés)
Hardver meghibásodása	alacsony	jelentéktelen - végzetes	Biztonsági mentés. (verziókezelés)
Határidő elszalasztása	alacsony	komoly	Összehangolt munka.
Tanácskozásról hiányzás	alacsony	tolerálható	Egy csapattag mindig legyen jelen.

3. Táblázat: Kockázatelemzés

2.3 Feladatleírás

2.3.1 Eredeti feladatkiírás:

Egy kis német város, *Verkehrsmeldungen am Uml* útjain járművek közlekednek. Minden jármű automatikusan az úton tartja magát, nem ütközik más járművekkel és a közlekedési szabályoknak engedelmeskedik (pl. piros lámpánál, stop táblánál egy időre megáll).

Elágazásoknál a járművek véletlenszerűen választanak a lehetőségek között. A járművek folyamatosan lépnek ki és be a városhatárnál. A járművek különböző sebességgel közlekedhetnek, de nem előzik meg egymást valamint az ütközéselkerülésből adódóan hátulról nem ütköznek, pusztán lelassulnak az előttük lévő jármű sebességére.

Az utak egyirányúak és minden szabály ki van táblázva (nincs jobbkezes kereszteződés).

A bankrablók speciálisan tuningolt autója kivétel a szabály alól, ők szabadon közlekedhetnek a városban, minden szabályt áthágva. Miközben megpróbálnak eljutni a banktól a rejtekhelyig, ügyelni kell, hogy ne ütközzenek más autóval és az üldöző rendőr se érje el őket.

2.3.2 Részletes feladatleírás

Az általunk készítendő alkalmazás célja a játékos szórakoztatás. Ez egy valós idejű játék formájában kel életre, melyben a játékos egy bankrablásból menekülő autót irányít *Verkehrsmeldungen am Uml* nevű város kalandos útjain. *Verkehrsmeldungen am Uml* egy különleges város, mivel napról napra változik a város térképe és sosem apadnak ki a könnyen kirabolható bankok. A felhasználó feladata az, hogy elvezesse a rablók kocsiját a rejtekhelyig anélkül, hogy az üldöző rendőrök utolérnék, vagy túl sokszor ütközne valamivel. Az utóbbi két esetben a játékos veszít, de lehetősége van új pályán, új játékot kezdeni.

A városban csak egyirányú utak vannak, de ezek keresztezhetik is egymást egy szinten (kereszteződés), illetve elmehetnek egymás fölött / alatt. Egy kereszteződéshez legfeljebb négy kijövő és négy bemenő út tartozhat (a komplexebbek visszavezethetők erre a két alapesetre, mint például a körforgalom, mely megfelelő számú hármas szintű kereszteződésből áll). Ezekben mindig rögzített az elsőbbségi sorrend, vagy jelzőlámpával vannak ellátva. Így a létrehozható, érdemben különböző pályák száma elegendően magas. Az utakon az előzés lehetetlen, mivel a magas telekárak miatt olyan keskenyre építették őket, hogy nem fér el egymás mellett két autó. A város központjában sétálóutcák is találhatók, ahová az átlag autósok nem mehetnek be, csak a rendőrök, és ők is csak akkor, ha használják a kék-piros megkülönböztető jelzést.

A kereszteződések tehát két fajta forgalomirányítási módszer szerint működnek: forgalomirányító lámpák és az elsőbbség megadására figyelmeztető táblák segítségével. Ezek működése megfelel a való életben megszokottaknak. A kereszteződésbe csak akkor hajthatnak be az autók (a rablók autóján kívül), ha az útiránynak megfelelő úthoz ütközés nélkül el tudnak jutni.

A program alapállapotban (pl. indítás után) a város szürke hétköznapjait mutatja, melyen látszanak az éppen közlekedő autók és néhány közlekedési rendőr. Az autók a város szélén lépnek be különböző sebességgel és gyakorisággal, valamint el is hagyhatják a várost, ha arra viszi őket a kötelesség. A közlekedők német precizitással, maradéktalanul betartják a közlekedési szabályokat, azaz piros lámpánál megállnak, stop táblánál elsőbbséget adnak, nem hajtanak be sétálóutcába, mindig az autójuknak megfelelő, a megengedett legnagyobb sebességgel haladnak, de ha utolérnek valakit, lassítanak. A német vezetők másik érdekes tulajdonsága az, hogy az éppen aktuális kedélyállapotuktól függően választanak útirányt, azaz sosem lehet megjósolni előre, hogy melyik irányba fordulnak.

Ebbe a rendszerbe csöppen bele a játékos, aki a banktól (a bankok helye adott, a pályával együtt) indulva el kell jusson a rejtekhelyre. Minden kereszteződésnél a játékosnak választási lehetősége van az útirányt tekintve, valamint várakozhat is, ha a helyzet úgy adja. Mivel a rablók autója speciálisan tuningolt, számára nem kötelező érvényűek a szabályok és a sebességét is szabadon változtathatja az autó fizikai határain belül. Az utak kötelező haladási iránya viszont rá is érvényes. A tuning nem csak a szabályok ellen véd: ez az autó kibír néhány ütközést átlagos járművekkel, de ilyenkor bizonyos valószínűséggel el is romolhat, mely sajnos a játék végét jelenti. A rendőrautók sérthetetlenek, ezért a rendőrrel való ütközés mindig végzetesnek bizonyul a játékos számára. Csak abban az esetben sikerülhet az elrejtőzés, ha a rejtekhely utcájában az érkezés pillanatában éppen nincs rendőr, aki megfigyelhetné őket.

A felhasználó több pályán bizonyíthatja ügyességét. Eredetileg csak egy pálya áll rendelkezésre, viszont ha azt teljesíti a játékos, kap egy új pályát, ami már egy kicsit nehezebb, több rendőr van, valamint a forgalom is nagyobb lehet.

A rendőrök véletlenszerűen közlekednek a pályán. Mivel a német rendőrség gyorsan reagál az autós üldözésekre, az idő előrehaladtával egyre több rendőr érkezik a pályára.

Amennyiben a rabló a bázisra ért a zsákmánnyal együtt, a játékos nyert, és továbbmehet a következő pályára, azonban a játéknak nem csak a gyors menekülés lehet a célja. Az ügyes játékosok már azzal is próbálkozhatnak, hogy minél több ideig képesek legyenek menekülni az egyre növekvő rendőrlétszám elől és csak később meneküljenek a bázisra. Ezen időkből készül egy rangsor a legjobb játékosokból.

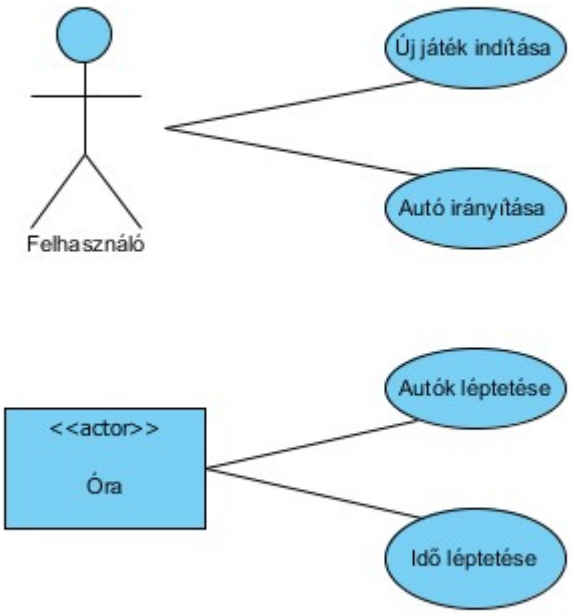
2.4 Szótár

Fogalom	*Meghatározás*
Felhasználó, játékos	A számítógépet használó és irányító személy.
A program alapállapota	Új játék indítása utáni állapot. Ez az állapot áll be az alkalmazás indításakor is.
Város	Bejárható játékterület.
Pálya	A város egy adott szerkezete (manifesztációja), mely az adott játék során nem változik.
Jármű	A városban mozgó elemek.
Jármű sebessége	A jármű azon tulajdonsága, mely meghatározza a pályán való közlekedés gyorsaságát (egységnyi idő alatt megadott útszakaszt).
Utolérés	Az a helyzet, amikor az egyik autó nem a saját sebességének megfelelően közlekedik annak érdekében, hogy elkerülje az ütközést.
Várakozás	A jármű azon állapota, amikor sebessége nulla.
Bankrabló	Ez a felhasználó által irányított jármű vezetője.
Ütközés	Két jármű ugyanazon a helyen van, ugyanabban az időpontban.
Út, utca	Ezen mozognak a járművek.
Egyirányú út/utca	Olyan út/utca, amelyre csak egy irányból lehetséges a behajtás.
Haladási irány	Az út egyirányúságával megegyező irány.
Sétáló utca	Olyan egyirányú utca, ahova csak a rabló és az őt üldöző rendőrök hajthatnak be.
Útirány	Egy kereszteződésből egy másik kereszteződésbe való utat kiválasztó fogalom.
Kereszteződés	Az utak azonos szinten való találkozása.
Kereszteződés szintje	A négy irányból egy kereszteződésen hány „foglalt”.
Városhatár	A város széle, ahol be- és kiléphetnek a számítógép által irányított járművek.
Közlekedési szabályok	Az elágazásokban ezek szerint mozognak a járművek a bankrablókét kivéve.
Lámpa	Kereszteződésben a járművek közlekedési szabályok szerinti továbbhaladását tiltó vagy engedélyező periodikusán működő berendezés.
Piros lámpa	A lámpa azon állapota, amikor a szóban lévő útról tiltja a kereszteződésbe való behajtást.
Elsőbbségi sorrend	Kereszteződésben a járművek kereszteződésbe való behajtásának szabályos sorrendje.
STOP tábla	Elágazásokban a járművek megállását előíró jelzés.
Speciálisan tuningolt autó	A bankrablók járműve, nem tartja be a közlekedési szabályokat és bizonyos mértékig ütközhet is.
Bank	Városban lévő hely. A felhasználó célja ennek kirablása.
Rejtekhely	Városban lévő hely, a felhasználó célja a rablás után ide menekülni.

Fogalom	*Meghatározás*
Elrejtőzés	Az az esemény, amikor a bankrablók autója a rejtekhely melletti elhaladásának következtében eltűnik (bizonyos feltételek teljesülése mellett: ld. feladatleírás). Ezzel a játékos teljesíti feladatát.
(Közlekedési) rendőr	Ellenség, célja, hogy elkapja a bankrablókat, azaz a játékosokat.
Elkap	A rendőr és a bankrabló ütközése.
Fogócska, üldözés	Ha a rendőr meglátja a bankrablót és utána indul, el akarja kapni. Ezt fogócskának nevezzük és addig tart, amíg a rendőr el nem kapja a rablót, illetve ha a az ütközik más járművel.
Zsákmány	A bankból elrabolt pénzmennyiség.
Szintek	Különböző nehézségű városok egymásutánja.
Szint teljesítése	Amikor a bankrabló eléri a rejtekhelyét.
Idő	A szint teljesítéséhez szükséges idő.

2.5 Essential use-case-ek

2.5.1 Use-case diagram



1. ábra: Essential use-case

2.5.2 Use-case leírások

Use-case neve	Új játék indítása
Rövid leírás	A játékos új játékot kezd.
Aktorok	Felhasználó
Forgatókönyv	Megjelenik a pálya alapállapotban.

Use-case neve	Autó irányítása
Rövid leírás	A játékos utasításokat ad a tuningolt autó irányítására.
Aktorok	Felhasználó
Forgatókönyv	Teljesül a játékos által kiadott utasítás.

Use-case neve	Autók léptetése
Rövid leírás	Az autók mozgásának alapja.
Aktorok	Óra
Forgatókönyv	Minden autó lehetőség szerint egységnyit halad.

Use-case neve	Idő léptetése
Rövid leírás	Másodperc alapú időméréshez.
Aktorok	Óra
Forgatókönyv	Az idő egy másodpercet lép. A jelzőlámpák frissülnek.

2.6 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.02.10. 19:00	1 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet. Döntés: Brucker elkészíti szótárt, Ferencz a use-case leírásokat, Orovecz a követelmény definíciót és a projekt tervet, Pocsai a feladatleírást.
2010.02.12. 17:00	1,5 óra	Orovecz	Tevékenység: Projekt definíció megírása.
2010.02.13. 8:00	2 óra	Pocsai	Tevékenység: Feladatleírás elkészítése.
2010.02.13. 13:00	1 óra	Brucker	Tevékenység: Szótár elkészítése.
2010.02.13. 17:00	1 óra	Ferencz	Tevékenység: Use-case leírások elkészítése.
2010.02.13. 19:00	0,25 óra	Ferencz	Tevékenység Verziókezelő rendszer beüzemelése.
2010.02.14. 14:00	1 óra	Brucker	Tevékenység: Beadandó szerkesztése, átnézése
2010.02.15. 17:15	1 óra	Brucker Ferencz Orovecz Pocsai	Konzultáció
2010.02.15. 18:20	0,5 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet. Elvégzett feladatok áttekintése. További tevékenységek meghatározása: Orovecz tökéletesíti a feladatleírást, Ferencz kibővíti a projekt definíciót
2010.02.16. 18:00	1 óra	Orovecz	Tevékenység: a feladatleírás bővítése.
2010.02.16. 18:30	1 óra	Ferencz	Tevékenység: a projekt definíció bővítése.
2010.02.16. 20:00	1 óra	Brucker	Tevékenység: a teljes beadandó átolvasása, apróbb hibák javítása.
2010.02.17. 10:00	1 óra	Ferencz Pocsai	Értekezlet: a teljes beadandó átolvasása. A beadandót teljesnek találtuk.

3. Analízis modell kidolgozása

3.1 Objektum katalógus

A felvett objektumokat a szótár (glossary) alapján határoztuk meg. Az objektumok száma (nyolc) egy pozitív jel számunkra, és azt sugallja, hogy ez a rendszer megfelelő méretű (a lehető legegyszerűbb, ami még nincs túlbonyolítva).

3.1.1 Game

A játékmodell működését összefoglaló osztály. Feladata, hogy indításkor minden objektum a megfelelő állapotba kerüljön, valamint az, hogy a játék főbb mozzanatait kezelje (új játék kezdése, játék elvesztése / megnyerése, stb.).

3.1.2 Timer

A megfelelő időzítésért felelős objektum. Feladata, hogy az események az előre meghatározott sebességgel hajtódjanak végre. Ez az osztály a város „mozgatója”.

3.1.3 City

Összefogja az utakat és a csomópontokat, gondoskodik a pontok közötti kapcsolat létrejöttéről, valamint a városhatáron be- és kilépő autók kezeléséről. Számon tartja a városban lévő autókat és cellákat.

3.1.4 Node

Autókat tartalmazó cella. A városban lévő utak és csomópontok építőeleme. Utat egy sorban összekötött részcsoport alkot, míg egy kereszteződést a cellák összekötött együttese. Tartalmazza az autók számára (tájékozódáshoz és ütközéselkerüléshez) nélkülözhetetlen adatokat.

3.1.5 Car

Reprezentálja az egyes autókat. Ez az objektum tartalmazza a szükséges logikát ahhoz, hogy a városban megvalósuljon a biztonságos közlekedés. Minden autó rendelkezik bizonyos mennyiségű védelemmel, ami ütközéseknél kap kulcsszerepet.

3.1.6 RunnerCar

A rablók speciálisan tuningolt kocsija, mely csak a felhasználó utasításait követi, a szabályokat viszont nem tartja be.

3.1.7 PoliceCar

A rendőrök járműve. A szabályok betartására vonatkozó logika megegyezik az egyszerű autókéval. A rendőrautó egyetlen előnye a többi autóval szemben az, hogy a rablókkal való ütközés esetén rögtön elkapja azt.

3.1.8 CarFactory

Az autók gyártását elvégző osztály. Véletlenszerű időközönként gyártja le a városba érkező autókat.

3.2 Osztályok leírása

3.2.1 Game

- **Felelősség**

Az osztály felelőssége, hogy kezelje a játék főbb mozzanatait (új játék kezdése, játék megnyerése, játék elvesztése). Egy másik fontos felelőssége az, hogy továbbítsa a Timer osztály által adott függvényhívást a többi osztály felé, ezáltal biztosítva a játék mozgását.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **carFactory: CarFactory** – tartalmazza az autók készítéséért felelős osztályt.
  - **city: City** – a város, amely felé továbbítani kell a Timer által adott „impulzust”.
  - **timer: Timer** – ettől kapja a Game osztály a mozgáshoz szükséges időzítést.

- **Metódusok**
  - **void tick()**: ezt a függvényt hívja meg a Timer osztály. Meghívja a CarFactory és a City megfelelő metódusait.
  - **void startGame()**: ez a függvény létrehozza a megfelelő osztályokat, majd meghívja a City osztályinicializáló függvényét.
  - **void endGame()**: elvégzi a játékos vesztése esetén a szükséges lépéseket.
  - **void winGame()**: a játékos győzelme esetén hívódik meg.

3.2.2 Timer

- **Felelősség**

A megfelelő időzítésért felelős objektum. Feladata, hogy az események megtörténjenek az előre meghatározott sebességgel. Ez az osztály a város „mozgatója”.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs. (Megj. A java beépített Runnable interfészét valósítja meg.)

- **Attribútumok**
  - **period: int** – Az ütemek közötti időbeli eltérés (periódus) milliszekundumban kifejezve.
  - **game: Game** – A Game osztályt ezen referencia segítségével éri el a Timer osztály.

- **Metódusok**
  - **void run()**: ez a metódus folyamatosan fut és megfelelő időközönként elvégzi az osztály feladatát.
  - **void setPeriod(int i)**: egyszerű beállító (setter) függvény, mely beállítja a megfelelő periódust.
  - **void setGame(Game game)**: egyszerű beállító (setter) függvény, mely beállítja a Game osztályra mutató tagváltozót.

3.2.3 City

- **Felelősség**

Összefogja az utakat és a csomópontokat, gondoskodik a pontok közötti kapcsolat létrejöttéről, valamint a városhatáron ki- és belépő autók kezeléséről. Számon tartja a városban lévő autókat és pontokat.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **game: Game –** a várost tartalmazó Game osztály.
  - **nodes: TreeMap<Integer, Nodes> -** ebben az adatszerkezetben tároljuk a Node osztályokat, így később könnyen elővehetők összekötés, illetve léptetés céljából.
  - **cars: LinkedList<Car>** - az autókat tartalmazó adatszerkezet.

- **Metódusok**
  - **void tick()**: az ütemet lekezelő függvény.
  - **void init()**: inicializálja az osztályt, megalkotva a szükséges Node tagokat, azokat megfelelően összekötve.
  - **void newCar(Car c)**: A megfelelő Node-ba elhelyezi a paraméterül kapott Car objektumot. Célszerűen olyan Node-ot választ, aminek nincs másik bemeneti Node-ja. (Az ilyen Node reprezentálja a város szélét)
  - **void newRunnerCar(Car c)**: Abban különbözik a newCar(Car c) függvénytől, hogy ez az előre kiválasztott Node-ok közül az egyikbe rakja az új autót (az ilyen Node-ok reprezentálják a bankokat).
  - **void setCarFactory(carFactory CarFactory)**: Egyszerű beállító (setter) függvény a megfelelő adattaghoz.
  - **Game getGame()**: Egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **removeCar(Car c)**: eltávolítja a paraméterül kapott autót a városból.

3.2.4 Node

- **Felelősség**

Autókat tartalmazó cella. A városban lévő utak és csomópontok építőeleme. Utat egy sorban összekötött részcsoport alkot, míg egy kereszteződést a cellák összekötött együttese.

Tartalmazza az autók számára (tájékozódáshoz és ütközéselkerüléshez) nélkülözhetetlen adatokat

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **trafficLight: int** – a jelzőlámpa állását megadó változó. Egy előre kiválasztott állapota fogja jelenteni azt, hogy a jelzőlámpa sárgán villog (azaz olyan mintha nem is lenne) és az autók a táblák segítségével állapítják meg az elsőbbséget.
  - **priority: LinkedList<int>** - elsőbbségi sorrendben tartalmazza az irányoknak megfelelő útvonalakat. Ez alapján tudja eldönteni bármelyik autós azt, hogy melyik iránynak kell elsőbbséget adnia.
  - **inNodes: Node[]** – az adott Node egy másik Node-dal való összeköttetésének eltárolására szolgáló adatstruktúra. Fontos a kapcsolat iránya is (befelé).
  - **outNodes: Node[]** – az adott Node egy másik Node-dal való összeköttetésének eltárolására szolgáló adatstruktúra. Fontos a kapcsolat iránya is (befelé).
  - **car: Car** – a kereszteződésen éppen áthaladó autót tartalmazza.
  - **hiding: Boolean** – annak megállapítására szolgál, hogy az éppen aktuális Node alkalmas-e a rablók autójának elrejtésére.
  - **city: City** – a tartalmazó városra mutató adattag.

- **Metódusok**
  - **void step()**: ez a függvény lépteti a jelzőlámpát (amennyiben az aktív).
  - **Node[] getOutNodes():** egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **Node[] getInNodes():** egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **LinkedList<Integer> getPriority**: egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **Car[] getCar()**: egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **void connect(Boolean in, int dir, Node n)**: beregisztrálja a kapcsolatot a csomópont ezen oldalán.
  - **setCar(Car c)**: elmenti (setter függvény) a paraméterül kapott autót.
  - **Boolean isHiding():** egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **int getTrafficLight():** a lámpa állását lekérdező (getter) függvény.

3.2.5 Car

- **Felelősség**

Ez az objektum tartalmazza a szükséges logikát ahhoz, hogy a városban megvalósuljon a biztonságos közlekedés. Minden autó rendelkezik bizonyos mennyiségű védelemmel, ami ütközéseknél kap kulcsszerepet.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **node: Node** – tartalmazza az éppen elfoglalt Node-ot.
  - **speed: int** – az autó sebessége. A autó valós sebességét az határozza meg, hogy hány ütemet vár két lépés között.
  - **wait: int** –a következő lépésig hány ütemet kell még várni.
  - **strength: int** – az autó megmaradt ereje, melynek az ütközéseknél van jelentősége.
  - **city: City** – a várost eltároló adattag.

- **Metódusok**
  - **void step()**: ez a függvény lépteti az autót.
  - **void collision(RunnerCar r):** ezt a függvényt a RunnerCar hívhatja meg, mégpedig akkor, ha a pályák keresztezik egymást.
  - **int getStrength():** egyszerű lekérdező (getter) függvény a megfelelő adattaghoz.
  - **void destroy():** az autó megsemmisítésénél elvégzi a szükséges lépéseket.
  - **void setCity(City city):** egyszerű beállító (setter) függvény.

3.2.6 RunnerCar

- **Felelősség**

Az osztály felelőssége, hogy a játékos parancsainak megfelelően lépjen, és kezelje az esedékes interakciókat.

