**Debreceni Egyetem**
**Informatikai Kar**
Információ Technológia Tanszék

# Valós idejű rendőr-ágens irányító a Robocar World Championshiphez

*Témavezető:*
**dr. Bátfai Norbert**
egyetemi adjunktus

*Készítette:*
**Balkus Gergő Máté**
programtervező informatikus hallgató

A dolgozat benyújtásához hozzájárulok.

dr. Bátfai Norbert

Debrecen
2016

# Tartalomjegyzék

# 1. fejezet

# Bevezetés

1.1. ábra. A Google által fejlesztett robotautó. Forrás: [16].

A közelmúltban nagy mértékű fejlődést vehettünk észre az autógyártásban, különösképp a magukat irányítani tudó autók kapcsán. Több és több autók gyártásával foglalkozó cég fejleszt robotautót. Ilyen például ezen a területen az elsőként megjelenő Google, melynek a robotautóját a 1.1. képen láthatjuk. Ezek a szokások dinamikus fejlődést mutatnak. A 2000-es évektől kezdve az autók elektronikus eszközei jelentős mértékben arról szóltak, hogy azok segítették a vezetőt különböző módokon. Ilyen eszközök például a tolató radarok, illetve a gyalogosfigyelő rendszerek is.

Az elkövetkező éveket azonban várhatóan az autonóm autók nagy mértékű megjelenése és elterjedése váltja fel. Ezek az autók mellett nemrég láthattuk a Tesla új modelljének a bemutatóját is, mely ugyancsak erősíti a feltevést, miszerint az elektromos hajtású, illetve az úgynevezett „hibrid” hajtású kocsik előtt áll a jövő. Elektromos hajtású kocsikat manapság is láthatunk már az utcákon. Ezeket könnyen felismerhetjük a zöld hátterű rendszámtáblájukól.

A fent említettek alapján tisztán látszik, hogy az autóipar szemléletváltás előtt áll. A nem újrahasznosítható energiaforrásokon alapulú motorokat és a vezetői élményt kezdi lassan felváltani a hosszabb távon fenntartható, újrahasznosítható energiaforrást használó motorokkal hajtott autonóm gépkocsik, mellyel az utazáshoz a pihentető időtöltés vízióját próbálják hozzárendelni.

1.2. ábra. Egy elképzelt okos város vektorgrafikus ábrázolása. Forrás: [29].

2050-re a világ lakosságának 70%-a valószínűleg városokban fog élni [32]. Ez a gyors városiasodás új kihívásokat állít fel a város infrastruktúrája felé. Több kérdés is felmerül ilyenkor. Hogyan tudunk biztonságosan „irányítani” egy ilyen nagy méretű népességet? Mely szolgáltatásokkal lássa el a városi ügyintézés azért, hogy fenntartható legyen a város?

Ezeket a kérdéseket is megválaszolja az Okos városok kutatási területe, mely eredményei egyre inkább láthatóak a mindennapi életben is. Talán az egyik legfontosabb kérdés ez ügyben a városi közlekedés. Ahogy nő a népesség, úgy nő azok az emberek (és kocsik)

száma akik a város infrastruktúráját, illetve úthálózatát használják. Hogyan tud segíteni a városi management, hogy az ott élők a lehető leghatékonyabban tudjanak közlekedni? Ezt a kérdést válaszolja meg a smart traffic management.

Az Smart City Research and Development terület láthatóan fontos szerepet fog kapni az elkövetkezendő 10-20 évben. Előrejelzések azt mutatják, hogy 2023-ra több mint 170 milliárd USD beruházás valósul meg a világon [24]. Az informatikai fejlesztések ezen a téren már egyre jobban beférkőztek a hétköznapokba is, mint például az iCity [17], a FIWare [13], vagy a [33]. Ezeken túl egyre többen kísérleteznek azzal, hogy a város embereinek a mindennapjait hatékonyabbá és biztonságosabbá varázsolja ([23], [14], [30]).