- **Ősosztályok**

Car

- **Interfészek**

Nincs

- **Attribútumok**

Nincs

- **Metódusok**
  - **void CollisionWithCar(Car c)** – az a hagyományos autó hívja meg ezt a függvényt, aminek a RunnerCar előzetesen meghívta a collision(RunnerCar c) függvényét.
  - **void CollisionWithPolice(PoliceCar c) –** az a rendőrautó hívja meg ezt a függvényt, aminek a RunnerCar előzetesen meghívta a collision(RunnerCar c) függvényét.
  - **void step()** – a RunnerCar más megfontolások alapján lép, mint a Car, ezért ez a függvény felül lett definiálva.

3.2.7 PoliceCar

- **Felelősség**

Felelőssége megegyezik a Car objektuméval, annyi különbséggel, hogy ütközés esetén másképpen viselkedik.

- **Ősosztályok**

Car

- **Interfészek**

Nincs

- **Attribútumok**

Nincs

- **Metódusok**
  - **void Collision(RunnerCar c) -** Ezt a függvényt a RunnerCar hívhatja meg, mégpedig akkor, ha a pályák keresztezik egymást.

3.2.8 CarFactory

- **Felelősség**

A CarFactory felelőssége, hogy megfelelő időközönként új autókat küldjön a városba, ezáltal fenntartva a városban az egyensúlyt. Ez az osztály felel azért is, hogy ne csak hagyományos autók (Car), hanem rendőrautók (PoliceCar) is legyenek a városban.

- **Ősosztályok**

Nincs

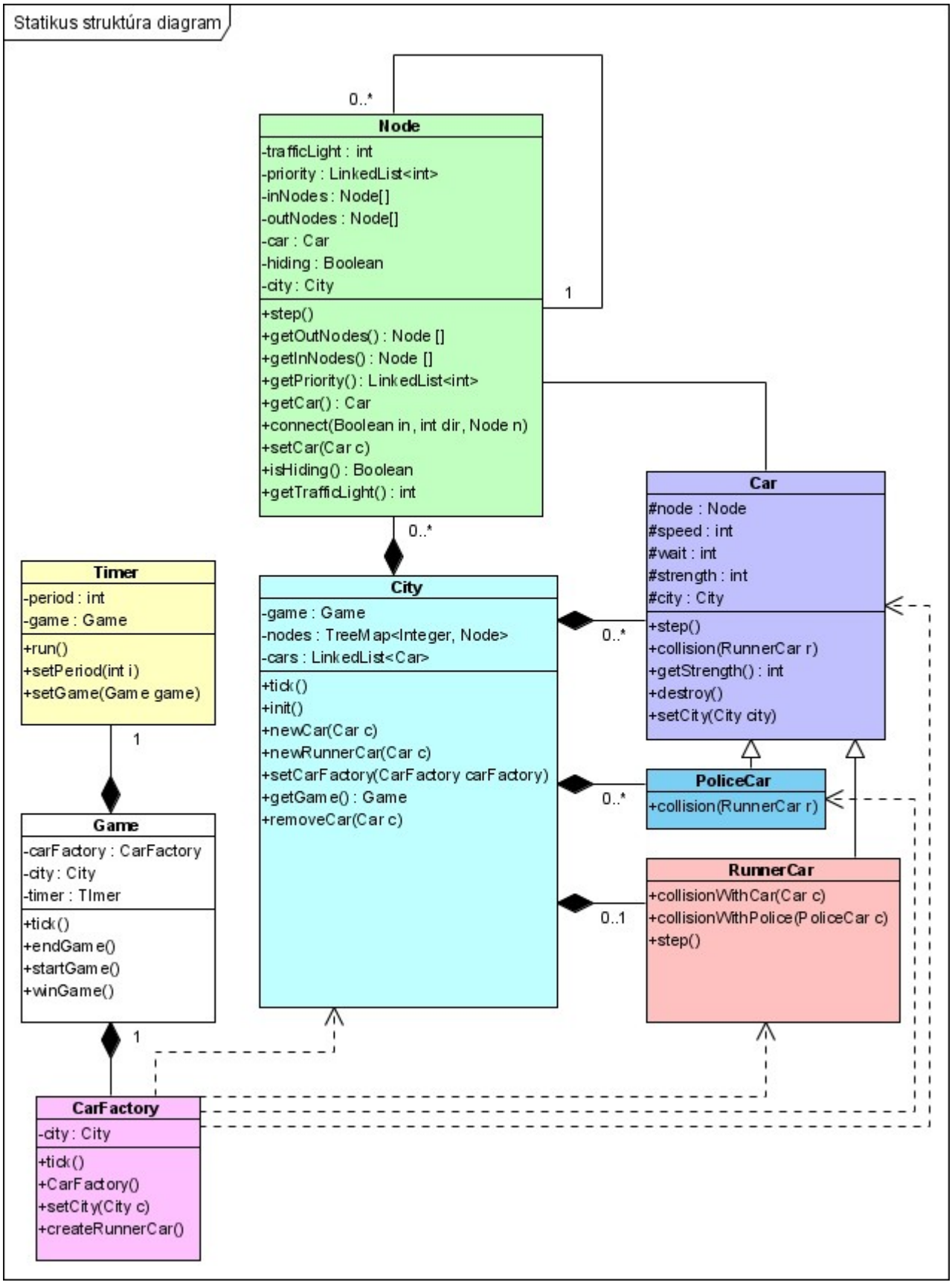
- **Interfészek**

Nincs

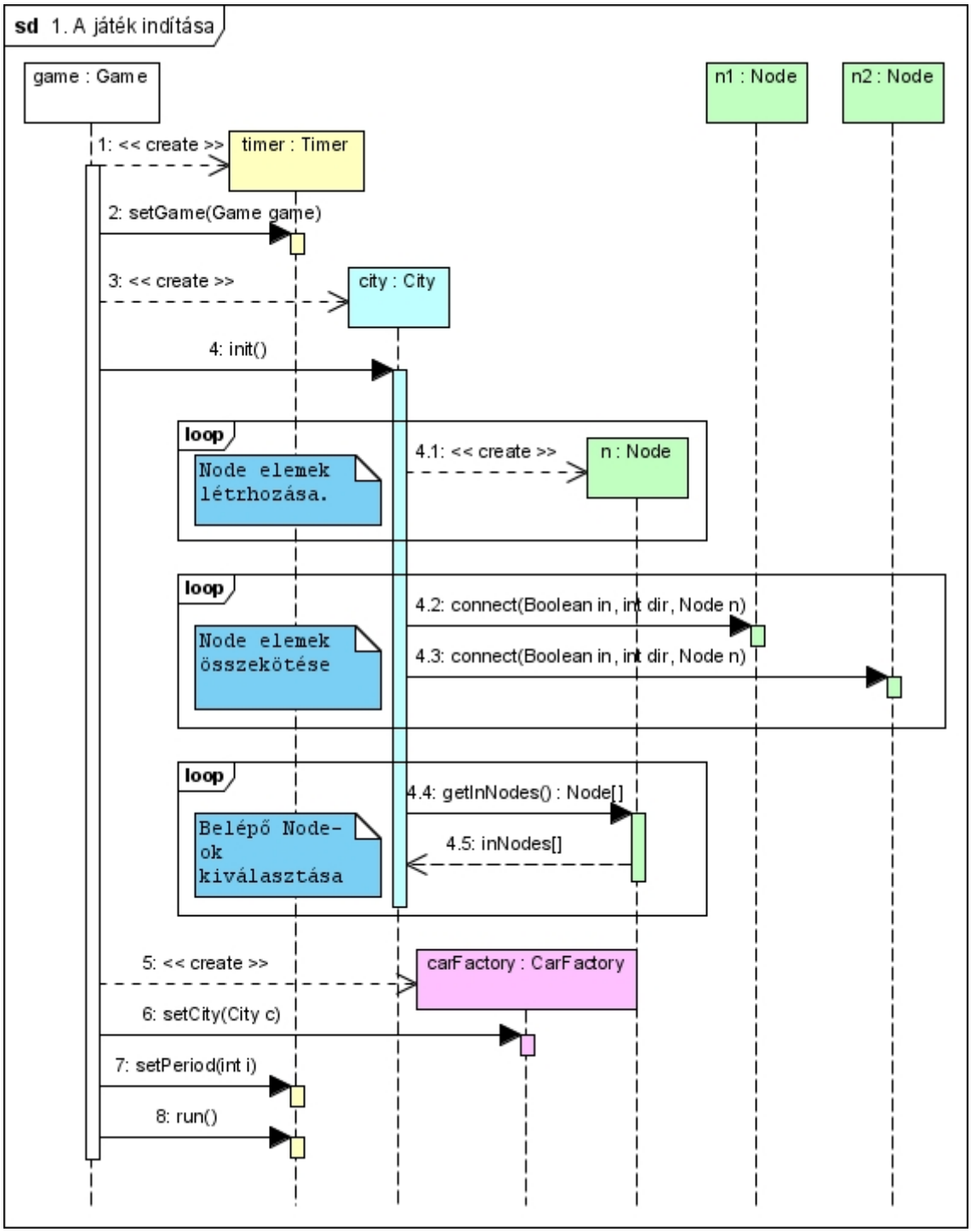
- **Attribútumok**
  - **city: City**– A CarFactory ismeri a várost, aminek átadja a legyártott autókat.

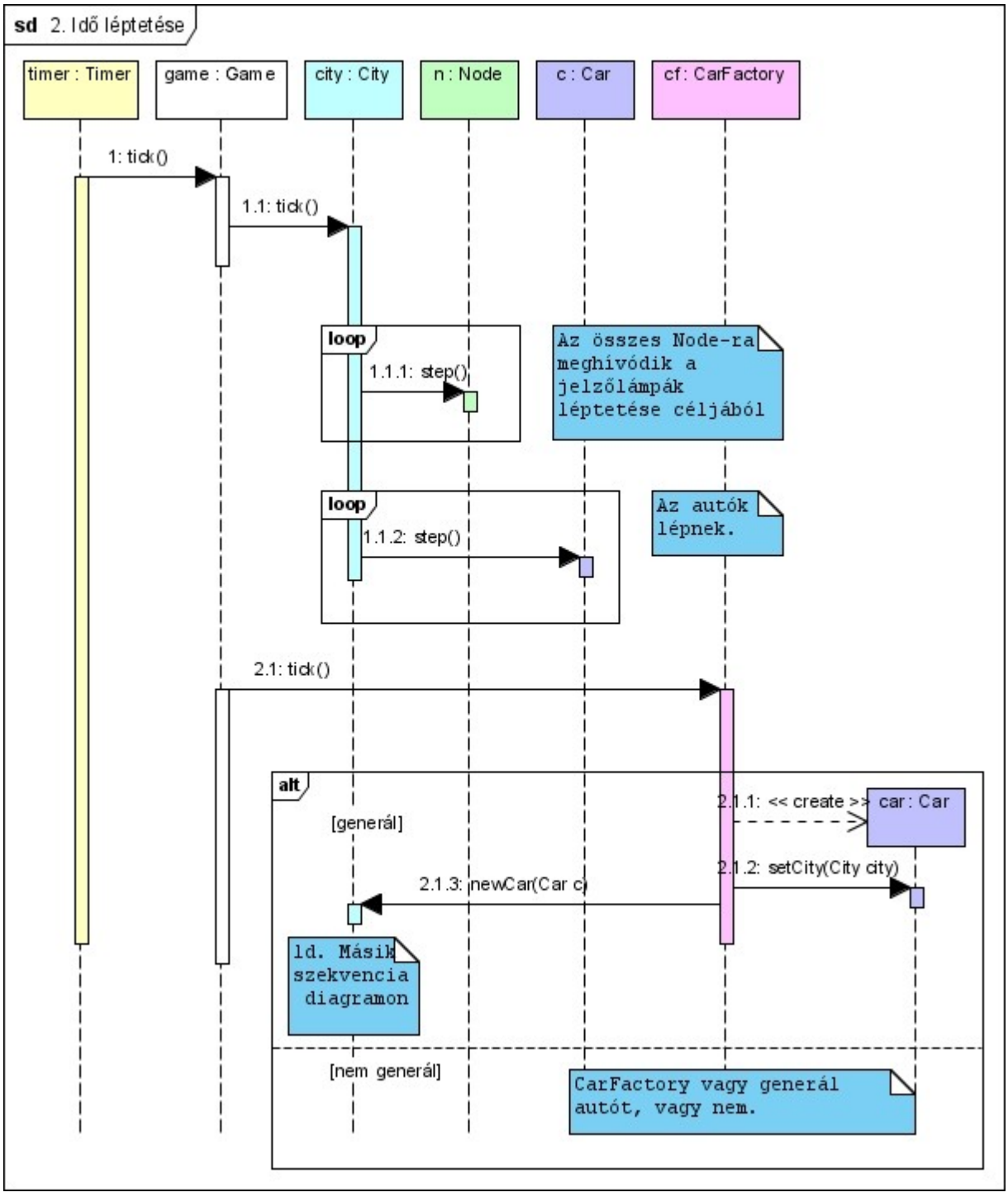
- **Metódusok**
  - **void tick()** – a léptetéshez szükséges metódus. Ennek hatására a CarFactory bizonyos valószínűséggel legyárt egy autót (Car-t vagy PoliceCar-t).
  - **void CarFactory()** – a CarFactory konstruktora.
  - **void setCity** – egyszerű beállító (setter) függvény.
  - **void createRunnerCar()** – Ez a függvény akkor kerül meghívásra, amikor a játékos elkezdi a játékot azért, hogy a City objektumnak átadja a megfelelő RunnerCar objektumot.

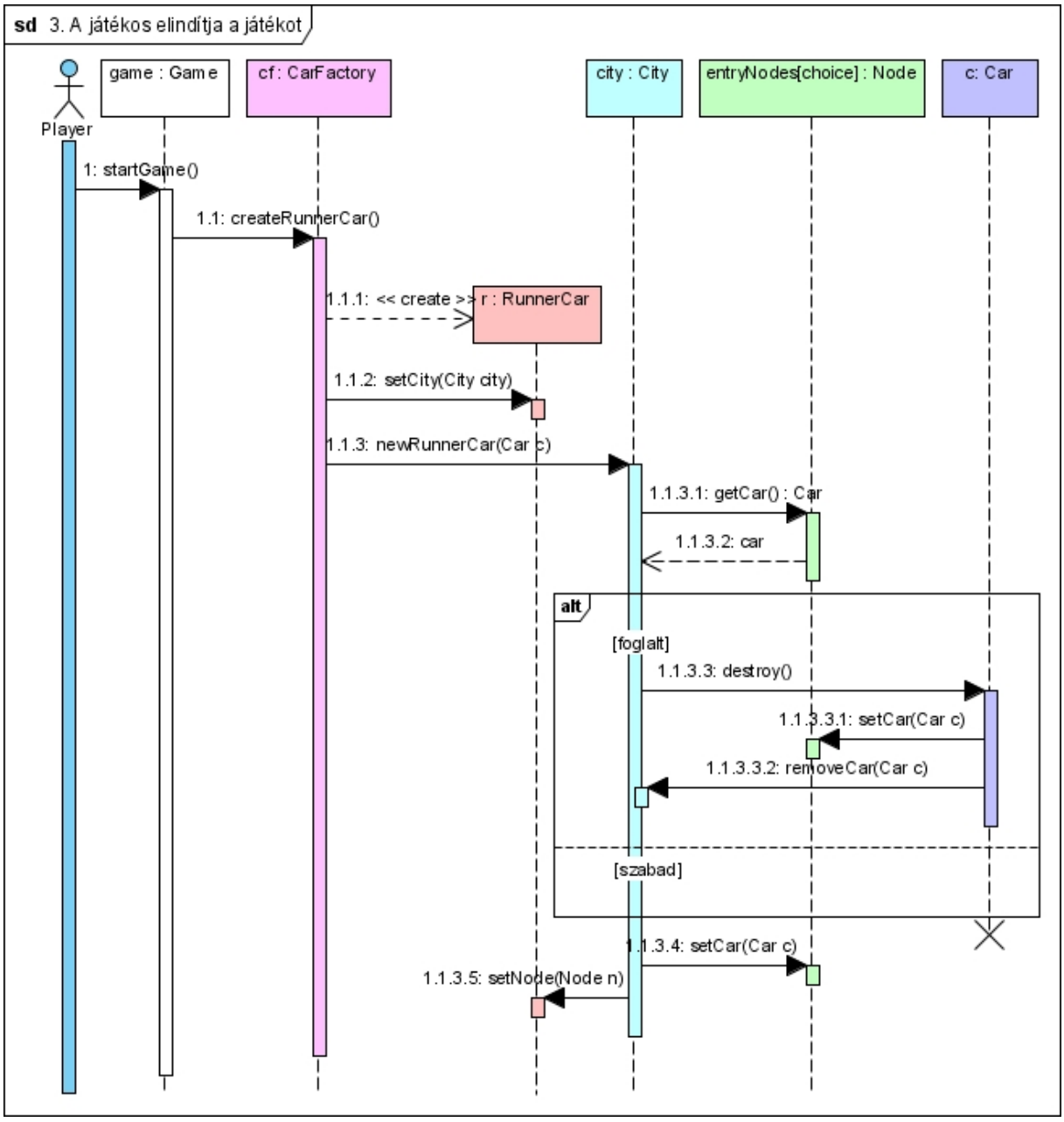
3.3 Statikus struktúra diagramok

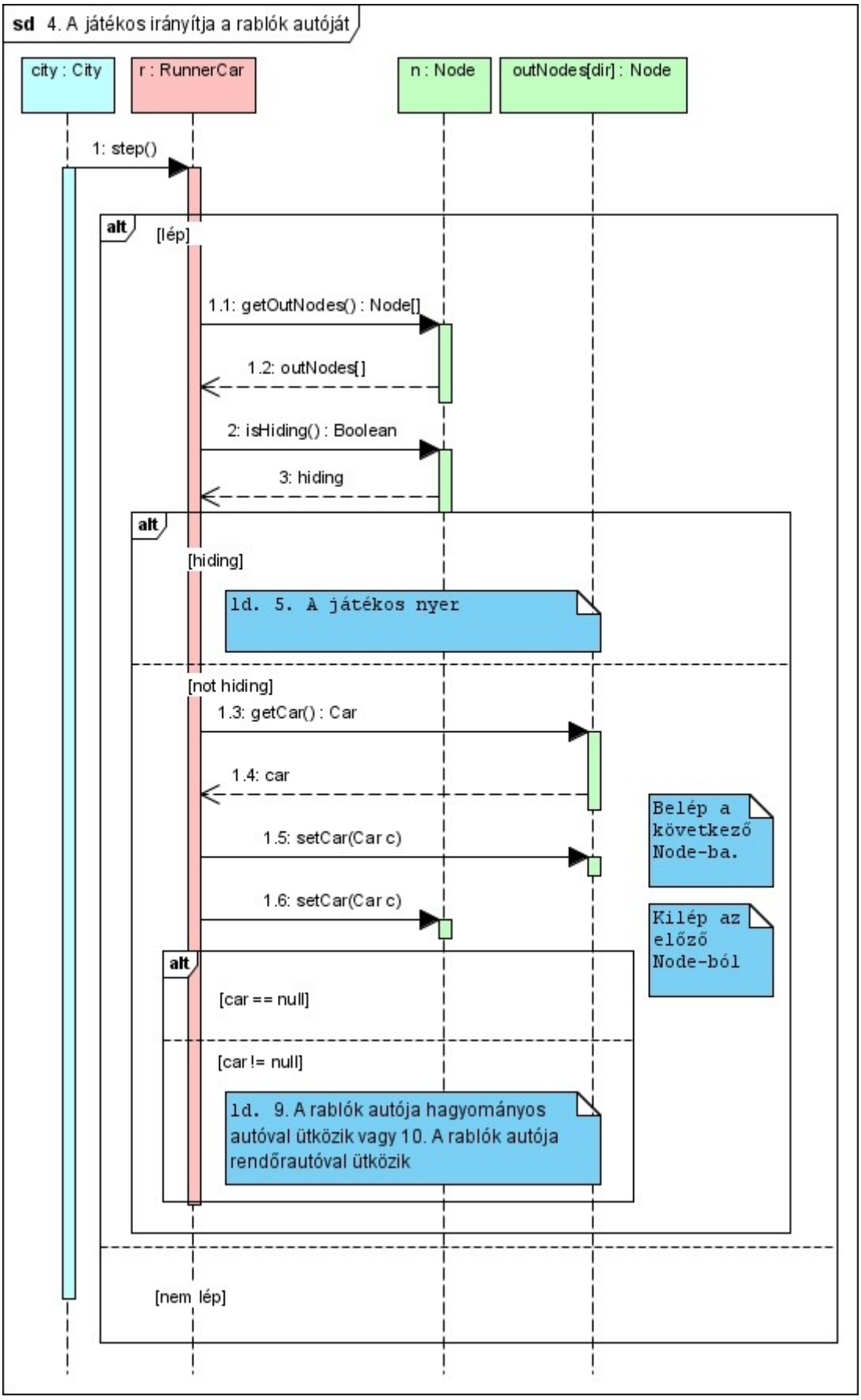


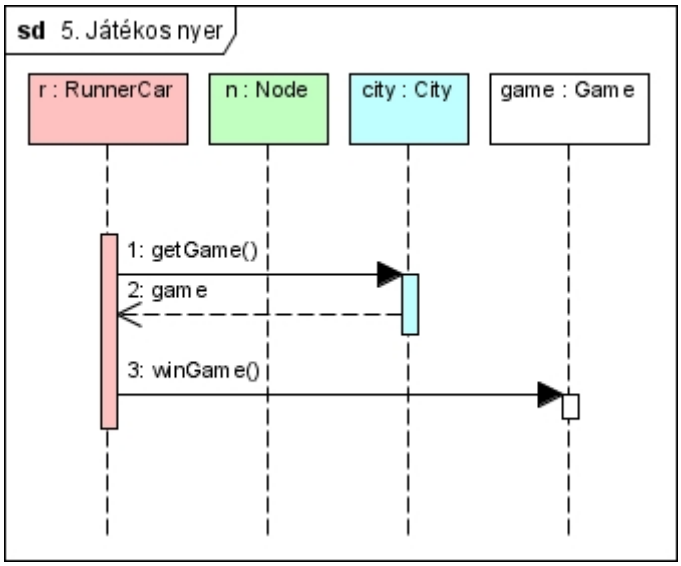
3.4 Szekvencia diagramok

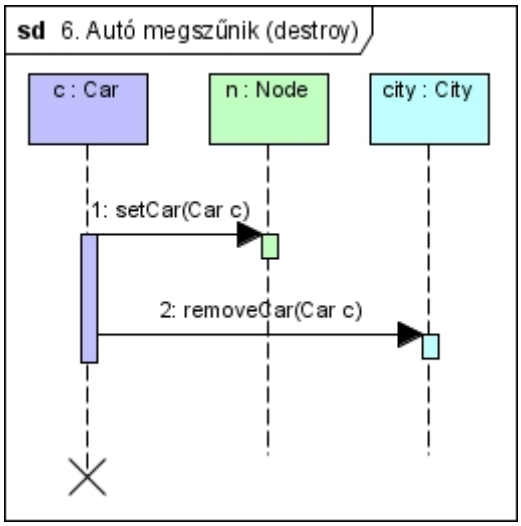


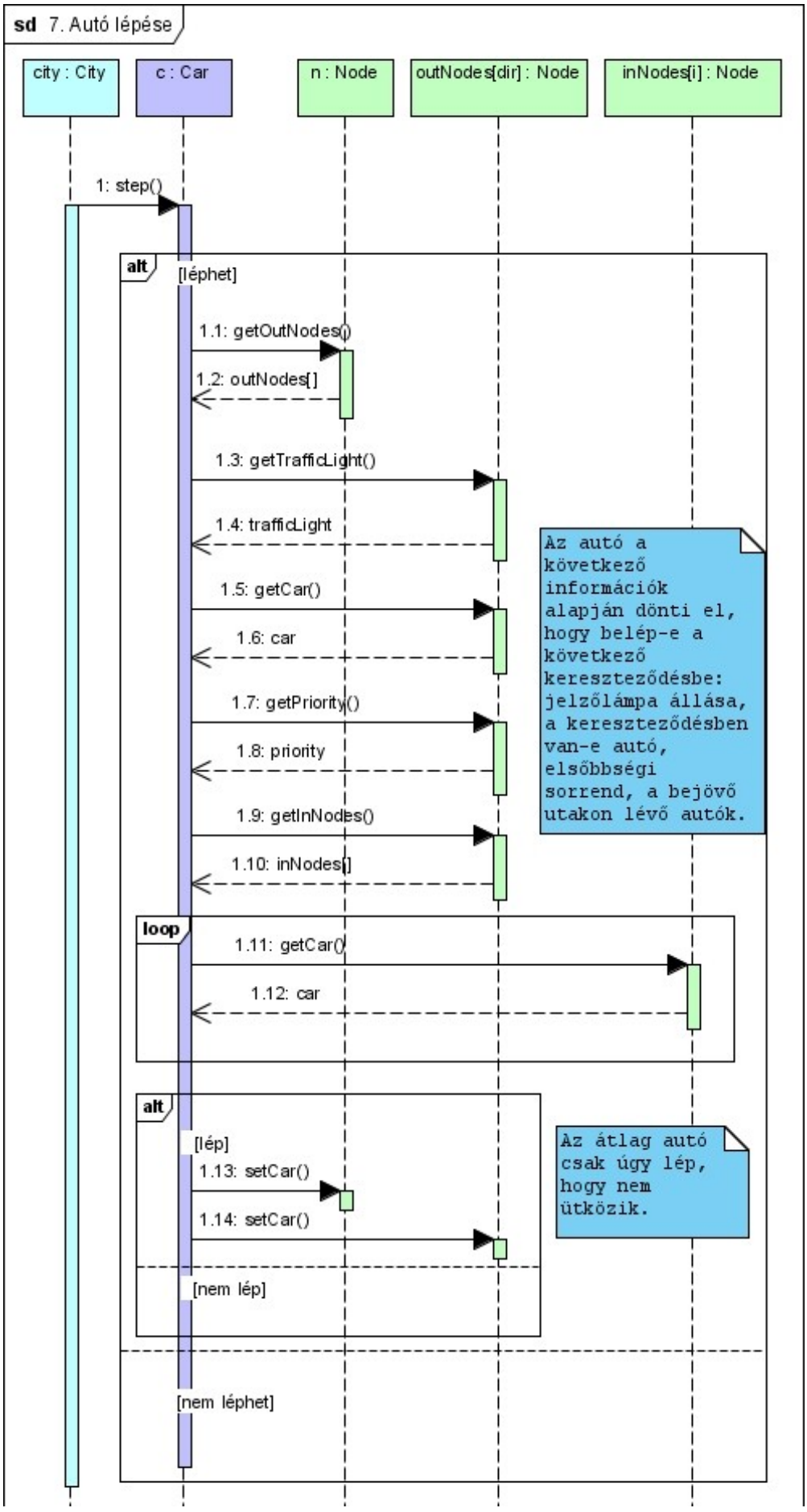

sd 2. Idő léptetése
timer : Timer
game : Game
city : City
n : Node
c : Car
cf : CarFactory
1: tick()
1.1: tick()
loop
1.1.1: step()
Az összes Node-ra meghívódik a jelzőlámpák léptetése céljából
loop
1.1.2: step()
Az autók lépnek.
2.1: tick()
alt
[generál]
2.1.1: << create >>
car : Car
2.1.2: setCity(City city)
2.1.3: newCar(Car c)
ld. Másik szekvencia diagramon
[nem generál]
CarFactory vagy generál autót, vagy nem.

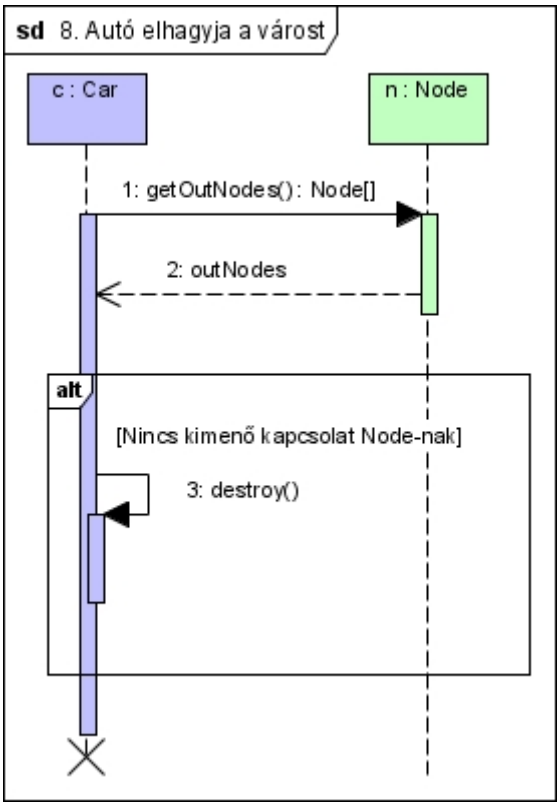

sd 3. A játékos elindítja a játékot
Player
game : Game
cf : CarFactory
city : City
entryNodes[choice] : Node
c: Car
1: startGame()
1.1: createRunnerCar()
1.1.1: << create >>
r : RunnerCar
1.1.2: setCity(City city)
1.1.3: newRunnerCar(Car c)
1.1.3.1: getCar() : Car
1.1.3.2: car
alt
[foglalt]
1.1.3.3: destroy()
1.1.3.3.1: setCar(Car c)
1.1.3.3.2: removeCar(Car c)
[szabad]
1.1.3.4: setCar(Car c)
1.1.3.5: setNode(Node n)

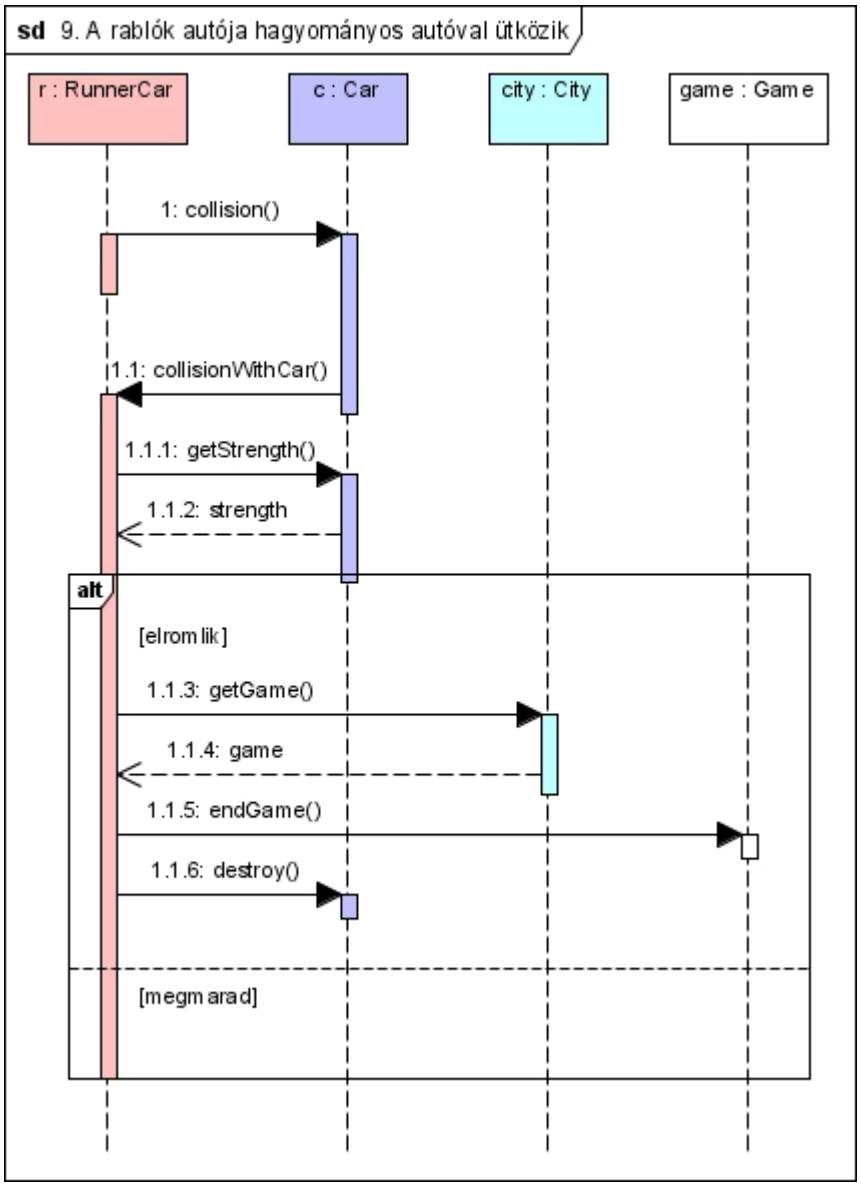

sd 4. A játékos irányítja a rablók autóját
city : City
r : RunnerCar
n : Node
outNodes[dir] : Node
1: step()
alt
[lép]
1.1: getOutNodes() : Node[]
1.2: outNodes[]
2: isHiding() : Boolean
3: hiding
alt
[hiding]
ld. 5. A játékos nyer
[not hiding]
1.3: getCar() : Car
1.4: car
1.5: setCar(Car c)
1.6: setCar(Car c)
Belép a következő Node-ba.
Kilép az előző Node-ból
alt
[car == null]
[car != null]
ld. 9. A rablók autója hagyományos autóval ütközik vagy 10. A rablók autója rendőrautóval ütközik
[nem lép]

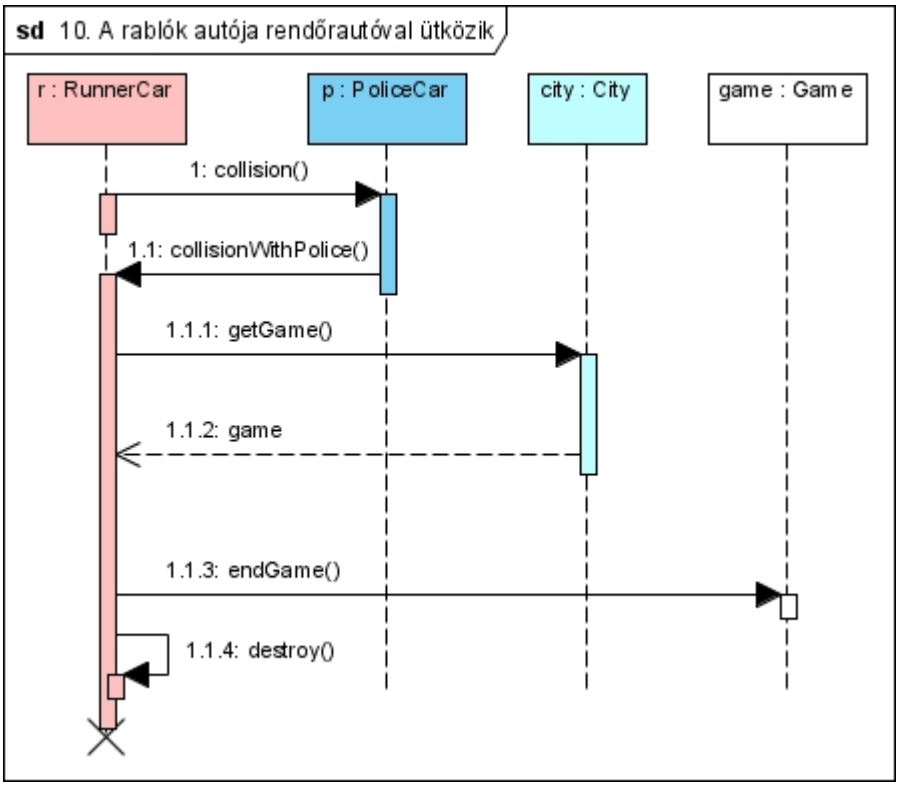

sd 5. Játékos nyer
r : RunnerCar
n : Node
city : City
game : Game
1: getGame()
2: game
3: winGame()

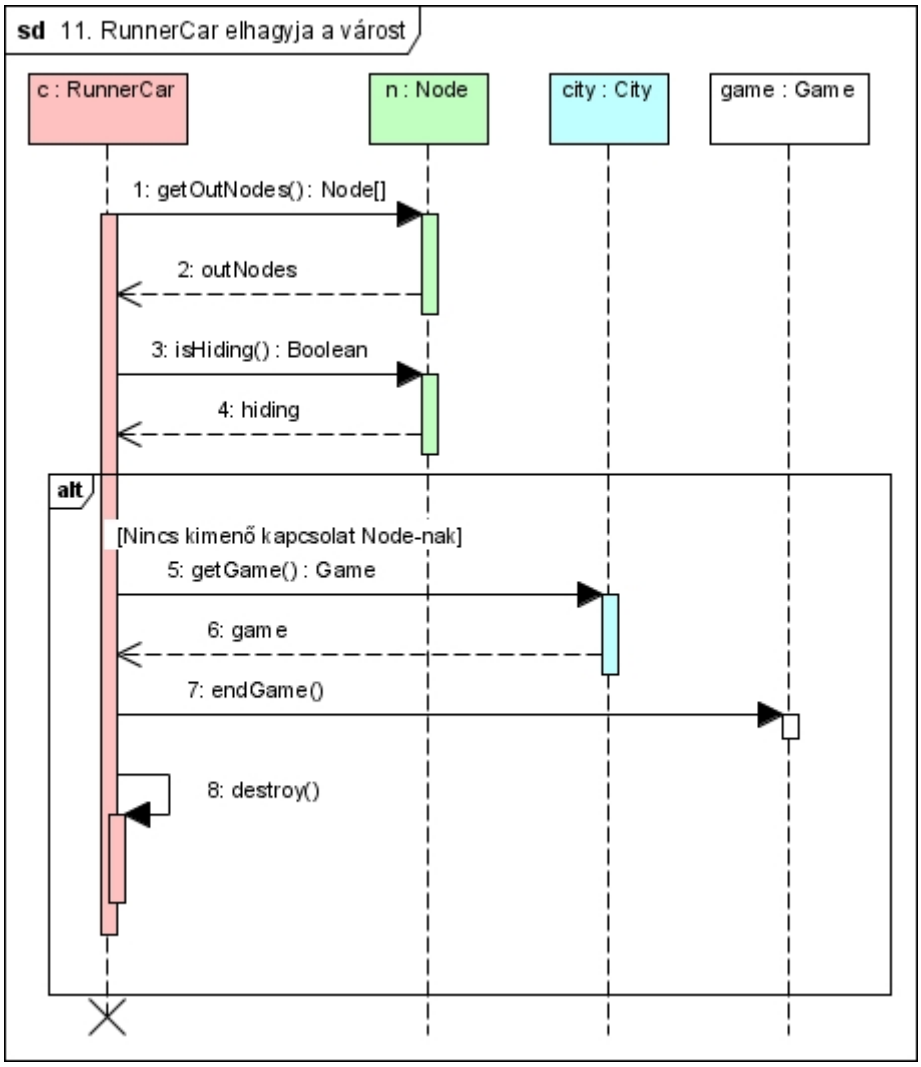

sd 6. Autó megszűnik (destroy)
c : Car
n : Node
city : City
1: setCar(Car c)
2: removeCar(Car c)

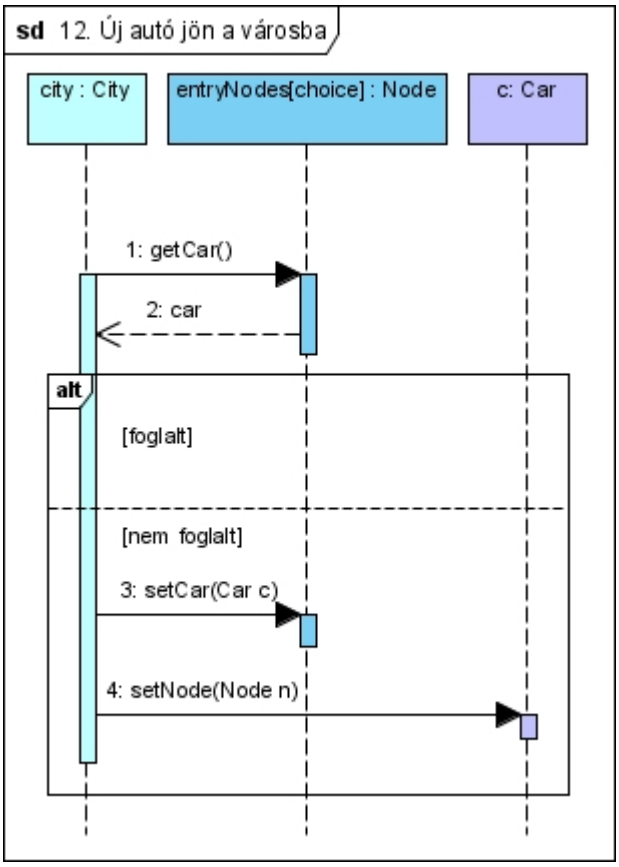

sd 7. Autó lépése
city : City
c : Car
n : Node
outNodes[dir] : Node
inNodes[i] : Node
1: step()
alt
[léphet]
1.1: getOutNodes()
1.2: outNodes[]
1.3: getTrafficLight()
1.4: trafficLight
Az autó a következő információk alapján dönti el, hogy belép-e a következő kereszteződésbe: jelzőlámpa állása, a kereszteződésben van-e autó, elsőbbségi sorrend, a bejövő utakon lévő autók.
1.5: getCar()
1.6: car
1.7: getPriority()
1.8: priority
1.9: getInNodes()
1.10: inNodes[]
loop
1.11: getCar()
1.12: car
alt
[lép]
1.13: setCar()
1.14: setCar()
Az átlag autó csak úgy lép, hogy nem ütközik.
[nem lép]
[nem léphet]


sd 8. Autó elhagyja a várost
c : Car
n : Node
1: getOutNodes() : Node[]
2: outNodes
alt
[Nincs kimenő kapcsolat Node-nak]
3: destroy()


sd 9. A rablók autója hagyományos autóval ütközik
r : RunnerCar
c : Car
city : City
game : Game
1: collision()
1.1: collisionWithCar()
1.1.1: getStrength()
1.1.2: strength
alt
[elromlik]
1.1.3: getGame()
1.1.4: game
1.1.5: endGame()
1.1.6: destroy()
[megmarad]


sd 10. A rablók autója rendőrautóval ütközik
r : RunnerCar
p : PoliceCar
city : City
game : Game
1: collision()
1.1: collisionWithPolice()
1.1.1: getGame()
1.1.2: game
1.1.3: endGame()
1.1.4: destroy()


sd 11. RunnerCar elhagyja a várost
c : RunnerCar
n : Node
city : City
game : Game
1: getOutNodes() : Node[]
2: outNodes
3: isHiding() : Boolean
4: hiding
alt
[Nincs kimenő kapcsolat Node-nak]
5: getGame() : Game
6: game
7: endGame()
8: destroy()


sd 12. Új autó jön a városba
city : City
entryNodes[choice] : Node
c: Car
1: getCar()
2: car
alt
[foglalt]
[nem foglalt]
3: setCar(Car c)
4: setNode(Node n)

3.5 State-chartok

Egyik osztály sem tartalmaz olyan állapotokat, melyek szemléltetése célszerű lenne state-charttal.

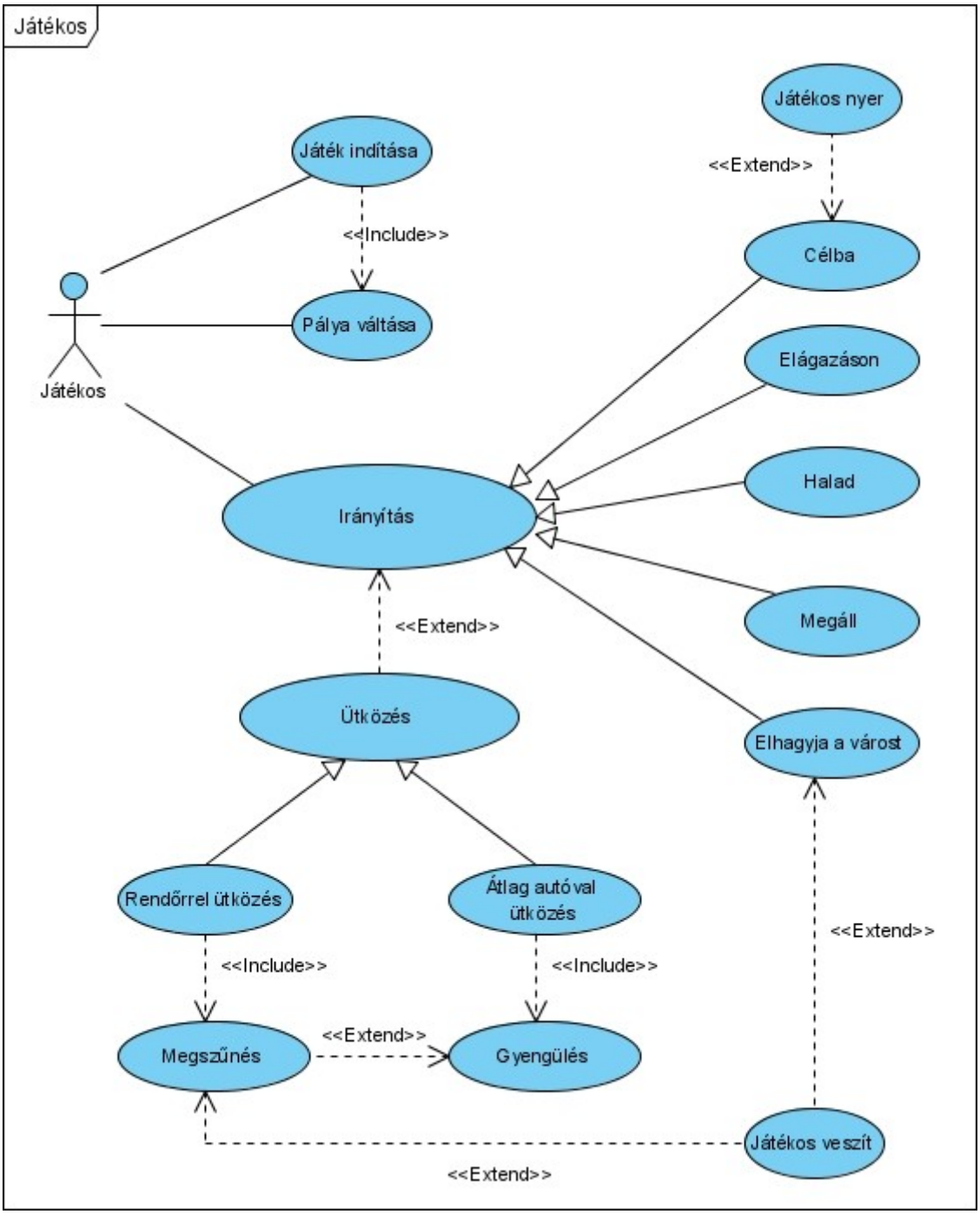
3.6 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.02.18. 16:15	0,5 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet. A következi heti teendők áttekintése. Előzetes (konzultáció előtti) feladatok kiosztása. Ferencz készít egy előzetes osztálydiagramot és terveket a működésükre (state-chart is).
2010.02.19. 12:00	1 óra	Ferencz Pocsai	Értekezlet. Az osztálydiagram megvitatása. Néhány változás bevezetése.
2010.02.21. 14:30	2 óra	Brucker	Dokumentáció írás az osztálydiagram alapján.
2010.02.21. 18:30	2 óra	Brucker	Dokumentáció írás folytatása az osztálydiagram alapján.
2010.02.21. 17:00	0,5 óra	Pocsai	Osztálydiagram digitalizálása.
2010.02.22. 17:00	1 óra	Ferencz Pocsai	Konzultáció. Eredményei: lényeges változások az osztálymodellben, iterációs munkamódszer bevezetése.
2010.02.22. 19:00	3 óra	Ferencz	Objektumdiagram készítése, szekvencia diagramok készítése. (1. iteráció)
2010.02.22. 20:00	1 óra	Orovecz	Osztálykatalógus megírása.
2010.02.23. 09:45	0,5 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet: Hogyan javítsunk az osztálymodellünkön. Eredmény: apróbb módosítások néhány osztály adattagjai és függvényei között.
2010.02.23. 17:00	2 óra	Pocsai	Osztálymodell tökéletesítése (2. iteráció).
2010.02.23. 09:45	0,5 óra	Brucker Ferencz Pocsai	Értekezlet: Hogyan javítsunk az osztálymodellünkön. Eredmény: nem találtunk javítanivalót.
2010.02.23. 20:00	2 óra	Orovecz	Beadandó összeállítása.
2010.02.24. 09:45	0,5 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet: Beadandó áttekintése. Eredmény: néhány szekvenciadiagram nem túl átlátható, ezek újrarajzolása, valamint néhány fogalmazásbeli pontosítás.
2010.02.24 20:00	1 óra	Ferencz	Beadandó javítása.
2010.02.24 23:00	1 óra	Brucker	Beadandó ellenőrzése. Nem talált hibát.
2010.02.24 24:00	0,5 óra	Orovecz	Nyomtatás.

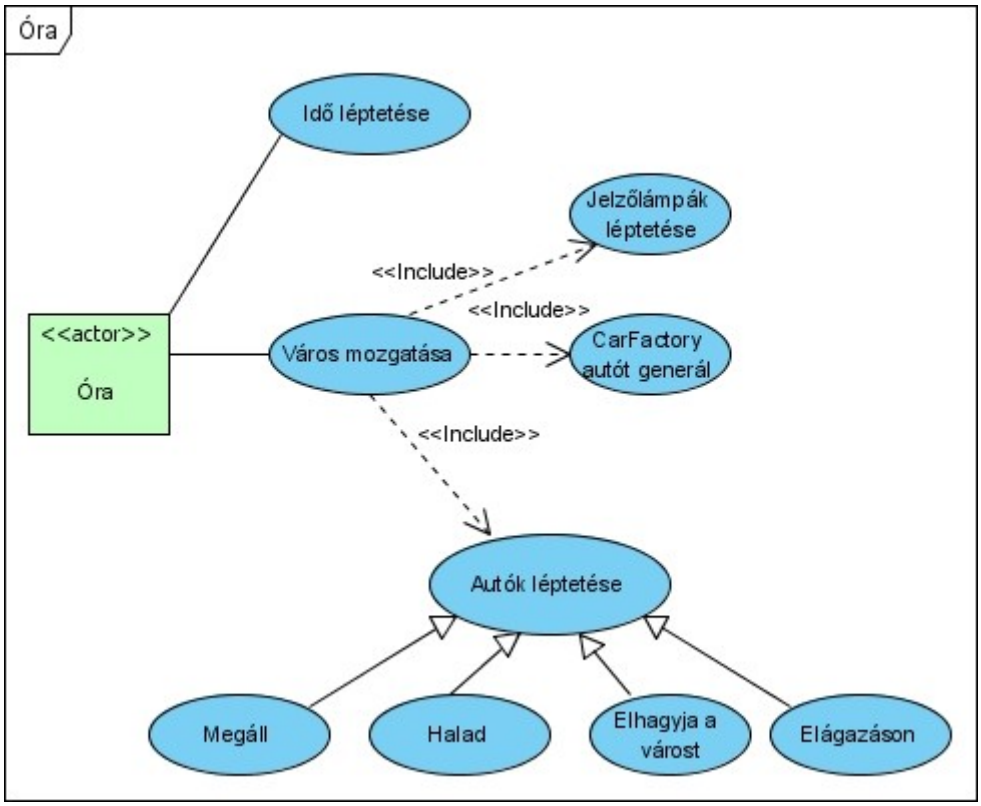
1. Szkeleton tervezése

1.1 A szkeleton modell valóságos use-case-ei

1.1.1 Use-case diagram



2. ábra: Játékoshoz tartozó Use-case diagram



3. ábra: Az órához tartozó Use-case diagram

1.1.2 Use-case leírások

Use-case neve	Játék indítása
Rövid leírás	A játékos elindítja a játékot.
Aktorok	Játékos
Forgatókönyv	Betöltődik az első pálya.

Use-case neve	Pálya váltása
Rövid leírás	A felhasználó tud pályát váltani.
Aktorok	Játékos
Forgatókönyv	A pálya megváltozik.

Use-case neve	Irányítás
Rövid leírás	A játékos irányítja a bankrablók autóját.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója a játékos irányítása szerint lép.

Use-case neve	Célba
Rövid leírás	A játékos a célba irányítja a bankrabló autóját.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója eltűnik. A játékos nyer.

Use-case neve	Játékos nyer
Rövid leírás	A játékos megnyeri a játékot.
Aktorok	Játékos
Forgatókönyv	A játékos nyer.

Use-case neve	Elhagyja a várost
Rövid leírás	A játékos úgy irányítja a bankrablók autóját, hogy az elhagyja a várost.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója elhagyja a várost. A játékos vesztett.

Use-case neve	Játékos veszít
Rövid leírás	A játékos elveszti a játékot.
Aktorok	Játékos
Forgatókönyv	A játékos elvesztette a játékot.

Use-case neve	Elágazáson
Rövid leírás	A játékos az elágazásban irányítja a bankrabló autóját.
Aktorok	Játékos
Forgatókönyv	A rablók autója a játékos által meghatározott irányba megy.

Use-case neve	Halad
Rövid leírás	A játékos elágazás nélküli útszakaszon irányítja a bankrabló autóját.
Aktorok	Játékos
Forgatókönyv	A rablók autója továbblép a következő mezőre.

Use-case neve	Megáll
Rövid leírás	A játékos megállítja a bankrabló autóját.
Aktorok	Játékos
Forgatókönyv	A rablók autója helyben marad.

Use-case neve	Ütközés
Rövid leírás	A játékos egy másik autóval ütközik.
Aktorok	Játékos
Forgatókönyv	Az ütközést kezeljük.

Use-case neve	Rendőrrel való ütközés
Rövid leírás	A bankrablók autója egy rendőrautóval ütközik.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója megszűnik. A játékos veszít.

Use-case neve	Megszűnés
Rövid leírás	A bankrablók autója összetörik.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója megszűnik. A játékos veszít.

Use-case neve	Átlag autóval való ütközés
Rövid leírás	A bankrablók autója egy átlag autóval ütközik.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója veszít erejéből, melynek eredménye megszűnés lehet.

Use-case neve	Gyengülés
Rövid leírás	A játékos autója egy ütközés során veszít erejéből.
Aktorok	Játékos
Forgatókönyv	A bankrablók autója veszít erejéből, melynek eredménye megszűnés lehet.

Use-case neve	Idő léptetése
Rövid leírás	Az időszámláló lép.
Aktorok	Óra
Forgatókönyv	A játékidőt mérő szám eggyel növekszik.

Use-case neve	Város léptetése
Rövid leírás	A város objektumai lépnek.
Aktorok	Óra
Forgatókönyv	A város mozgó elemei lépnek.

Use-case neve	CarFactory autót generál
Rövid leírás	Új autó keletkezése.
Aktorok	Óra
Forgatókönyv	Bizonyos valószínűséggel új autó jelenik meg a pályán.

Use-case neve	Jelzőlámpa léptetése
Rövid leírás	A jelzőlámpa egy másik állapotra vált.
Aktorok	Óra
Forgatókönyv	A jelzőlámpa más irányból engedi be az autókat a mezőbe.

Use-case neve	Autók léptetése
Rövid leírás	Minden autó továbbhaladhat az úton.
Aktorok	Óra
Forgatókönyv	Az autók lehetőségeik szerint lépnek.

Use-case neve	Elágazáson
Rövid leírás	Az autó egy elágazáson halad tovább.
Aktorok	Óra
Forgatókönyv	Az autó irányt választ és lehetőség szerint tovább lép.

Use-case neve	Elhagyja a várost
Rövid leírás	Az autó kimegy a városból.
Aktorok	Óra
Forgatókönyv	Az autó elhagyja a várost, a játék szempontjából megszűnik.

qUse-case neve	Halad
Rövid leírás	Az autó az úton továbbhalad.
Aktorok	Óra
Forgatókönyv	Az autó a következő mezőre lép.

Use-case neve	Megáll
Rövid leírás	Az autó megáll.
Aktorok	Óra
Forgatókönyv	Az autó az aktuális mezőn marad.

1.2 Architektúra.

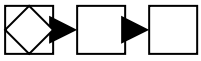
A fejlesztés során fontos a folyamatos tesztelés. Ez alól nem kivételek a use-case-ek sem. A szkeleton nélkülözhetetlen a programozás során: segítségével világosan és áttekinthetően tudjuk ellenőrizni, hogy a program futása közben a szekvencia diagramnak megfelelő módon követik-e egymást a metódusok. A teszt során előre megtervezett és megírt pályákat használhatunk, melyek segítségével szemléletesen tudjuk bemutatni az objektumok közötti kommunikációt.

A folyamatok egymás után futnak le, a végrehajtás egyszálú, ezért nem kell foglalkoznunk konkurrens folyamatokkal.

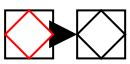
A pályák elkészítésekor törekedtünk az egyszerűségre, de figyeltünk arra is, hogy az eredeti célt ne tévesszük szem elől. Ebből adódóan igyekeztünk minden reprezentatív esetet figyelembe venni. A különböző tesztesetek, lényegében, különböző objektum konfigurációk, melyek segítségével „előhívhatóak” a szekvencia diagramok.

Megjegyzés a szemléltető grafikákhoz: az ábrákat kizárólag az osztályok közötti kapcsolat szemléltetése céljából készítettük, nem reprezentálják a játék grafikus kinézetét. A négyzetek mezőket jelképeznek, míg a bennük lévő rombuszok az autók (piros a rablók autója, kék a rendőr, fekete az átlagos autó).

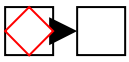
1. Tesztpálya:
   - Mezők száma: 3



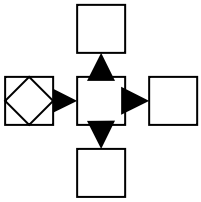
   - Mezők tartalma: Car, üres, üres.
   - Analóg esetek: PoliceCar, üres, üres.
   - Az első tesztpálya azt mutatja be, hogy megfelelő időközönként az autó hogyan lép.



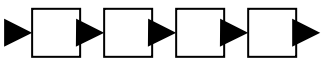
2. Tesztpálya:
   - Mezők száma:2
   - Mezők tartalma: RunnerCar, Car.
   - Analóg esetek: RunnerCar, PoliceCar.
   - Autó és a rablók autója összeütközik.
3. Tesztpálya:



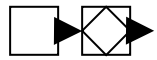
   - Mezők száma:2
   - Mezők tartalma: RunnerCar, üres (bázis)
   - Itt a rablók autója beérkezik a bázisra.
4. Tesztpálya:
   - Mezők száma:5



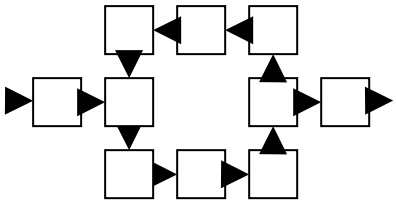
   - Mezők tartalma: Car, Üres, Üres, Üres, Üres
   - Analóg esetek: RunnerCar, Üres, Üres, Üres, Üres
   - Itt az átlag autók és kereszteződések működését vehetjük szemügyre. Ennek segítségével ellenőrizhetjük, hogy jól működik-e a kereszteződés a programunkban.



5. Tesztpálya:
   - Mezők száma: 4
   - Mezők tartalma: üres, üres,üres,üres
   - Azt vizsgáljuk, hogy időnként belépnek-e autók a városba.
6. Tesztpálya:



   - Mezők száma:2
   - Mezők tartalma: Car, üres.
   - Az autó elhagyja a várost.
7. Tesztpálya:



   - Mezők száma: 10
   - Mezők tartalma: Kezdetben mindegyik üres.
   - Autók keletkeznek és közlekednek a pályán.

1.3 A szkeleton kezelői felületének terve, dialógusok

A feladatkiírás alapján, a szkeleton változat célja, annak bizonyítása, hogy az objektum és dinamikus modellek a definiált feladat egy modelljét alkotják: A szkeleton egy program, amelyben már valamennyi, a végső rendszerben is szereplő business objektum szerepel. Az objektumoknak csak az interfésze definiált. Valamennyi metódus az indulás pillanatában az ernyőre szöveges változatban kiírja a saját nevét, majd meghívja azon metódusokat, amelyeket a szolgáltatás végrehajtása érdekében meg kell hívnia. Amennyiben a metódusból valamely feltétel fennállása esetén hívunk meg más metódusokat, akkor a feltételre vonatkozó kérdést interaktívan az ernyőn fel kell tenni és a kapott válasz alapján kell a továbbiakban eljárni.

Az általunk megvalósított szkeleton egy program váz, amelyet az eddigi tervek és modellek alapján készítünk el, abból a célból, hogy teszteljük azok helyességét. A szkeleton a metódushívásokról futási naplót készít a képernyőre (igény szerint fájlba), melyekkel a hívások sorrendje meghatározható. Kiírja az összes meghívott metódus nevét, paraméterlistáját, az őt tartalmazó objektum típusát és azonosítóját. A program a

felhasználóval is a konzolon keresztül kommunikál, például döntési helyzet esetén hozzá fordul és az általa adott választól függ a program futásának folytatása.

Célunk, hogy ezek alapján könnyen megállapítható legyen, hogy az előző fejezetben leírt szekvencia diagramok megvalósíthatók-e.

A szkeleton egy egyszerű választó képernyővel indul, felajánlva a lehetséges teszteseteket a felhasználónak.

Metódus hívás esetén a futási napló üzenetek kiírása a következő formában történik:

- Objektum neve
- Osztályneve
- Objektum hashcode-ja (integer típusú, Java specifikus objektum-azonosító)
- Metódus neve és paraméterlistája (az esetlegesen átadott objektumok típusa, neve, hashcode-ja)
- Hívás esetén CALL, visszatérés esetén RETURN
- Az interpretálást segítő megjegyzések „@”-al kezdődnek (pl. inicializálás kezdete/vége, stb.).

Példa:

```
> Car waiting counter reached zero? (y/n)
< y
> c|1011|: Car CALL n|2031|: Node -> getOutNodes();
> @Példa megjegyzés
> n|2031|: Node -> getOutNodes() RETURN outNodes|1024|: Node[];
> c|1011|: Car CALL n2|2033|: Node -> getTrafficLight();
> n2|2033|: Node -> getTrafficLight() RETURN trafficLight|91|: int;
> n2|2033|: Node -> getCar() RETURN car|90|: Car;
```

1.4 Szekvencia diagramok a belső működésre

A belső működést leíró szekvencia diagramok megtalálhatók az analízis modell dokumentáció 3.4-es pontjában.

1.5 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.03. 19:00	1 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet: A megvalósítandó fejezetek áttekintése. Eredménye: Brucker és Pocsai megvalósítják a beadandó első változatát.
2010.03.06. 15:00	1,5 óra	Brucker Pocsai	Értekezlet Skypeon keresztül, feladat megismerése, részletek megbeszélése.
2010.03.06. 21:00	1 óra	Brucker	Diagramok elkészítése.
2010.03.07. 17:00	2,5 óra	Brucker	A szkeleton kezelői felületének terve, dialógusok. Dokumentáció.
2010.03.07. 18:00	3 óra	Pocsai	Az Architektúra című fejezet kidolgozása. Tesztesetek megalkotása.
2010.03.07. 12:00	2 óra	Brucker Ferencz Pocsai	Értekezlet. Eddigi munka áttekintése. Hibakeresés. Javítanivalók kijelölése.
2010.03.09. 22:00	1 óra	Brucker	Use-case diagramok hibáinak javítása, use-case leírások kitöltése.
2010.03.10. 17:00	4 óra	Ferencz	Beadandó összeszerkesztése. Apróbb változtatások. Ellenőrzés, nyomtatás.
2010.03.10. 23:00	1 óra	Orovecz	Beadandó ellenőrzése. Nem talált hibát.

2. Szkeleton beadás

2.1 Fordítási és futtatási útmutató

2.1.1 Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
build.bat	120 byte	2010.03.17	Fordításra használt batch fájl
run.bat	76 byte	2010.03.17	Futtatásra használt batch fájl.
build_and_run.bat	145 byte	2010.03.17	Fordítás és futtatás.
src/Car.java	6 019 byte	2010.03.17	Osztály forráskódja.
src/CarFactory.java	2 326 byte	2010.03.17	Osztály forráskódja.
src/City.java	16 444 byte	2010.03.17	Osztály forráskódja.
src/Controller.java	3 666 byte	2010.03.17	Osztály forráskódja.
src/Game.java	4 540 byte	2010.03.17	Osztály forráskódja.
src/Node.java	4 777 byte	2010.03.17	Osztály forráskódja.
src/PoliceCar.java	530 byte	2010.03.17	Osztály forráskódja.
src/RunnerCar.java	5673 byte	2010.03.17	Osztály forráskódja.
src/Timer.java	1 789 byte	2010.03.17	Osztály forráskódja.

2.1.2 Fordítás

Az egyszerűsítés, és a hibák elkerülésének érdekében a fordítást a *build.bat* nevű batch fájl végzi. A java útvonal beállítása után (pl. set PATH=„c:\Program Files\Java\jdk1.6.0_18\bin”) célszerű végrehajtani. A fordítás eredménye a */bin* könyvtárban lesz látható: elkészülnek a *.class fájlok.

Az előállításhoz szükséges feltétel a legalább 1.6-os verziójú Java JDK kit megléte.

Lehetőség van a Fordítás és a Futtatás együttes elvégzésére a *buil_and_run.bat* segítségével.

2.1.3 Futtatás

A futtatás egy batch fájl elindításával lehetséges (*run.bat*), természetesen csak az után, hogy az alkalmazás hiba nélkül lefordításra került.

Futtatáskor a program egy menüvel indít. A menü pontjai különböző tesztpályákat indítanak. A tesztpályákat úgy alkottuk meg, hogy minden szekvencia diagram ellenőrizhető legyen a segítségükkel (megfelelően megválaszolva a program által feltett kérdéseket). A futtatás során ajánlom, hogy az előző fejezetben leírt tesztesetek „térképét” használja a felhasználó, mivel ezek alapján könnyen tesztelhető a szkeleton.

A szkeleton, a felhasználó dolgát megkönnyítendő, jelzi a szekvenciák elejét és végét. A diagramokat megismételtük ennek a fejezetnek a végén, mivel néhány apróbb változtatást eszközöltünk.

2.2 Értékelés

Az elmúlt hetekben a csapat megtanult együtt dolgozni, és kialakultak a különböző szerepek, ez nagyban egyszerűsítette a közös munkát. Kezdetben kissé szokatlan volt, de rövid időn belül sikerült a tagoknak alkalmazkodni a csapatmunka aspektusaihoz. A legfontosabb lépés a megfelelő kommunikáció kialakulása volt, hiszen tulajdonképp ez az egyéni- és a csapatmunka közti legszembetűnőbb különbség. Szerencsére ezt a lépést probléma nélkül sikerült megtenni.

Már előre sejtettük, hogy a közös munka nem lesz zökkenőmentes, ha egy csapattag hiányzása miatt a többiek nem tudnak a hiányzó tag munkájához hozzáférni, és úgy általában gondot fog jelenteni, ha nem tudja mindenki bármely időpontban a projekt egy részét elérni. Erre a problémára készített Ferencz egy SVN oldalt, amely mindenki számára elérhető a nap bármely időpontjában. Az eddigi munka során ez igen nagy segítség volt.

A jó kommunikációnak köszönhetően csupán ritkán fordultak elő apróbb hibák, melyeket gyorsan javítottunk is. Ennek eredményeként a végleges munkatermékek jól sikerültek.

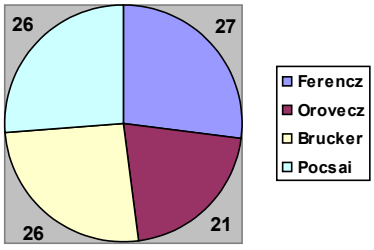
A munka megszervezése és a feladatok kiosztása fontos tényező volt, mivel a csapattagok különböző feladatok ellátásában voltak igazán hatékonyak. Ez az idő múlásával alakult ki, ahogy az egyes tagok végeztek saját feladatrészlegükkel, és közösen megvizsgáltuk az eredményeket. Ezáltal a munkavégzés minősége és sebessége jelentősen javult. Ez nagyon lényeges, mivel az eddig elvégzett munka az alapja mindennek, és merőben befolyásolja a végeredményt.

A végzett munkaórák százalékos eloszlását korrigáltuk az általunk megállapított szubjektív tényezőkkel (munka pontossága, nehézsége, határidő, stb.) és ebből adódott a végső százalék.

A csapatban végzett munkaórák eloszlása:

Ferencz: 38,75 óra, 32,22%
Orovecz: 23,50 óra, 19,54%
Brucker: 30,50 óra, 25,36%
Pocsai: 27,50 óra, 22,88%
Összesen: 120,25 óra, 100%

Tag neve	Munka százalékban
Ferencz Endre (PEBU1F)	27%
Orovecz Ferenc (B0HUPW)	21%
Brucker Amanda Mária (UHZMDX)	26%
Pocsai Csaba (BOYKMJ)	26%



2.3 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.11. 15:00	3 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet a szkeleton megvalósításáról. Orovecz elkészíti a program vázát és a menüt.
2010.03.11. 20:00	3 óra	Orovecz	Tevékenység: A program váz és a menü elkészítése.
2010.03.12. 12:00	2 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet a további megvalósításról. Ferencz elkészíti az első változatot.
2010.03.12. 19:00	5 óra	Ferencz	Szkeleton első változatának elkészítése.
2010.03.13. 20:00	3 óra	Ferencz	Szkeleton második változatának elkészítése.
2010.03.14. 14:00	2 óra	Brucker Orovecz	Étrekezlet a szkeletonról. Javítanivalók leszögezése.
2010.03.14. 18:00	2 óra	Pocsai	Szekvencia diagramok kiegészítése / javítása.
2010.03.14. 20:00	2 óra	Ferencz	Beadandó megfogalmazása / megszerkesztése
2010.03.16. 19:00	1 óra	Brucker	Beadandó átnézése.
2010.03.16. 20:00	1 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet a szkeleton működéséről.
2010.03.17. 16:00	2 óra	Ferencz	Hibák javítás.
2010.03.17. 24:00	0,5 óra	Orovecz	Nyomtatás.

3. Prototípus koncepciója

3.1 Prototípus interface-definíciója

A feladatkiírás szerint a prototípus program célja annak demonstrálása, hogy a program elkészült, helyesen működik, valamennyi feladatát teljesíti. A prototípus változat egy elkészült program kivéve a kifejlett grafikus interfészt. A változat tervezési szempontból kész, az ütemezés, az aktív objektumok kezelése meg van oldva. A business objektumok - a megjelenítésre vonatkozó részeket kivéve - valamennyi metódusa a végleges algoritmusokat tartalmazza.

A prototípus program kezelése konzolon keresztül fog történni. A teszteléshez előre meghatározott teszteseteket készítünk. Ezekkel kapcsolatos alapelv, hogy azok átláthatóak, megismerhetőek és reprodukálhatóak legyenek. Csapatunk ezeket szem előtt tartja, és ezeknek megfelelően állítja elő a prototípust és a tesztfelületet.

Ahhoz, hogy a tesztesetek reprodukálhatók legyenek, ki kell küszöbölni a véletlen faktort, mely ugyan változatossá teszi a végső játékot, de a tesztelés során elfogadhatatlan az, hogy nem ellenőrizzük le a véletlen esemény összes lehetséges kimenetelét. Éppen ezért lehetőség lesz arra, hogy minden egyes lépés determinisztikus legyen.

3.1.1 Az interfész általános leírása

A prototípus program a szabványos be- és kimeneten kommunikál a felhasználóval. Parancsokat kap a szabványos bemenetről vagy előre elkészített parancsfájlból. A tesztelendő pályákat fájlból tölti be (bővebben 7.1.2).

A program minden egyes parancs végrehajtásakor kiírja a szabványos kimenetre a parancs eredményét, de a pályát csak a felhasználó külön kérésére rajzolja ki. Ezzel elkerülhető, hogy a konzolos kijelzőn történő kirajzolás nehézségei miatt ugráljon a kép és túl gyakran túl sok mindent kelljen kirajzolni. A felhasználó kérheti az aktuális pálya-állapot fájlba történő mentését is. Ez lényeges szempont az ellenőrzésnél, mivel így bizonyos fokig egyszerűsíthető a tesztelők feladata.

3.1.2 Bemeneti nyelv

A pályának vannak statikus és, működés közben állapotot változtató, dinamikus elemei. Statikus elemnek számítanak például az utak, csomópontok, jelzőlámpák elhelyezkedése, a bank, a rejtekhely. Ezeket minden egyes tesztesetnél egy, a pályát leíró, fájl tartalmazza és ugyanazon teszteset minden betöltésekor azonos helyet foglalnak el a pályán.

A dinamikus elemek közé tartoznak a különböző típusú autók és a húsvéti nyuszi. Ezek elhelyezése többféleképpen is történhet. Elhelyezheti őket a felhasználó egyesével, tetszőleges helyre, mindenből, amennyit szeretne. Ilyenkor a véletlenszerűség ki van kapcsolva. Azonban ez meglehetősen hosszadalmas és bonyolult, valamint a véletlenszerűségnek is tesztelhetőnek kell lennie. Ezért minden olyan esetben, amikor a felhasználó (a bemeneti fájl) nem ad véletlen elemeket specifikáló parancsokat, a játék nem determinisztikus módon folytatja menetét.

Az alkalmazásunk egyszálú, ezért a teszteléshez nem szükséges az események időbeni viszonyát vizsgálni.

loadCommands

Leírás: beolvassa és végrehajtja az `input_commands.txt`-ben megadott parancsokat. Ez használható pálya betöltésére is.

Opciók: *`loadCommands`* `input_commands.txt`

addNode

Leírás: új csomópontot ad a pályához.
Opciók: `addNode id type hiding`, ahol az első paraméter az egyedi azonosító, a második a csomópont típusa (lámpás – 0, különben 24 féle elsőbbségi sorrend képzelhető el „2031” alakban megadva), az utolsó pedig meghatározza, hogy az adott csomópont rejtekhely-e.

connect

Leírás: összeköt két már meglévő csomópontot.
Opciók: `connect node1 node2 dir1 dir2`, ahol az első két paraméter adja meg az összekötendő csomópontok egyedi azonosítóját, a második kettő pedig a kapcsolódás irányát. A kapcsolat egyirányú.

addCar

Leírás: hozzáadja az autót a paraméterként megadott csomóponthoz.
Opciók: `addCar id type strength speed node`, ahol az első paraméter az autó egyedi azonosítóját, a második az autó típusát (0 – átlagos autó, 1 – rendőrautó, 2 - nyúl), a harmadik az életerejét, a negyedik az autó sebességét (nyuszi feltűnésének időtartamát), míg az utolsó a csomópontot tartalmazza.

addRobberCar

Leírás: hozzáadja az autót a paraméterként megadott csomóponthoz.
Opciók: `addRobberCar strength node`, a paraméterek: életerő, a csomópont egyedi azonosítója.

moveRobberCar

Leírás: Lépteti a rablók autóját.
Opciók: `moveRobberCar dir`, a paraméterben megadott irányba lépteti a rablók autóját.

setNextMove

Leírás: a paraméterként beadott autó (első paraméter) következő lépését (második paraméter) állítja be (a véletlen faktor kiküszöbölése céljából).
Opciók: `setNextMove id dir`

tickCar

Leírás: A megadott autónak küld egy „tick”-et.
Opciók: `tickCar id`

tick

Leírás: lépteti az időt, ezáltal a pályán lévő összes időzített elem „lép” egyet. Tesztesetekben nem célszerű használni, mivel az elemek következő lépése nem determinisztikus.
Opciók: nincs.

setTrafficLight

Leírás: a paraméterben megadott sorszámú közlekedési lámpát, a megadott irányra állítja.
Opciók: `setTrafficLight node dir`

writeToConsole

Leírás: kiírja a pálya aktuális állapotát a standard kimenetre. A pálya állapota alatt azon parancsok sorozatát értjük, melyekkel felállítható az aktuális helyzet. Bizonyos információk ilyenkor természetesen elveszhetnek, mint például az autók esetében a következő lépésig való várakozási idő. (ld. Példa a 7.1-es fejezet végén)
Opciók: nincs.

writeToFile

Leírás: kiírja a pálya aktuális állapotát a felhasználó által megadott `output.txt`-be. A pálya állapota alatt azon parancsok sorozatát értjük, melyekkel felállítható az aktuális helyzet. Bizonyos információk ilyenkor természetesen elveszhetnek, mint például az autók esetében a következő lépésig való várakozási idő. (ld. Példa a 7.1-es fejezet végén)
Opciók: *`writeToFile`* `output.txt`

reset

Leírás: Alapállapotba állítja a prototípust.
Opciók: nincs.

help

Leírás: Kiírja a felhasználható utasításkészletet.
Opciók: nincs.

exit

Leírás: Kilépés. A teszt véget ér.
Opciók: nincs.

Az irányok definiálása az adott csomóponthoz viszonyítva:

0 – Észak (Fel)
1 – Kelet (Jobbra)
2 – Dél (Le)
3 – Nyugat (Balra)

Csapatunk úgy találta célszerűnek, hogy nem definiálunk külön nyelvet a pálya beolvasásához, hanem a megfelelő parancsok sorozata során épülne fel a pálya. Ezt a döntést azzal indokoljuk, hogy ezáltal az implementáció sokkal egyszerűbbé válik, csökken a hibák lehetősége (ez felbecsülhetetlen) és a bővítések bevezetése is egyszerűbbé válik.

3.1.3 Kimeneti nyelv

A sikeresen végrehajtott parancsoknál a program nem ír ki semmit a képernyőre. Amennyiben a végrehajtás sikertelen, a következő hibaüzenetek jelenhetnek meg:

Hibaüzenetek:

FileNotFound: Nem olvasható a megadott bemeneti fájl.
FileWriteError: Nem írható a megadott kimeneti fájl.
CommandNotFound: Nem létező parancs érkezett.
ParameterMismatch: Valamilyen gond van a parancs paraméterezésével.
NodeNotFound: Nincs a pályán a megadott azonosítóval rendelkező csomópont.

CarNotFound: Nincs a pályán a megadott azonosítóval rendelkező autó.
IdOccupiedError: A megadott egyedi azonosító már foglalt.
NodeOccupiedError: A megadott csomópont foglalt.
DirOccupiedError: A megadott csomópontnak ez a kimeneti/bemeneti iránya már foglalt.

Két parancs szolgál arra, hogy a modell állapotát lekérdezzük: *writeToConsole* és *writeToFile*. Ezek előállítják a legrövidebb olyan parancssorozatot, melyekkel a pálya aktuális állapota előállítható. Ha belegondolunk, ez nem bonyolult, mivel a város statikus elemei könnyen kiírhatók, a dinamikus elemek pedig az *addCar* paranccsal azonnal hozzáadhatók. Az egyetlen megoldandó kihívás az, hogy ezek a parancsok egy elvárt sorrendben generálódjanak. Erre a problémára a következő szabályt találtuk ki: a csomópontok azonosítójuk sorrendjében jönnek létre, majd a kapcsolatok (az első és a második kapcsolatuk alapján növekvő sorrendbe), utána az autók (szintén növekvő sorrendbe) és legvégül a rablók autója.

Példa egy autó lépésére:

Teszt bemenete	Standard kimenet

`addNode 1 1 false`
`addNode 2 24 false`
`addCar 1 0 100 1`
`setNextMove 1 1`
`tick`
`writeToFile Output.txt`	

	Output.txt

	`addNode 1 1 false`
	`addNode 2 24 false`
	`addCar 1 0 100 2`

3.2 Összes részletes use-case

A részletes (fizikai) use-casek megalkotásánál figyelembe vettük azt, hogy mindegyik use-casehez tartozzon egy jól definiált parancs is.

Use-case neve	Játék indítása
Rövid leírás	A játék akkor indul el, amikor a rablók autója megjelenik a pályán.
Aktorok	A játékos.
Forgatókönyv	A rablók autója megjelenik a pályán.

Use-case neve	Pálya választása
Rövid leírás	A játékos pályát választ.
Aktorok	A játékos.
Forgatókönyv	A pálya betöltődik

Use-case neve	Autó belép a városba.
Rövid leírás	Új autó érkezik a városba.
Aktorok	Timer.
Forgatókönyv	CarFactory autót generál.

Use-case neve	Húsvéti Nyuszi megjelenik
Rövid leírás	A húsvéti nyuszi néha feltűnik a pályán.
Aktorok	Timer
Forgatókönyv	A húsvéti nyuszi megjelenik a pályán.

Use-case neve	Karakter mozgatása
Rövid leírás	A játékos irányítja a rablók autóját üres vagy foglalt mezőre.
Aktorok	A játékos.
Forgatókönyv	A karakter mozog, majd az esetleges ütközést lekezeljük. A karakter ütközhet autóval, rendőrautóval és a nyuszival. Ezek az esetek külön kerülnek feldolgozásra.

Use-case neve	Autók mozgtása
Rövid leírás	Minden autó lép a pályán.
Aktorok	Timer
Forgatókönyv	A város autói lépnek.

Use-case neve	Jelzőlámpák léptetése
Rövid leírás	A jelzőlámpák átállnak.
Aktorok	Timer
Forgatókönyv	Minden jelzőlámpa átáll a következő érvényes állapotára.

3.3 Tesztelési terv

A teszteléshez előre elkészítjük a bemeneti és az elvárt kimeneti fájlokat. A tesztelés során megvizsgáljuk, hogy a játék a megfelelő eredményt produkálja-e. Amennyiben jó a kimenet, a tesztet sikeresnek tekintjük, különben megpróbáljuk behatárolni a hiba okát és kijelölünk egy felelőst, aki a hiba elhárítását elvégzi.

Teszt-eset neve	Pálya beolvasása és kiírása
Rövid leírás	Az alkalmazás beolvas, majd kiír egy adott pályát (csomópontokkal, utakkal és autókkal).
Teszt célja	Leellenőrzi, hogy a város jól épül fel, a csomópontok közötti kapcsolat megfelelően létrejön.

Teszt-eset neve	Játékos lépeget
Rövid leírás	Egy kellően összetett pályán a rablók autója meghatározott útvonalon megy végig, majd leellenőrizzük a lépések követésének helyességét.
Teszt célja	A pálya konzisztenciájának leellenőrzése, valamint a RobberCar lépéseinek követése.

Teszt-eset neve	Autók közlekednek.
Rövid leírás	Egy közepesen bonyolult pályán az autók meghatározott lépéssorozatot járnak végig, miközben új autók is keletkezhetnek.
Teszt célja	A különböző autók mozgásának ellenőrzése.

Teszt-eset neve	Játékos és rendőr
Rövid leírás	Itt csak egy egyszerű pályára lesz szükség. Megvizsgáljuk a rablók autójának és a rendőrautónak az ütközését.
Teszt célja	A rendőr viselkedésének vizsgálata.

Teszt-eset neve	Játékos és autók
Rövid leírás	Itt csak egy egyszerű pályára lesz szükség. Megvizsgáljuk a rablók autójának az átlagos autókkal való ütközését. A végén a játékos elhagyja a várost (veszít).
Teszt célja	A rendőr viselkedésének vizsgálata.

Teszt-eset neve	A nyuszi
Rövid leírás	Egy egyszerű pályán megvizsgáljuk a nyuszi által adott előny helyes működését.
Teszt célja	A nyuszi viselkedésének vizsgálata.

Teszt-eset neve	A játék
Rövid leírás	Egy bonyolult pályán az autók, a rendőrautók és a rablók autója közlekedik, ütközik. Időnként nyuszik is megjelenhetnek. A végén a játékos beér a bázisra.
Teszt célja	A rendőr viselkedésének vizsgálata.

3.4 Tesztelést támogató segéd- és fordítóprogramok specifikálása

A teszteléshez nem kívánunk különösebb segédeszközt felhasználni. A bemeneti és kimeneti állományokat manuálisan fogjuk előállítani.

A tesztelés menetének könnyítése érdekében tervezzük egy egyszerűbb tesztelő program írását, amely csak annyit tud, hogy előállítja a program kimenetét az összes megadott bemenetre és összeveti az előre elvárt állományokkal.

3.5 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.18. 15:00	1 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet a prototípusról. Néhány elképzelés megvalósíthatóságának megvitatása. Feladatok leosztása: Brucker 7.1, Pocsai 7.2, Orovecz:7.3,7.4.
2010.03.19. 10:00	1 óra	Orovecz	Tesztesetek megalkotása.
2010.03.19. 12:00	1 óra	Orovecz Ferencz	Értekezlet a tesztesetekről. Eredménye: több teszteset szükséges.
2010.03.20. 19:00	4 óra	Orovecz	7.3 és 7.4 fejezetek megírása, ellenőrzése, elküldése.
2010.03.20. 20:00	2 óra	Pocsai	Use-case-ek elkészítése.
2010.03.21. 18:15	1 óra	Pocsai Brucker	Értekezlet a use-case-ek és az utasítások összefüggéseiről.
2010.03.22. 13:00	2 óra	Pocsai	7.2-es fejezet elkészítése, véglegesítése.
2010.03.22. 18:00	2 óra	Ferencz	Az eddig elkészített feladatok ellenőrzése, átnézése.
2010.03.23. 18:15	1 óra	Brucker	A feladat megismerése.
2010.03.23. 20:40	2,5 óra	Brucker	7.1, 7.1.1, 7.1.2, 7.1.3 pontok elkészítése.
2010.03.23. 23:50	0,75 óra	Brucker	Az általam elkészített feladatrészek áttekintése, javítása, továbbküldés a csapatkapitánynak.
2010.03.24. 17:00	5 óra	Ferencz	Néhány lényeges módosítás az utasításokban és a use-case-ekben. Néhány teszteset pontosítása. Ellenőrzés. Nyomtatás.

4. Részletes tervek

4.1 Osztályok és metódusok tervei.

4.1.1 Controller

- **Felelősség**

Az osztály felelőssége az, hogy összekösse a felhasználót a prototípus mögött lévő modellel. Ez az osztály dolgozza fel a beérkező parancsokat, meghívja a modell megfelelő metódusát.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - -game: **GAME**
- **Metódusok**
  - **-void interpret(String command)**: Ez metódus végzi el a parancsok feldolgozását.
  - **+void main(String[] args)**: A program belépési pontja.

4.1.2 Game

- **Felelősség**

A Game osztályon keresztül érhető el a modell minden funkciója. Ez az osztály tartja számon a játék paramétereit. A prototípusban még nem olyan lényeges elem, mint amilyen a végső grafikus változatban lesz.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **+random: Random**: Minden osztály ezt a statikus adattagot használja a véletlen elemek generálásához.
  - **-carFactory: CarFactory**: Az autók „gyártását” végző attribútum.
  - **-city:City**: A városra mutató adattag.
  - **-timer: Timer:** Az időzítést végző osztályra mutató adattag.
  - **-gameState: int**: A játék állapotát tárolja (játék indítására vár, játék folyamatban, stb.).

- **Metódusok**
  - **+Game()**: konstruktor.
  - **+addCar(int, int, int, int, int)**: addCar parancshoz tartozó metódus.
  - **+addNode(int, int, boolean)**: addNode parancshoz tartozó metódus.
  - **+addRobber(int, int)**: addRobber parancshoz tartozó metódus.
  - **+connect (int, int, int, int):** connect parancshoz tartozó metódus.
  - **+endGame():** Akkor kerül meghívásra, ha a játékos valamilyen okból elvesztette a játékot (prototípusban lényegi funkciót nem valósít meg).
  - **+moveRobberCar(int):** moveRobberCar parancshoz tartozó metódus.
  - **+setNextMove(int, int):** setNextMove parancshoz tartozó metódus.
  - **+setTrafficLight(int, int):** a setTrafficLight parancshoz tartozó metódus.
  - **+startGame():** Akkor kerül meghívásra, ha a játékos elkezdi a játékot (prototípusban lényegi funkciót nem valósít meg).
  - **+tick():** A tick parancshoz tartozó metódus.
  - **+tickCar():** A tickCar parancshoz tartozó metódus.
  - **+winGame():** Akkor kerül meghívásra, ha a játékos megnyeri a játékot (prototípusban lényegi funkciót nem valósít meg).
  - **+writeOut(PrintStream Out):** A kiíró típusú parancsok meghívása esetén fut le.

4.1.3 City

- **Felelősség**

A City osztály tárolja és kezeli a pálya összes elemét (autók, csomópontok).

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **-cars: TreeMap<Integer, Car>:** Az autókat tároló adatstruktúra.
  - **-game: Game**: a game osztályra mutató referencia.
  - **-nodes: TreeMap <Integer, Car>:** A csomópontokat tároló adatstruktúra.
  - **-runnerCar: RunnerCar:** A játékos által irányított autóra mutató referencia.

- **Metódusok**
  - **+City(Game)**: Az osztály konstruktora.
  - **+addNode(Node, Integer):** Új csomópontot helyez el a pályán.
  - **+connect(int, int, int, int):** Összeköt két, már meglévő csomópontot.
  - **+getCar(int): Car**: Visszatér a paraméterként megadott azonosítóval rendelkező autóval.
  - **-getEmptyNodes(): LinkedList<Node>:** Privát metódus belső használatra. CarFactory által generált autó elhelyezésére szolgál.
  - **-getEntryNodes(): LinkedList<Node>:** Privát metódus. Visszatér az üres mezők listájával. A húsvéti nyúl elhelyezéséhez szükséges.
  - **+getFreeIdForCar(): int:** Visszaad egy, még szabad azonosítót.
  - **+getGame(): Game:** A game attribútum getter függvénye.

- **+getRunnerCar():** A runnerCar attribútum getter függvénye.
- **+newBunny(Bunny):** Új húsvéti nyulat helyez a pályára (véletlen faktorral).
- **+newCar(Car):** Új autót helyez a pályára (véletlen faktorral).
- **+newCar(Car, int):** Új autót helyez a megadott sorszámú csomópontra.
- **+newRunnerCar(RunnerCar):** Elhelyezi a rablók autóját a pályán (véletlen faktorral).
- **+newRunnerCar(RunnerCar, int):** Elhelyezi a rablók autóját a megadott sorszámú csomópontra.
- **+removeCar(Car):** Eltávoltítja a megadott autót a városból.
- **+removeRunnerCar():** Eltávolítja a rablók autóját a városból.
- **+setTrafficLight(int, int):** Beállítja a megadott sorszámú csomópontban lévő lámpát a megadott irányra.
- **+tick():** A város összes eleme lép (véletlen faktorral).
- **+writeOut(PrintStream):** Kiírja a városban lévő összes elemet.

4.1.4 Timer

- **Felelősség**

A timer osztály felel a megfelelő időközönként bekövetkező impulzusokért. A prototípusban nincs lényeges funkciója.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **-game: Game:** A game osztályra mutató referencia.
  - **-period int**: Az impulzusok közötti idő hosszát tárolja.

- **Metódusok**
  - **+Timer()**: Az osztály konstruktora.
  - **+setGame(Game)**: Beállítja a game osztályra mutató referenciát.
  - **+setPeriod(int)**: Beállítja az impulzusok közötti időintervallum hosszát.
  - **+tick():** Csak a prototípusban használt függvény a „tick”-ek generálásához.

4.1.5 Node

- **Felelősség**

A Node osztály felel az éppen benne elhelyezkedő autó tárolásáért, valamint azért, hogy átadja a kellő információkat az autóknak ahhoz, hogy megfelelően tudjanak közlekedni.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **-car: Car**: A tárolt autóra mutató referencia.
  - **-city: City:** A városra mutató referencia.
  - **-hiding: Boolean:** Ennek a változónak az állásától függően lehet ez a csomópont a rablók egyik bázisa.
  - **-id: int:** Ez az adattag tárolja a csomópont azonosítóját.
  - **-inNodes: Node[]:** Azokat a csomópontokat tárolja ez a vektor, amelyekkel bejövő kapcsolatban áll az aktuális csomópont.
  - **-nextTrafficLight: int:** Ezt az adattagot a felhasználó állíthatja be. Az adattag változása után rögtön beáll a tényleges jelzőlámpa is. Ennek a változónak csak akkor van jelentősége, ha a felhasználó nem létező állapotot akar beállítani.
  - **-outNodes: Node[]:**Azokat a csomópontokat tárolja ez a vektor, amelyekkel kimenő kapcsolatban áll az aktuális csomópont.
  - **-priority: LinkedList<Integer>:** Ez az adattag tárolja a csomóponthoz tartozó elsőbbségi sorrendet.
  - **-trafficLight: int:** A jelzőlámpa állását tároló változó.

- **Metódusok**
  - **+Node(int, City, int, Boolean)**: A Node konstruktora. A paraméterezése megegyezik az addNode parancséval.
  - **+activeInNodes(): int:** A metódus megszámolja, hogy hány érvényes bejövő kapcsolata van a csomópontnak.
  - **+activeOutNodes(): int:** A metódus megszámolja, hogy hány érvényes kimenő kapcsolata van a csomópontnak.
  - **+connect(Boolean, int, Node):** A metódus beállítja a megfelelő kapcsolatot a csomópontban.
  - **+getCar(): Car:** A car attribútum getter függvénye.
  - **+getCity(): City:** A city attribútum getter függvénye.
  - **+getDir(Node): int:** Visszatér a megadott Node-hoz való kapcsolat irányával.
  - **+getId(): int:** Az id attribútum getter függvénye.
  - **+getInNodes(): Node[]:** az inNodes attribútum getter függvénye.
  - **+getOutNodes(): Node[]:** az outNodes attribútum getter függvénye.
  - **+getPriority(): LinkedList<Integer>:** a priority attribútum getter függvénye.
  - **+getTrafficLight(): int:** A trafficLight attribútum getter függvénye.
  - **+getType(): int:** Visszatér a kereszteződés típusával.
  - **+isHiding(): Boolean:** A hiding attribútum getter függvénye.
  - **+setCar(Car):** A car attribútum setter függvénye.
  - **+setTrafficLight(int):** Beállítja a jelzőlámpát a megadott irányra.
  - **+step():** Lépteti a jelzőlámpát.

4.1.6 CarFactory

- **Felelősség**

Az autók generálásáért felelős osztály.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **-city: City**: a városra mutató referencia

- **Metódusok**
  - **+createCar(int, int, int, int, int)**: Autó létrehozása. Paraméterezése megegyezik az addCar parancséval.
  - **+createRunnerCar(int, int)**: A rablók autójának létrehozása. Paraméterezése megegyezik a createRunnerCar parancséval.
  - **+setCity(City)**: A city attribútum setter függvénye.
  - **+Tick()**: Autó létrehozása véletlen faktorral.

4.1.7 Car

- **Felelősség**

Az osztály felelőssége, hogy megvalósítsa egy autó működését.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**
  - **#city: City**: A városra mutató referencia.
  - **#id: int**: Az autó egyedi azonosítóját tároló adattag.
  - **#nextMove**: **int:** A következő lépést tároló adattag,
  - **#node: Node:** A tartalmazó csomópontra mutató referencia.
  - **#speed: int:** az autó sebessége.
  - **#strength: int:** az autó ereje.
  - **#wait: int:** belső adattag, mely a kivárandó impluzusok számát tárolja.

- **Metódusok**
  - **+Car(int, int, int, City)**: Az osztály konstruktora.
  - **+collision (RunnerCar)**: A rablóval való ütközés lekezelését végző függvény.
  - **+destroy():** Az autó megszüntetését végző metódus. Kitörli az osztály példányára szóló referenciákat.
  - **+getId(): int:** Az id attribútum getter függvénye.
  - **+getNode(): Node:** A node attribútum getter függvénye.
  - **+getSpeed(): int:** A speed attribútum getter függvénye.
  - **+getStrength(): int:** A strength attribútum getter függvénye.
  - **+getType(): int:** Visszatér az autó típusával.
  - **-givePriority(Node): boolean:** Belső függvény, csak az átláthatóság kedvéért készült.
  - **+setCity(City): void:** Beállítja a városra mutató referenciát.
  - **+setNextMove(int): void:** Beállítja az autó következő lépését.

- **+setNode(Node): void:** Beállítja a tartalmazó csomópontra mutató referenciát.
- **+step(): void:** Lépteti az autót.

4.1.8 PoliceCar

- **Felelősség**

Az osztály felelőssége, hogy megvalósítsa a modellben a rendőrautó működését,

- **Ősosztályok**

Car

- **Interfészek**

Nincs

- **Metódusok**
  - **+PoliceCar(int, int, int, City):** Az osztály konstruktora.
  - **+collision(RunnerCar): void:** Az ütközést lekezelő (callbacket megvalósító) függvény.
  - **+getType(): int:** Visszaadja az autó típusát.
  - **+step(): void:** Lépteti a rendőrautót.

4.1.9 RunnerCar

- **Felelősség**

Az osztály felelőssége, hogy megvalósítsa a modellben a játékost, azaz a rablók autóját.

- **Ősosztályok**

Car

- **Interfészek**

Nincs

- **Attribútumok**
  - **-bunnyPower: int**: A húsvéti nyúl által adott erő. Ez minden lépésnél csökken.

- **Metódusok**
  - **+RunnerCar(int, City):** Az osztály konstruktora.
  - **+collisionWithBunny(Bunny):** Az ütközést lekezelő függvény.
  - **+collisionWithCar(Car):** Az ütközést lekezelő függvény.
  - **+collisionWithPolice(PoliceCar):** Az ütközést lekezelő függvény.
  - **+destroy():** Kitörli a rablók autójára mutató referenciát és értesíti a Game osztályt a játék végéről.
  - **+step(int):** Lépteti a rablók autóját a paraméterként kapott irányba.

4.1.10 Bunny

- **Felelősség**

Az osztály felelőssége, hogy megvalósítsa a modellben a húsvéti nyulat, azaz a rablók autójának védelmet nyújtó objektumot.

- **Ősosztályok**

Car

- **Interfészek**

Nincs

- **Metódusok**
  - **+Bunny(int, int, int, City):** Az osztály konstruktora.
  - **+Collision(RunnerCar):** Az ütközést lekezelő (callbacket megvalósító) függvény.
  - **+getType():int:** Visszaadja az autó típusát.
  - **+step():** Lépteti a nyulat, ami a megjelenésének idejét csökkenti.

4.2 A tesztek részletes tervei, leírásuk a teszt nyelvén

A tesztesetek megalkotásánál a következő szempontokat vettük figyelembe:

- Térkép: Minden tesztpályához tartozik egy térkép, ami vizuálisan szemlélteti a pálya alakját és a rajta elhelyezkedő szereplőket illetve fontosabb csomópontokat.
- Bemenet: A kiadott parancsok, melyek lefuttatják a tesztet és rögzítik az eredményt.
- Elvárt kimenet: A tesztek helyes működése során elvárt eredmények.

Csapatunk úgy határozott, hogy a tesztparancsokat egy-egy előre definiált szövegfájlban helyezi el, így nem szükséges azokat mindig egyesével megadni. Természetesen a parancsok manuálisan is megadhatók.

Jelmagyarázat:

 Üres mező

 Bankrabló

Rendőr

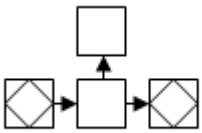
Búvóhely

 Átlagos autó

 Húsvéti nyúl

4.2.1 1. Teszteset

- Leírás
  Az első pálya funkciója igen egyszerű, létrehoz egy kis pályát, néhány úttal, egy csomóponttal és két autóval. A cél az, hogy ellenőrizzük az utak/csomópontok pontos létrejöttét, hogy helyesen vannak-e összekötve, valamint, hogy az autók létrejönnek.



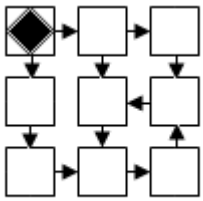
- Ellenőrzött funkcionalitás, várható hibahelyek

Lehetséges hibák lehetnek: a csomópontok paraméterei változnak, a csomópontok összeköttetése változik/megszűnik, autók eltűnnek, stb.

Bemenet	Elvárt kimenet
addNode 0 1230 false addNode 1 1230 false addNode 2 1230 false addNode 3 1230 false connect 1 2 1 3 connect 2 0 0 2 connect 2 3 1 3 addCar 0 0 10 2 1 addCar 1 0 10 2 3 writeToFile test1_out.txt	addnode 0 1230 false addnode 1 1230 false addnode 2 1230 false addnode 3 1230 false connect 1 2 1 3 connect 2 0 0 2 connect 2 3 1 3 addcar 0 0 10 2 1 addcar 1 0 10 2 3

4.2.2 2. Teszteset

- Leírás
  A második tesztben azt ellenőrizzük le, hogy a rabló, azaz a játékos végigmegy-e pontosan azon az úton, amit előre meghatároztunk, tehát a rabló viselkedésére vagyunk kíváncsiak.

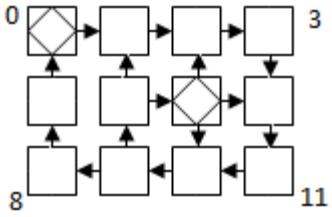


- Ellenőrzött funkcionalitás, várható hibahelyek
  Ez a teszteset ellenőrzi le, hogy a rablók autója helyesen követi a bejövő parancsokat. Lehetséges hibák: a rablók autója máshol helyezkedik el a teszt végén, a rablók autója eltűnik, esetleg megduplázódik.

Bemenet	Elvárt kimenet
addNode 0 1230 false	addnode 0 1230 false
addNode 1 1230 false	addnode 1 1230 false
addNode 2 1230 false	addnode 2 1230 false
addNode 3 1230 false	addnode 3 1230 false
addNode 4 1230 false	addnode 4 1230 false
addNode 5 1230 false	addnode 5 1230 false
addNode 6 1230 false	addnode 6 1230 false
addNode 7 1230 false	addnode 7 1230 false
addNode 8 1230 false	addnode 8 1230 false
connect 0 1 1 3	connect 0 1 1 3
connect 0 3 2 0	connect 0 3 2 0
connect 1 2 1 3	connect 1 2 1 3
connect 1 4 2 0	connect 1 4 2 0
connect 2 5 2 0	connect 2 5 2 0
connect 3 6 2 0	connect 3 6 2 0
connect 4 7 2 0	connect 4 7 2 0
connect 5 4 3 1	connect 5 4 3 1
connect 6 7 1 3	connect 6 7 1 3
connect 7 8 1 3	connect 7 8 1 3
connect 8 5 0 2	connect 8 5 0 2
addRobberCar 10 0	addrobbercar 10 4
moveRobberCar 1	
moveRobberCar 2	
moveRobberCar 2	
moveRobberCar 1	
moveRobberCar 0	
moveRobberCar 3	
writeToFile test2_out.txt	

8.2.1 3. Teszteset

- Leírás
  Ez egy összetettebb pálya az eddigiekhez képest. Kezdetben van két autó a pályán, az egyik (0) a 0-s, a másik(1) a 6-os számú node-on. Az 1-es számú kocsi először északra megy, majd a 0-s megpróbál kétszer kelet felé menni, de csak egyszer tud, mivel nem ütközhet az 1-es autóval. Ezután elindul az 1-es, és tud utána menni a 0-ás is. Az előre meghatározott lépéseknek úgy lesz vége, hogy a 0-ás a 3-as node-on áll, az 1-es pedig a 11-esen. Ekkor létrejön még egy autó(2) a 0-ás node-on, melynek a sebessége fele az eddigieknek. Az utolsó tick-re az új kocsi nem mozdul a sebessége miatt, de az 1-es kocsinál be kell állítani, hogy melyik irányba menjen, mert ő az egyetlen, aki elér egy olyan csomópontot, ahol nem egyértelmű, melyik irányt választja (9-es).



- Ellenőrzött funkcionalitás, várható hibahelyek
  A teszt célja, hogy megvizsgálja az autók viselkedését minden lehetséges elhelyezésben. A hiba leginkább az autók végső elhelyezkedésében lehet.

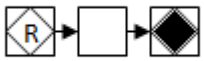
Bemenet	Elvárt kimenet
addNode 0 1230 false	addnode 0 1230 false
addNode 1 3210 false	addnode 1 3210 false
addNode 2 3210 false	addnode 2 3210 false
addNode 3 1230 false	addnode 3 1230 false
addNode 4 1230 false	addnode 4 1230 false
addNode 5 1230 false	addnode 5 1230 false
addNode 6 1230 false	addnode 6 1230 false
addNode 7 0123 false	addnode 7 0123 false
addNode 8 1230 false	addnode 8 1230 false
addNode 9 1230 false	addnode 9 1230 false
addNode 10 1230 false	addnode 10 1230 false
addNode 11 1230 false	addnode 11 1230 false
connect 0 1 1 3	connect 0 1 1 3
connect 1 2 1 3	connect 1 2 1 3
connect 2 3 1 3	connect 2 3 1 3
connect 3 7 2 0	connect 3 7 2 0
connect 4 0 0 2	connect 4 0 0 2
connect 5 1 0 2	connect 5 1 0 2
connect 5 6 1 3	connect 5 6 1 3
connect 6 2 0 2	connect 6 2 0 2
connect 6 7 1 3	connect 6 7 1 3
connect 6 10 2 0	connect 6 10 2 0
connect 7 11 2 0	connect 7 11 2 0
connect 8 4 0 2	connect 8 4 0 2
connect 9 5 0 2	connect 9 5 0 2
connect 9 8 3 1	connect 9 8 3 1
connect 10 9 3 1	connect 10 9 3 1
connect 11 10 3 1	connect 11 10 3 1
addCar 0 0 10 1 0	addcar 0 0 10 1 10

just_another_team

addCar 1 0 10 1 6 setNextMove 0 1 setNextMove 1 0 tickCar 1 tickCar 0 tickCar 0 tickCar 1 tickCar 0 tickCar 1 tickCar 0 tickCar 1 addCar 2 0 10 2 0 tickCar 0 tickCar 1 tickCar 2 tickCar 0 tickCar 1 tickCar 2 setNextMove 1 0 tickCar 0 tickCar 1 tickCar 2 writeToFile test3_out.txt	addcar 1 0 10 1 5 addcar 2 0 10 2 1

8.2.2 4. Tesztpálya

- Leírás
  A negyedik teszt ismét egy egyszerű teszt, itt azt vizsgáljuk, hogy ha a rendőr rálép a rablóra, akkor elkapja-e.



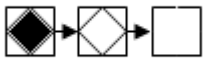
- Ellenőrzött funkcionalitás, várható hibahelyek
  A legfontosabb ellenőrzött funkció az ütközés kezelése. Azt várjuk, hogy a rablók autója megszűnik, de hibák esetén sok más kimenet előfordulhat.

Bemenet	Elvárt kimenet
addNode 0 1230 false addNode 1 1230 false addNode 2 1230 false connect 0 1 1 3 connect 1 2 1 3 addCar 0 1 10 1 0 addRobberCar 15 2 tickCar 0 tickCar 0 writeToFile test4_out.txt	addnode 0 1230 false addnode 1 1230 false addnode 2 1230 false connect 0 1 1 3 connect 1 2 1 3 addcar 0 1 10 1 2

4.2.3 5. Tesztpálya

- Leírás

  Az ötödik pályán a rabló átlagos kocsival való ütközését vizsgáljuk. A lépések végén a rablók autója elhagyja a várost.



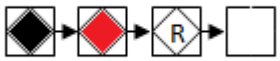
- Ellenőrzött funkcionalitás, várható hibahelyek

  Lehetséges hibák: nem a rablók autója marad meg az ütközés után; a rablók autója nem hagyja el a várost; ütközés után nem semmisül meg az átlagos autó.

Bemenet	Elvárt kimenet
addNode 0 1230 false addNode 1 1230 false addNode 2 1230 false connect 0 1 1 3 connect 1 2 1 3 addCar 0 0 10 1 1 addRobberCar 15 0 moveRobberCar 1 moveRobberCar 1 moveRobberCar 1 writeToFile test5_out.txt	addnode 0 1230 false addnode 1 1230 false addnode 2 1230 false connect 0 1 1 3 connect 1 2 1 3

4.2.4 6. Tesztpálya

- Leírás
  A tesztpálya lényege, hogy a rabló elüt egy húsvéti nyulat, és amiatt immunitást élvez a rendőrök ellen, tehát egy ütközés során sem tudják őt elkapni.

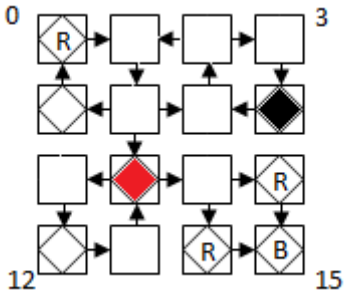


- Ellenőrzött funkcionalitás, várható hibahelyek
  A teszt során a rablók autójának az interakcióját vizsgáljuk meg a húsvéti nyúllal. Ebből kifolyóan a lehetséges hibák: a húsvéti nyúl nem ad plusz erőt, nem tűnik el a húsvéti nyúl.

Bemenet	Elvárt kimenet
addNode 0 1230 false addNode 1 1230 false addNode 2 1230 false addNode 3 1230 false connect 0 1 1 3 connect 1 2 1 3 connect 2 3 1 3 addCar 0 2 10 1 1 addCar 1 1 10 1 2 addRobberCar 15 0 moveRobberCar 1 moveRobberCar 1 moveRobberCar 1 writeToFile test6_out.txt	addnode 0 1230 false addnode 1 1230 false addnode 2 1230 false addnode 3 1230 false connect 0 1 1 3 connect 1 2 1 3 connect 2 3 1 3 addrobbercar 15 3

4.2.5 7. Tesztpálya

- Leírás
  Ez a pálya a legösszetettebb. Itt egy egész játék megy végbe, mivel a pályán előfordulnak az átlagos autók, a rendőrautók és a rabló. A rabló a végén a rejtekhelyre lép, tehát nyer (itt csak annyi látszik, hogy eltűnik).



- Ellenőrzött funkcionalitás, várható hibahelyek
  Lehetséges hibahelyek: a rablók autója nem tűnik el a bázison, nem oda érkeznek az autók, ahová terv szerint kell jutniuk.

Bemenet	Elvárt kimenet
addNode 0 1230 false	addnode 0 1230 false
addNode 1 3210 false	addnode 1 3210 false
addNode 2 1230 false	addnode 2 1230 false
addNode 3 1230 false	addnode 3 1230 false
addNode 4 1230 false	addnode 4 1230 false
addNode 5 1230 false	addnode 5 1230 false
addNode 6 1230 false	addnode 6 1230 false
addNode 7 1230 false	addnode 7 1230 false
addNode 8 1230 false	addnode 8 1230 false
addNode 9 1230 false	addnode 9 1230 false
addNode 10 1230 false	addnode 10 1230 false
addNode 11 1230 false	addnode 11 1230 false
addNode 12 1230 false	addnode 12 1230 false
addNode 13 1230 false	addnode 13 1230 false
addNode 14 1230 false	addnode 14 1230 false
addNode 15 1230 true	addnode 15 1230 true
connect 0 1 1 3	connect 0 1 1 3
connect 1 5 2 0	connect 1 5 2 0
connect 2 1 3 1	connect 2 3 1 3
connect 2 3 1 3	connect 2 1 3 1
connect 3 7 2 0	connect 3 7 2 0
connect 4 0 0 2	connect 4 0 0 2
connect 5 4 3 1	connect 5 6 1 3
connect 5 6 1 3	connect 5 9 2 0
connect 5 9 2 0	connect 5 4 3 1
connect 6 2 0 2	connect 6 2 0 2
connect 7 6 3 1	connect 7 6 3 1
connect 8 12 2 0	connect 8 12 2 0
connect 9 8 3 1	connect 9 10 1 3
connect 9 10 1 3	connect 9 8 3 1
connect 10 11 1 3	connect 10 11 1 3
connect 10 14 2 0	connect 10 14 2 0
connect 11 15 2 0	connect 11 15 2 0
connect 12 13 1 3	connect 12 13 1 3
connect 13 9 0 2	connect 13 9 0 2
connect 14 15 1 3	connect 14 15 1 3
addCar 0 0 5 1 4	addcar 1 0 10 1 12
addCar 1 0 10 1 12	addcar 2 1 10 1 9

just_another_team

addCar 2 1 10 1 0 addCar 3 1 10 1 11 addCar 4 1 10 1 14 addRobberCar 20 7 moveRobberCar 3 tickCar 2 tickCar 0 moveRobberCar 0 tickCar 2 tickCar 0 moveRobberCar 3 setNextMove 2 1 tickCar 2 moveRobberCar 2 tickCar 2 addCar 5 2 10 1 9 moveRobberCar 2 setNextMove 2 3 tickCar 2 moveRobberCar 1 tickCar 2 moveRobberCar 1 setNextMove 2 2 tickCar 2 moveRobberCar 2 writeToFile test7_out.txt	addcar 4 1 10 1 14

4.3 A tesztelést támogató programok tervei

A tesztelés során egyetlen segédeszközt szeretnénk használni. Ez a segédeszköz (a diff utasításhoz hasonlóan) kiírja a két szöveges állomány közötti eltéréseket. Ezzel a programmal a tesztelés folyamata nagyrészt automatizálható.

Példa: diff test_out.txt test_expected_output.txt
Error at line 3: „`addCar 0 0 5 1 4`”, „`addCar 0 0 5 1 3`”

4.4 Változások a prototípusban használt utasítások között

A konzultáció után egy új utasítás integrálása mellett döntöttünk:

tickCar

Leírás: A megadott autónak küld egy „tick”-et.
Opciók: *`tickCar`* `id`

Ezáltal az utasításkészlet a következők szerint alakul:

```
loadCommands input_commands.txt
addNode id type hiding
connect node1 node2 dir1 dir2
addCar id type strength speed node
addRobberCar strength node
moveRobberCar dir
setNextMove id dir
setTrafficLight node dir
tickCar id
tick
writeToConsole
writeToFile output.txt
help
exit
```

4.5 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.25. 18:00	3 óra	Brucker Ferencz Pocsai Orovecz	Megbeszélés a teendőkről. Eredménye: Orovecz kidolgozza a teszteseteket, Ferencz pedig a prototípus osztályait.
2010.03.26. 12:00	5 óra	Ferencz	Prototípus tervezése.
2010.03.26. 14:00	4 óra	Orovecz	Tesztesetek tervezése.
2010.03.29. 17:00	1,5 óra	Brucker Pocsai Orovecz	Konzultáció
2010.03.29. 19:00	1 óra	Brucker Pocsai	Értekezlet az utasításokról. Eredménye: új utasítást kell bevezetni: addCar id
2010.03.30. 8:00	0,5 óra	Brucker Ferencz	Értekezlet az új utasításról, valamint a meglévők áttekintése.
2010.03.30. 17:00	3 óra	Pocsai	Prototípus osztályainak áttekintése, ellenőrzése.
2010.03.30. 20:00	2 óra	Brucker	Prototípus osztályainak dokumentálása.
2010.03.30. 22:00	3 óra	Orovecz	Tesztesetek véglegesítése.
2010.03.31. 19:00	4 óra	Ferencz	Beadandó anyag véglegesítése.

1. Prototípus beadása

1.1 Fordítási és futtatási útmutató

1.1.1 Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
src\controller\Controller.java	11014 byte	2010.04.12	Forráskód.
src\model\Bunny.java	671 byte	2010.04.12	Forráskód.
src\model\Car.java	3815 byte	2010.04.12	Forráskód.
src\model\CarFactory.java	1605 byte	2010.04.12	Forráskód.
src\model\City.java	7895 byte	2010.04.13	Forráskód.
src\model\Game.java	2787 byte	2010.04.13	Forráskód.
src\model\Node.java	3953 byte	2010.04.12	Forráskód.
src\model\PoliceCar.java	2014 byte	2010.04.12	Forráskód.
src\model\RunnerCar.java	2468 byte	2010.04.12	Forráskód.
src\model\Timer.java	635 byte	2010.04.12	Forráskód.
test_expected_out\test1.txt	175 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test2.txt	402 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test3.txt	599 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test4.txt	117 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test5.txt	98 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test6.txt	156 byte	2010.04.09	Elvárt kimenet.
test_expected_out\test7.txt	767 byte	2010.04.09	Elvárt kimenet.
test_in\test1.txt	218 byte	2010.04.09	Bemenet.
test_in\test2.txt	547 byte	2010.04.09	Bemenet.
test_in\test3.txt	879 byte	2010.04.09	Bemenet.
test_in\test4.txt	199 byte	2010.04.09	Bemenet.
test_in\test5.txt	228 byte	2010.04.09	Bemenet.
test_in\test6.txt	286 byte	2010.04.09	Bemenet.
test_in\test7.txt	1172 byte	2010.04.09	Bemenet.
build.bat	156 byte	2010.04.11	Lefordítja a forráskódot.
run.bat	96 byte	2010.04.09	Futtatja a prototípust.
test.bat	468 byte	2010.04.11	Lefuttatja a teszteket.
check.bat	447 byte	2010.04.11	Ellenőrzi a tesztek kimenetét.
build_test_check.bat	91 byte	2010.04.11	Elvégzi a teszteléshez szükséges összes műveletet.
cmp.exe	152064 byte	2010.04.11	Összehasonlít két szöveges állományt.

1.1.2 Fordítás

Az egyszerűsítés, és a hibák elkerülésének érdekében a fordítást a *build.bat* nevű batch fájl végzi. A java útvonal beállítása után (pl. set PATH=„c:\Program Files\Java\jdk1.6.0_18\bin”) célszerű végrehajtani. A fordítás eredménye a */bin* könyvtárban lesz látható: elkészülnek a *.class fájlok.

Az előállításhoz szükséges **feltétel a legalább 1.6-os verziójú Java JDK kit megléte**.

1.1.3 Futtatás

A futtatás egy batch fájl elindításával lehetséges (*run.bat*), természetesen csak az után, hogy az alkalmazás hiba nélkül lefordításra került. A tesztelés elvégzéséhez rendelkezésre áll még a *test.bat*, amely lefuttatja a prototípuson a bemeneti fájlokat, valamint a *check.bat,* amely összehasonlítja a kapott kimeneti állományokat az elvárt kimenettel.

1.1.4 Gyors tesztelés

Lehetőség van az összes lépés gyors elvégzésére a *build_test_check.bat* segítségével, de ebben az esetben is be kell állítani a java elérési útvonalát. A tesztelés után érdemes megnézni a keletkezett kimeneteket és a felhasznált bemenetet is.

1.2 Tesztek jegyzőkönyvei

1.2.1 Teszteset1

Tesztelő neve	Orovecz
Teszt időpontja	2010.04.07 17:00

1.2.2 Teszteset2

Tesztelő neve	Orovecz
Teszt időpontja	2010.04.07 17:00

1.2.3 Teszteset3

Tesztelő neve	Orovecz
Teszt időpontja	2010.04.07 17:00

1.2.4 Teszteset4

Tesztelő neve	Pocsai
Teszt időpontja	2010.04.06 19:00

1.2.5 Teszteset5

Tesztelő neve	Pocsai
Teszt időpontja	2010.04.06 19:00

1.2.6 Teszteset6

Tesztelő neve	Brucker
Teszt időpontja	2010.04.08 12:00

1.2.7 Teszteset7

Tesztelő neve	Brucker
Teszt időpontja	2010.04.08 13:00

Megismételt tesztek:

Tesztelő neve	Orovecz
Teszt időpontja	2010.04.07 17:00
Teszt eredménye	Sikertelen. Eltérés: addNode 7 0123 false addNode 7 123 false
Lehetséges hibaok	Kiíratás hibás.
Változtatások	A hiba egyszerűen javítható volt.

Tesztelő neve	Brucker
Teszt időpontja	2010.04.08 13:00
Teszt eredménye	Sikertelen. Eltérés: A rendőr nem lép rá a rablóra.
Lehetséges hibaok	Hibás feltétel.
Változtatások	A hiba gyorsan javítható volt.

Meglepett minket a hibák (nagyon) alacsony száma, de természetesen örültünk annak, hogy az alapos tervezés és az összetett szkeleton elkészítése után a prototípus már nem jelentett különösebb kihívást csapatunknak.

1.3 Értékelés

Az elmúlt időszakban a csapatunk összeszokott, és a már kialakult szerepek segítségével gyorsabban és hatékonyabban folyt a munkavégzés. A kezdeti nehézségek után sikerült a tagoknak alkalmazkodni a csapatmunka aspektusaihoz, ami nem volt könnyű feladat. A legfontosabb lépés a megfelelő kommunikáció kialakulása volt, hiszen tulajdonképpen ez az egyéni- és a csapatmunka közti legszembetűnőbb különbség. Szerencsére ezt a lépést probléma nélkül sikerült megtenni, hála a különböző számítógépes programoknak, melyek segítségével sikerült olyankor is eredményesen konzultálni, mikor akár több száz km-re voltunk egymástól.

Már a projekt elején észrevettük, hogy jó volna, ha létrehoznánk egy a munkát segítő tárhelyet, amelyre fel tudjuk tölteni a különböző részeit a projektnek. Erre a problémára készített Ferencz egy SVN oldalt, amely mindenki számára elérhető a nap bármely időpontjában. Az eddigi munka során ez igen nagy segítség volt.

Fentebb említettem, hogy nagyon fontosnak tartjuk a jó kommunikációt tagok között. Ennek eredményeképp egyre kevesebbszer fordultak elő hibák, melyeket igyekeztünk gyorsan javítani. Időnként konzultációkat szerveztünk különböző problémák megvitatására, megoldások keresésére. Az ilyen alkalmak kapcsán érezhettük a csapatmunka igazi előnyét: sok alkalommal sikerült egymásnak jó ötletekkel szolgálni, és így a feladatmegoldás is gyorsabban ment. Ennek eredményeként a végleges munkatermékek eredményesen sikerültek.

Hamar rájöttünk, hogy a projekt szervezett munkavégzést igényel, hiszen nem vagyunk egyformák, mindenki másféle feladatban jeleskedett. Ez az idő múlásával alakult ki, ahogy az egyes tagok végeztek saját feladatrészlegükkel, és közösen megvizsgáltuk az eredményeket, ellenőriztük egymás munkáját. Ezáltal a munkavégzés minősége a többszöri ellenőrzés következtében jelentősen javult. Ez nagyon lényeges, mivel az eddig elvégzett munka merőben befolyásolja a végső eredményt.

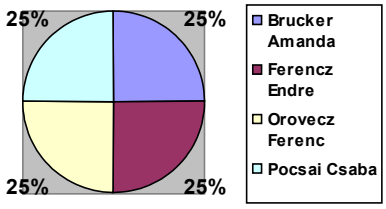
just_another_team

A végzett munkaórák százalékos eloszlását korrigáltuk az általunk megállapított szubjektív tényezőkkel (munka pontossága, nehézsége, határidő, stb.) és ebből adódott a végső százalék.

A csapatban végzett munkaórák eloszlása:

Ferencz: 27.5 óra, 26,63%
Brucker: 24.75 óra, 23.97%
Orovecz: 25.50 óra, 24.69%
Pocsai: 25,50 óra, 24.69%
Összesen: 103,25 óra, 100%

Tag neve	Értékelés
Brucker Amanda Mária (UHZMDX)	25%
Ferencz Endre (PEBU1F)	25%
Orovecz Ferenc (B0HUPW)	25%
Pocsai Csaba (BOYKMJ)	25%



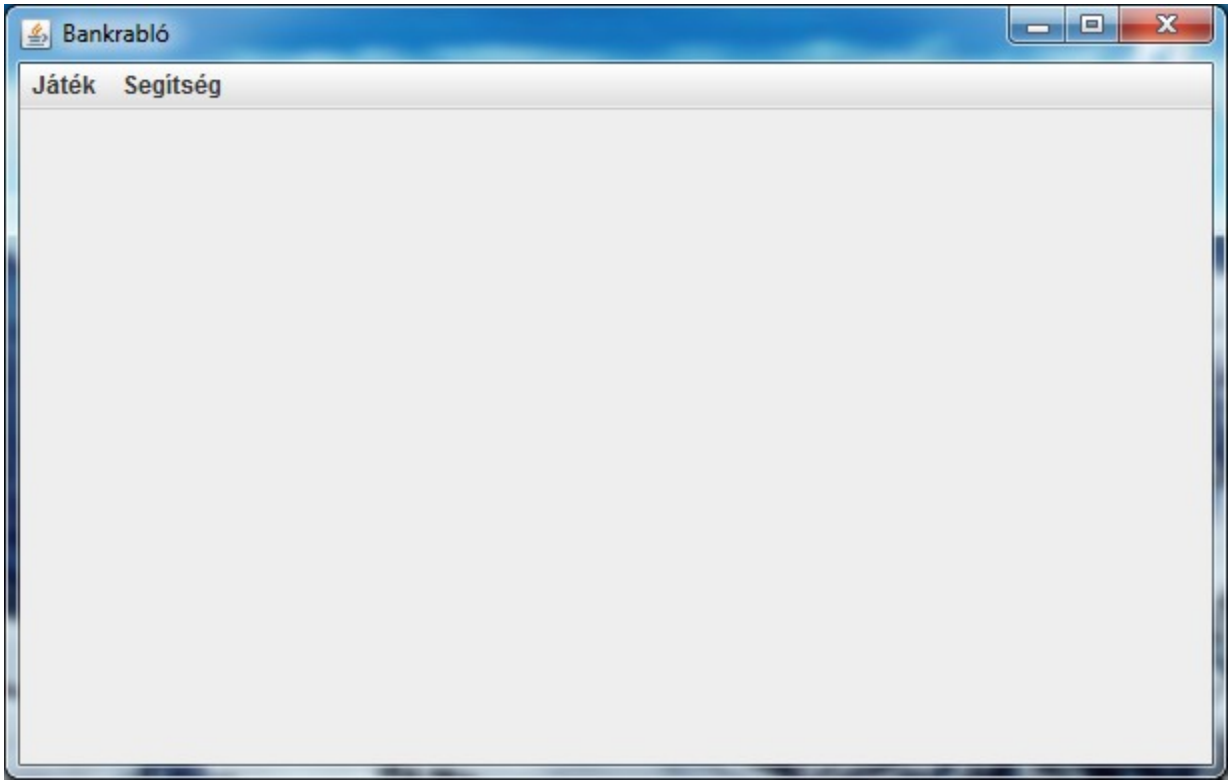
1.4 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.30. 18:00	4 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet. Döntés: Brucker megvalósítja a Controller, Bunny, Car, CarFactory osztályokat. Pocsai megvalósítja a City, Game, Node osztályokat. Orovecz pedig a PoliceCar, RunnerCar és a Timer osztályokat a megbeszéltek alapján.
2010.03.31. 19:00	6 óra	Brucker	Tevékenység: Az osztályok implementálása.
2010.03.31. 18:00	3 óra	Pocsai	Tevékenység: Az osztályok implementálása.
2010.04.01. 19:00	2,5 óra	Pocsai	Tevékenység: Az osztályok implementálása.
2010.04.03. 19:00	2 óra	Orovecz	Tevékenység: Az osztályok implementálása.
2010.04.05. 19:00	2 óra	Ferencz	Az osztályok ellenőrzése, az integráció elvégzése, apróbb hibák javítása.
2010.04.06. 19:00	0,5 óra	Pocsai	Teszteset elvégzése.
2010.04.07. 17:00	1 óra	Orovecz	Teszteset elvégzése.
2010.04.08. 12:00	0,25 óra	Brucker	Teszteset elvégzése.
2010.04.08. 13:00	0,25 óra	Brucker	Teszteset elvégzése.
2010.04.13. 19:00	1 óra	Pocsai	Beadandó összeállítása. Nyomtatás.

2. Grafikus felület specifikációja

2.1 A grafikus interfész

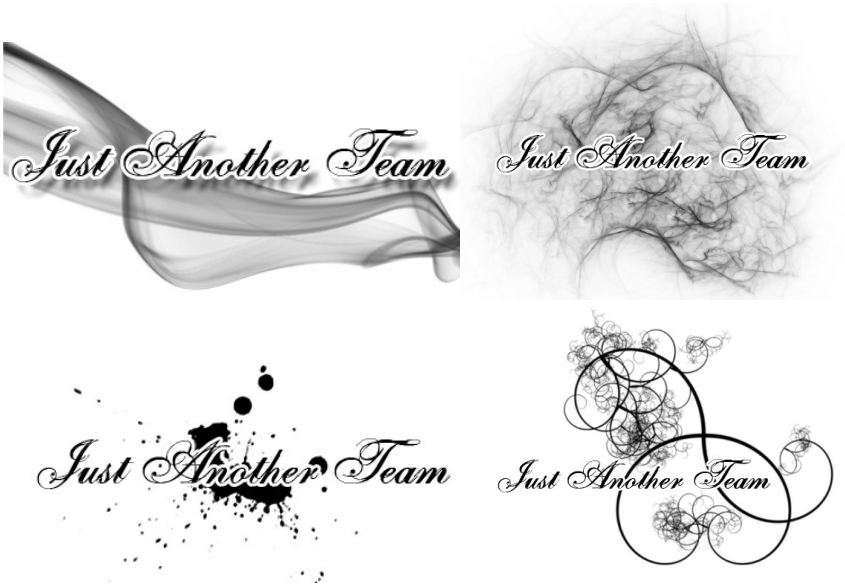
Célunk egy olyan grafikus környezet megvalósítása, mely a felhasználók számára könnyen átlátható, ugyanakkor funkciókban gazdag és felemelő játékélményt nyújt.



A program egy rövid menüt tartalmaz, melynek két fő menüpontja van: *Játék* és *Segítség*. A *Játék* menüpont alatt a következő funkciókat lehet elérni: *Új játék*, *Szünet*, *Kilépés*. A *Segítség* gombra való kattintás rögtön aktiválja a beépített súgót, mely egy általános leírást tartalmaz a kezelőfelület használatáról.

Készítettünk néhány látványtervet a csapat logójával, a felhasznált ikonokkal kapcsolatban:

just_another_team



A kezelőfelület legfőbb része maga a játéktér, amelyre kirajzolódik *Verkehrsmeldungen am Uml* térképe, a rajta közlekedő autókkal, rendőrökkel és az esetlegesen megjelenő húsvéti nyúllal. A pálya szélső részein néhány, a játék szempontjából releváns, adat lesz látható, mint például az eltelt idő, megmaradt életerő, stb.

2.2 A grafikus rendszer architektúrája

2.2.1 A felület működési elve

A tervezés során lényeges szempont, hogy megfelelően elkülönüljön a modell, a megjelenítés és a vezérlés. Ahhoz, hogy ez az elkülönülés kihangsúlyozott legyen az alkalmazásunkat három csomagra (package) osztottuk. A „model” elnevezésű a modell összes osztályát, a „controller” az irányításhoz szükséges logikát, míg a „view” a megjelenítéshez szükséges elemeket tartalmazza.

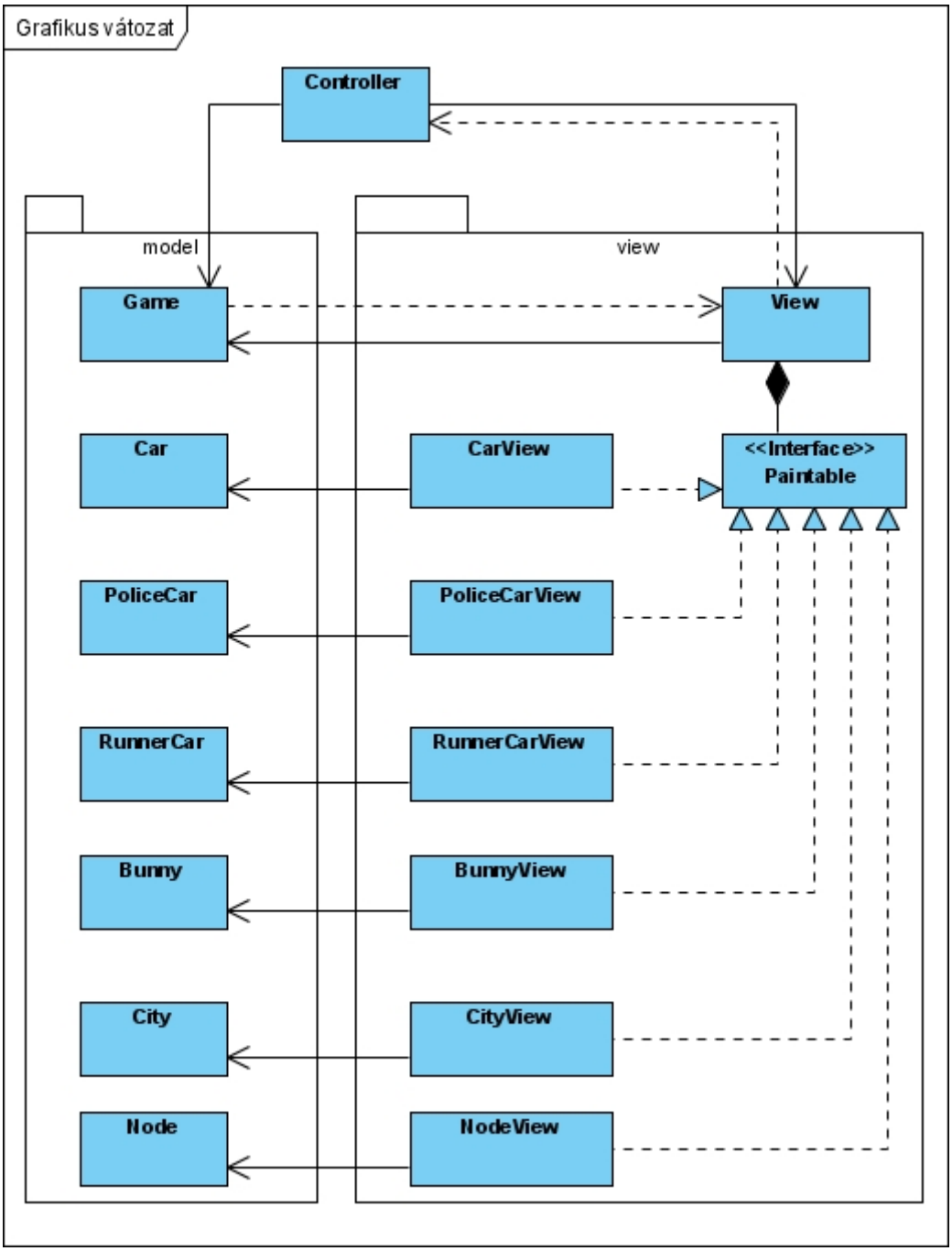
Csapatunk a push alapú megoldás alkalmazását tartotta célszerűnek, tehát a modell értesíti a felületet, hogy változás történt és megtörténik az újrarajzolás. A modell ütemes változása miatt esett erre a választásunk, mivel ez egyszerűen megvalósítható, de mégis robosztus megoldás.

Szeretnénk kihasználni a Java nyelv által nyújtott modern lehetőségeket, ezért a megjelenítést a Swing grafikus komponensekre építjük.

2.2.2 A felület osztály-struktúrája

Ahhoz, hogy a megjelenítés minél jobban elkülönüljön a modelltől, szükségesnek láttuk új osztályok bevezetését. A játék során hat osztály esetében van szükség arra, hogy valamilyen módon kapcsolatban álljanak a felhasználói felülettel. Ezek a következők: Game, City, Node, Car, PoliceCar, Bunny, RunnerCar. A hozzájuk kapcsolódó új osztályok (megfelelő sorrendben): View, CityView, NodeView, CarView, PoliceCarView, BunnyView, RunnerCarView.

Ez a kapcsolat tükröződik a következő statikus osztálydiagramon, melyen csak a grafikus (view) réteg szempontjából lényeges osztályokat és kapcsolatokat tüntettük fel. Ahhoz, hogy a diagram átlátható maradjon, az osztályok attribútumait és metódusait nem ábrázoltuk. Ezek, részletes leírásukkal a következő fejezetben találhatóak.



4. ábra Statikus osztálydiagram

2.3 A grafikus objektumok felsorolása

2.3.1 View

- **Felelősség**

A View osztály felelőssége, hogy biztosítsa a megjelenítési környezetet a tartalmazott komponensek számára. Ez az osztály gondoskodik arról, hogy megfelelő időpontban és sorrendben meghívja a játék összes megjelenítéssel kapcsolatos objektum kirajzoló függvényét.

- **Ősosztályok**

Nincs (JPanel)

- **Attribútumok**
  - **-game: Game** - Game osztályra mutató referencia.
  - **-components: LinkedList<Paintable> -** tartalmazott objektumok.
  - **-nodeViews: TreeMap<Integer, NodeView> -** a csomópontokat külön is letároljuk, hogy a koordinátákat ki tudjuk nyerni az autók számára.

- **Metódusok**
  - **+repaint() –** Az ősosztályból örökölt metódus. A felület frissítését végzi. Indirekt módon meghívja a paintComponent() metódust.
  - **+paintComponent(Graphics g)** – Ez a metódus végzi a felület tényleges újrarajzolását. Az összes tartalmazott objektum paint() metódusát meghívja, megfelelő sorrendben és időben.
  - **+setGame(Game game)** – setter függvény
  - **+createBunnyView(Bunny bunny)** – összekapcsol egy modellbeli nyulat a view rétegbeli „nyúlmegjelenítővel”.
  - **+createCarView(Car c, int type)** – összekapcsol egy modellbeli autót egy View rétegbeli autó megjelenítőjével.
  - **+createPoliceCarView(PoliceCar c) –** összekapcsol egy modellbeli rendőrautót egy View rétegbeli rendőrautó megjelenítőjével.
  - **+createRunnerCarView(RunnerCar c) -** összekapcsol egy modellbeli rablóautót egy View rétegbeli rabló autó megjelenítőjével.
  - **+createCityView(City city, int level)** – összekapcsolja a modellbeli várost egy View rétegbeli várossal.
  - **+createNodeView(Node node, int posX, int posY, int angle) – –** összekapcsolja a modellbeli várost egy View rétegbeli csomóponttal.
  - **+getNodeViews():TreeMap<Integer, NodeView> -** getter függvény.

2.3.2 Paintable

- **Felelősség**

Interfész, mely egységesíti a kirajzolandó objektumok kezelését.

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.

2.3.3 CarView

- **Felelősség**

Az átlagos autók grafikus megjelenítésért felelős osztály

- **Interfészek**

Paintable

- **Attribútumok**
  - **-car: Car** – a megjelenítő objektumhoz kapcsolt autó
  - **-view: View** – a View objektumra mutató referencia
  - **-type: int –** az autó típusa.

- **Metódusok**
  - **+paint(Graphics g) -** metódus, mely elvégzi az adott komponens kirajzolását.
  - **+CarView(Car car, View view, int type) -** konstruktor.

2.3.4 PoliceCarView

- **Felelősség**

A rendőrautók grafikus megjelenítésért felelős osztály.

- **Interfészek**

Paintable

- **Attribútumok**
  - **-policeCar: PoliceCar**: ehhez a megjelenítő objektumhoz kapcsolt rendőrautó.
  - **-view: View** – a View objektumra mutató referencia

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.
  - **+PoliceCarView(PoliceCar policeCar, View view) -** konstruktor.

2.3.5 RunnerCarView

- **Felelősség**

A rablók autójának grafikus megjelenítésért felelős osztály.

- **Interfészek**

Paintable

- **Attribútumok**
  - **-runnerCar: RunnerCar** - ehhez a megjelenítő objektumhoz kapcsolt autó.
  - **-view: View** – a View objektumra mutató referencia

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.
  - **+RunnerCarView(RunnerCar runnerCar, View view) -** konstruktor.

2.3.6 BunnyView

- **Felelősség**

A húsvéti nyúl grafikus megjelenítésért felelős osztály.

- **Interfészek**

Paintable

- **Attribútumok**
  - **-bunny: Bunny -** ehhez a megjelenítő objektumhoz kapcsolt nyúl.
  - **-view: View** – a View objektumra mutató referencia

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.
  - **+BunnyView(Bunny bunny, View view) -** konstruktor.

2.3.7 CityView

- **Felelősség**

A város megjelenítésért felelős osztály.

- **Interfészek**

Paintable

- **Attribútumok**
  - **-city: City** - ehhez a megjelenítő objektumhoz kapcsolt város.
  - **-view: View** – a View objektumra mutató referencia

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.
  - **+CityView(City city, View view, int level):** konstruktor.

2.3.8 NodeView

- **Felelősség**

A csomópont (lámpa, STOP tábla) megjelenítésért felelős osztály.

- **Interfészek**

Paintable

- **Attribútumok**
  - **-node: Node** - ehhez a megjelenítő objektumhoz kapcsolt csomópont.
  - **-view: View** – a View objektumra mutató referencia

- **Metódusok**
  - **+paint(Graphics g)**: metódus, mely elvégzi az adott komponens kirajzolását.
  - **+NodeView(Node node, View view, int posX, int posY, int angle) -** konstruktor.
  - **+getPosX(): int** – getter függvény
  - **+getPosY(): int** – getter függvény
  - **+getAngle(): int** – getter függvény

2.3.9 Controller

- **Felelősség**

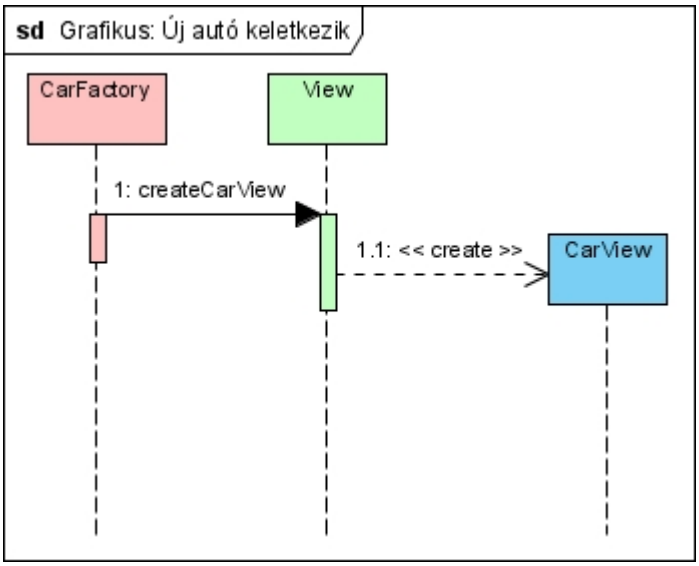
A játék vezérléséért felelős osztály. Továbbítja a modell számára a felhasználó felől érkező bemeneteket.

- **Attribútumok**
  - **-frame: JFrame**: A játék fő ablaka.
- **Metódusok**
  - **+main()**: A program belépési pontja.

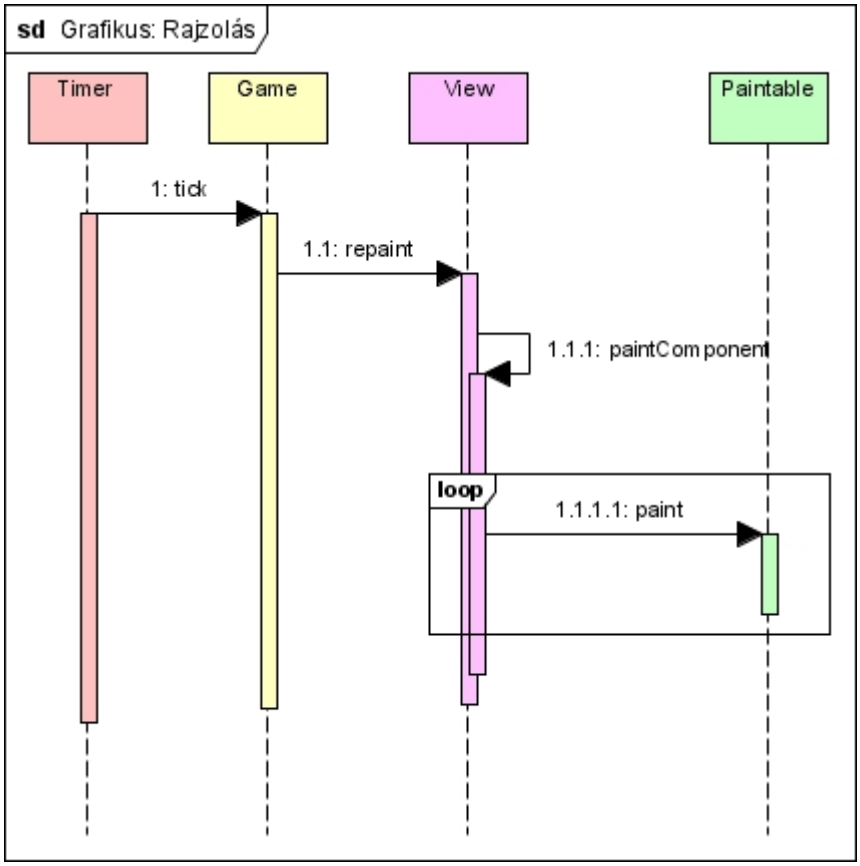
2.4 Kapcsolat az alkalmazói rendszerrel

Véleményünk szerint a kapcsolat az alkalmazói rendszerrel jól tükröződik az objektumok leírásából, ezért csak néhány esetben szeretnénk szekvencia diagrammal ábrázolni az eseményeket.

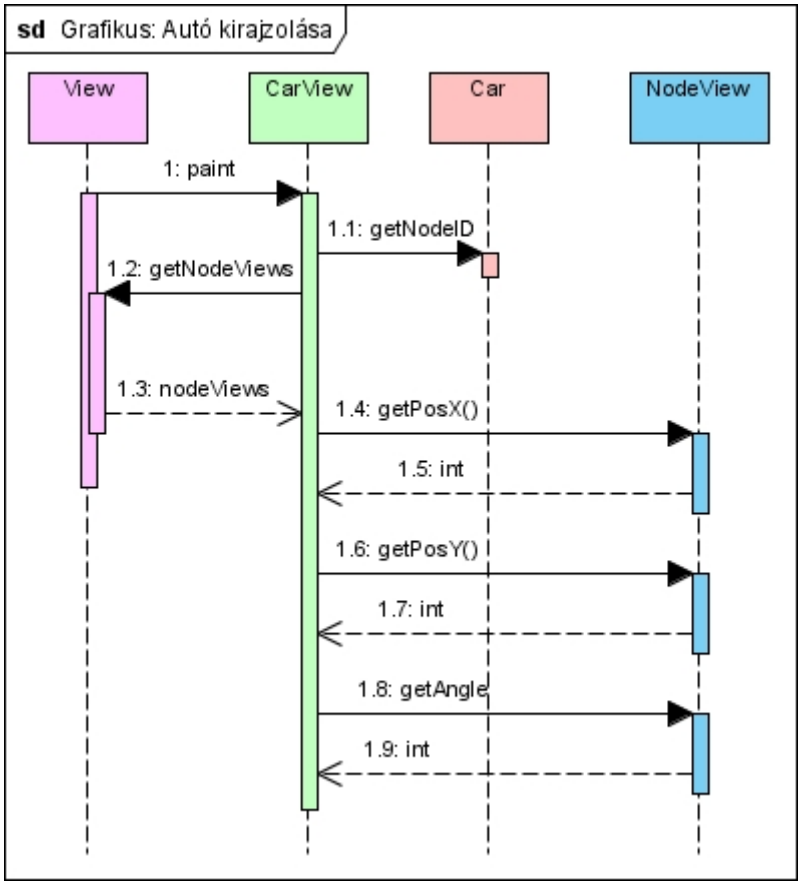
Új autó (a többi objektum kezelése is hasonló) keletkezik:



A kirajzolás folyamata:



A kirajzolások közül, egyedül a járművek kirajzolása érdekes:



2.5 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.04.19. 17:00	1,5 óra	Brucker Ferencz Orovecz Pocsai	Konzultáció
2010.04.19. 20:00	3 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet. Döntés: Ferencz megtervezi a grafikus rendszert, Pocsai megtervezi a kinézetét
2010.03.20. 17:00	5 óra	Ferencz	Tevékenység: Grafikus rendszer tervezése, statikus osztálydiagram, szekvencia diagramok, osztályok leírása
2010.03.20. 16:00	3 óra	Pocsai	Tevékenység: Grafikus felület kinézetének megtervezése.
2010.03.20. 22:00	1 óra	Brucker	Eddig elvégzett feladatok átnézése.
2010.03.21. 10:00	2 óra	Orovecz	Tevékenység: ellenőrzés, hibajavítás
2010.03.21. 16:00	2 óra	Brucker	Tevékenység: Orovecz által megfogalmazott észrevételek alkalmazása.
2010.03.21. 20:00	3 óra	Ferencz	Tevékenység: Beadandó véglegesítése, nyomtatás.

1. Grafikus változat beadása

1.1 Fordítási és futtatási útmutató

1.1.1 Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
src\controller\Controller.java	5117 byte	2010.05.05	Forráskód.
src\model\Bunny.java	515 byte	2010.05.05	Forráskód.
src\model\Car.java	4014 byte	2010.05.05	Forráskód.
src\model\CarFactory.java	1765 byte	2010.05.05	Forráskód.
src\model\City.java	17607 byte	2010.05.05	Forráskód.
src\model\Game.java	3626 byte	2010.05.05	Forráskód.
src\model\Node.java	4271 byte	2010.05.05	Forráskód.
src\model\PoliceCar.java	1898 byte	2010.05.05	Forráskód.
src\model\RunnerCar.java	2215 byte	2010.05.05	Forráskód.
src\model\Timer.java	835 byte	2010.05.05	Forráskód.
src\view\BunnyView.java	947 byte	2010.05.05	Forráskód.
src\view\CarView.java	1266 byte	2010.05.05	Forráskód.
src\view\CityView.java	981 byte	2010.05.05	Forráskód.
src\view\NodeView.java	1481 byte	2010.05.05	Forráskód.
src\view\Paintable.java	188 byte	2010.05.05	Forráskód.
src\view\PoliceCarView.java	1333 byte	2010.05.05	Forráskód.
src\view\RunnerCarView.java	1696 byte	2010.05.05	Forráskód.
src\view\View.java	3346 byte	2010.05.05	Forráskód.
img\bg0.jpg	45215 byte	2010.05.05	Kép.
img\bg1.jpg	64424 byte	2010.05.05	Kép.
img\bg2.jpg	57192 byte	2010.05.05	Kép.
img\bg3.jpg	60237 byte	2010.05.05	Kép.
img\bunny.png	979 byte	2010.05.05	Kép.
img\policecar.png	1190 byte	2010.05.03	Kép.
img\runner.png	1196 byte	2010.05.03	Kép.
img\car0.png	1157 byte	2010.05.03	Kép.
img\car1.png	1005 byte	2010.05.05	Kép.
img\car2.png	1060 byte	2010.05.05	Kép.
img\car3.png	940 byte	2010.05.05	Kép.
img\car4.png	945 byte	2010.05.05	Kép.
img\car5.png	1114 byte	2010.05.05	Kép.
img\car6.png	979 byte	2010.05.05	Kép.
build.bat	216 byte	2010.05.05	Lefordítja a prototípust.
grafikus.mf	58 byte	2010.05.05	Fordításhoz szükséges fájl.
grafikus.jar	30074 byte	2010.05.05	Általunk lefordított változat.

1.1.2 Fordítás és telepítés

Az egyszerűsítés, és a hibák elkerülésének érdekében a fordítást a *build.bat* nevű batch fájl végzi. A java útvonal beállítása után (pl. set PATH=„c:\Program

Files\Java\jdk1.6.0_18\bin”) célszerű végrehajtani. A fordítás eredménye a */bin* könyvtárban lesz látható: elkészülnek a *.class fájlok, valamint a futtatható *grafikus.jar* fájl.

Az előállításhoz szükséges **feltétel a legalább 1.6-os verziójú Java JDK kit megléte**.

1.1.3 Futtatás

A futtatás az elkészült jar fájllal lehetséges. Fontos, hogy az /img mappa a tartalmával együtt a könyvtárban legyen (ez a feltétel alapesetben teljesül).

1.1.4 Használat

A rablók autója piros színű. A játék menüből vagy billentyűzetről indítható (nyilakkal). A kék autók rendőrök, velük kerülni kell az ütközést, mivel ez minden olyan esetben végzetes, amikor a játékos nem rendelkezik a nyuszi erejével. Az átlagos autókkal való ütközés erőveszteséggel jár. Játék elkezdése előtt érdemes megfigyelni az autók mozgását (jelzőlámpák, STOP táblák).

A rablók autójának adatai a jobb alsó sarokban találhatók.

A játék célja a banktól eljutni a bázisig, mely elemek a pályán jól elkülönülnek.

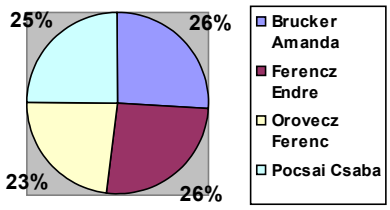
1.2 Értékelés

A végzett munkaórák százalékos eloszlását korrigáltuk az általunk megállapított szubjektív tényezőkkel (munka pontossága, nehézsége, határidő, stb.) és ebből adódott a végső százalék.

A csapatban végzett munkaórák eloszlása (a grafikus változatra):

Ferencz: 22,5 óra, 30,41%
Brucker: 18,5 óra, 25%
Orovecz: 15,5 óra, 20,94%
Pocsai: 17,50 óra, 23.64%
Összesen: 74 óra, 100%

Tag neve	Értékelés
Brucker Amanda Mária (UHZMDX)	26%
Ferencz Endre (PEBU1F)	26%
Orovecz Ferenc (B0HUPW)	23%
Pocsai Csaba (BOYKMJ)	25%



1.3 Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.04.25. 18:00	3 óra	Brucker Ferencz Orovecz Pocsai	Értekezlet a grafikus változatról. Eredménye: felosztottuk az implementálandó osztályokat.
2010.04.27. 13:00	6 óra	Orovecz	Tevékenység: Implementáció
2010.04.28. 16:00	4 óra	Pocsai	Tevékenység: Implementáció
2010.04.28. 17:00	5 óra	Brucker	Tevékenység: Implementáció
2010.05.01. 10:00	2,5 óra	Pocsai	Tevékenység: Az elkészített implementációk összefűzése, hibák javítása, hátralévő feladatok felosztása.
2010.05.01 13:00	3 óra	Brucker	Képek elkészítése, pályák szerkesztése.
2010.05.05 10:00	7 óra	Ferencz	Végső változat elkészítése, hibajavítás. Dokumentáció.
2010.05.06. 8:00	0,5 óra	Pocsai	Tevékenység: Néhány apróbb hiba javítása. Nyomtatás.

2. Összefoglalás

2.1 Projekt összegzés

Mi, a *Just Another Team* tagjai, szerettük a Szoftver laboratórium 4 tárgyat, tehát valószínűleg elfogultak lesznek a válaszaink.

A feladat hosszú volt és nehéz, pláne úgy, hogy azelőtt egyikünk sem dolgozott nagyobb feladaton. Mivel mindannyian kis otthoni „szobaprogramozók” voltunk, kihívás volt egy nagyobb projektet elkészíteni közösen, és elég sok új ismeretre sikerült szert tennünk eközben. A tárgy célja szerintünk pontosan az lehetett, hogy megtanítson minket a közös munkára és arra, hogyan kell egy nagy projektet az elejétől kezdve felépíteni, részfeladatokra tördelni és végül elkészíteni.

Mint már említettem, egyikünk sem dolgozott ezelőtt csapatban. Véleményem szerint ez volt az egyik legnehezebb feladat, hiszen különbözőek vagyunk, a gondolkozásunk is más, és kénytelen-kelletlen át kellett hidalni ezeket az akadályokat a közös munka érdekében. A munka legnehezebb részének a kezdeti modellalkotás bizonyult, hiszen egy jó, mondhatni hibátlan modellt kellett készíteni, amelyből a programot lekódolni már szinte „gyerekjáték” volt. Igyekeztünk a feladatot úgy megoldani, hogy mindenki ugyanolyan arányban vegyen részt benne és a hozzá legközelebb álló területen munkálkodhasson, továbbá, hogy az a csapattag kapjon egy adott feladatot, akinek voltak ötletei a megvalósításról, ezzel jelentősen megkönnyítettük egymás dolgát. Örömmel töltött el minket, hogy az eddig megszerezett ismereteinket egy nagyobb projekt keretében kamatoztatni tudtuk.

A pontozás és a feladatok nehézsége összhangban állt egymással. A nehezebb feladatokra több pontot lehetett kapni, esetleg lehetett pótolni azokat a részeket, amik nem bizonyultak elfogadhatóknak. Úgy gondoljuk, hogy ez a pontozási forma előnyös volt, hiszen nem minden feladat volt egyforma nehézségű. Többnyire elegendő volt az idő a feladatok megoldására. Természetesen ebben nagy szerepe volt annak, hogy megosztottuk a feladatokat egymás között.

Fentebb leírtam, hogy a tárgy célja szerintünk az lehetett, hogy megtanítsa a hallgatókat egy komoly projekten közösen dolgozni, ami igen előnyös, hiszen a való életre igyekszik felkészíteni minket. Úgy véljük ez nagy pozitívuma a tárgynak. Teljes mértékben elégettek voltunk mind a követelményekkel, mind az értékeléssel. Szívesen dolgoztunk a projekten.

Mindenképpen jó ötletnek tartottuk, hogy egy játékot kellett megírnunk, hiszen a bennünk szunnyadó gyermek felébredhetett egy kis időre a nehéz egyetemi napok idején. Bár konkrét ötlettel nem tudunk szolgálni jelenleg, játék program megírása szerintünk mindig is jó ötlet.

2.2 Statisztikák

2.2.1 A projektre fordított összes munkaidő, személyenként és összesen táblázatban

Tag neve	Szkeleton	Prototípus	Grafikus	Összesen
Brucker Amanda Mária (UHZMDX)	30.50 óra	24.75 óra	18.50 óra	73.75 óra
Ferencz Endre (PEBU1F)	38.75 óra	27.50 óra	22.50 óra	88.75 óra
Orovecz Ferenc (B0HUPW)	23.50 óra	25.50 óra	15.50 óra	64.50 óra
Pocsai Csaba (BOYKMJ)	27.50 óra	25.50 óra	17.50 óra	70.50 óra
Összesen:	120.25 óra	103.25 óra	74 óra	297.5 óra

2.2.2 Szkeleton forrása

Fájl neve	Forrássorok
src/Car.java	246
src/CarFactory.java	103
src/City.java	594
src/Controller.java	151
src/Game.java	186
src/Node.java	224
src/PoliceCar.java	25
src/RunnerCar.java	209
src/Timer.java	85
Összesen:	1823

2.2.3 Prototípus forrása

Fájl neve	Forrássorok
src\controller\Controller.java	446
src\model\Bunny.java	38
src\model\Car.java	170
src\model\CarFactory.java	63
src\model\City.java	285
src\model\Game.java	109
src\model\Node.java	178
src\model\PoliceCar.java	85
src\model\RunnerCar.java	112
src\model\Timer.java	40
Összesen:	1526

2.2.4 Grafikus változat forrása

Fájl neve	Forrássorok
src\controller\Controller.java	233
src\model\Bunny.java	31
src\model\Car.java	192
src\model\CarFactory.java	69
src\model\City.java	733
src\model\Game.java	172
src\model\Node.java	216
src\model\PoliceCar.java	83
src\model\RunnerCar.java	106
src\model\Timer.java	49
src\view\BunnyView.java	44
src\view\CarView.java	57
src\view\CityView.java	49
src\view\NodeView.java	74
src\view\Paintable.java	12
src\view\PoliceCarView.java	58
src\view\RunnerCarView.java	67
src\view\View.java	132
Összesen:	2377