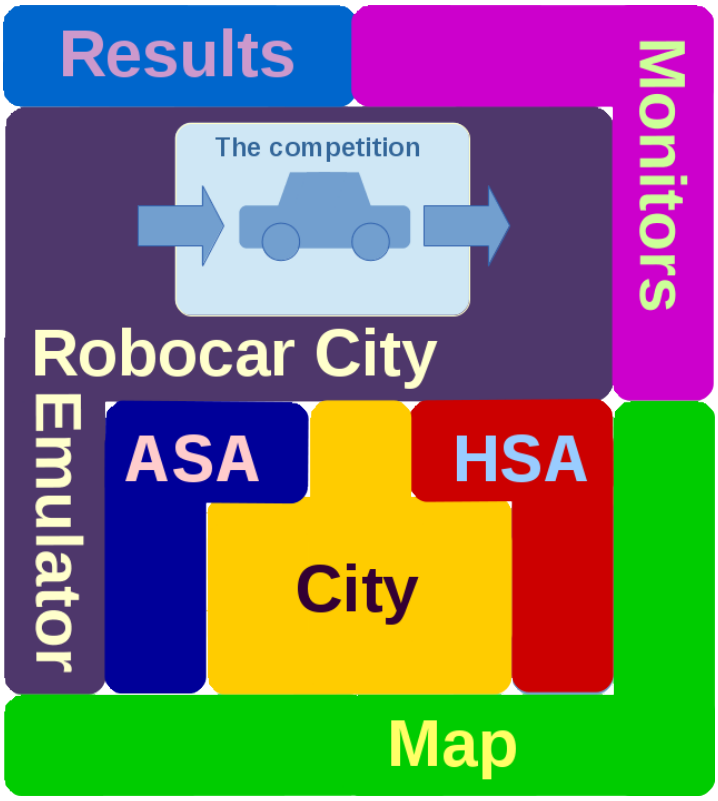
Sok tanulmány foglalkozik még azzal, hogy hogyan lehetne egy városnak az „okosságát" megmérni ([11], [10], [34], [25]). A városok „okosságáról" még egy sorrendet is készítettek [15]. Az okos városokban a vészhelyzetek megoldása is fontos szerepet kap [12].

# 2. fejezet

# A Robocar World Championship (OOCWC)

## 2.1. Ismerkedés az OOCWC rendszerrel

Az OOCWC rendszer célja, hogy az autonóm autók és az okos városok közötti összefüggéseket vizsgálja, kutatási valamint oktatási célokat is szolgál. A rendszer felépítését a 2.1. ábra szemlélteti.




2.1. ábra. Az OOCWC rendszer tetris terve. Forrás: [4].

Ez alapján a rendszer egyes elemei:

- Map – a szimuláció egy adott térképen értelmezett,
- City – a szimuláció működési egysége,
- The competition – a verseny célja / maga a verseny,
- ASA – automatikus adatgyűjtő rendszer,
- HSA – kézi adatgyűjtő rendszer,
- Robocar City Emulator – forgalom emuláció,
- Results – a verseny eredményei, illetve kísérletek eredményei,
- Monitors – megjelenítők, vizualizáció.

A rendszer többcélú. Egyrészt egy kutatási platformot kínál forgalomelemzésre, szimulációkra. A szimuláció az OpenStreetMap [27] térképein fut.

A rendszer másik célja, hogy egy verseny segítségével találjuk meg a lehető legjobb forgalomirányító algoritmust. Az OOCWC már több sikeres versenyen is túl van a Debreceni Egyetem Informatikai Karán [26]. Gyakori, hogy egy kutatás által létrejött rendszerre versenyt szerveznek a felhasználók körében. Erre egy jó példa az mesterséges intelligencia (AI - Artificial Intelligence) területén a RoboCup [21].

Fontos megemlíteni, hogy a Robocar World Cup kiválóan alkalmazkodik a különböző esetekre. Az oktatás és kutatások támogatására került kifejlesztésre a „Police Edition” (egy pillanatkép ebből a változatból a 4.1. ábrán látható). A cél, hogy rendőr-ágensekkel, melyeket a felhasználó által megírt irányító algoritmus vezérel, minél több gengszter-ágenst kapjunk el.

A rendszer jelenleg háromféle forgalmi egységet különböztet meg, routine cars, smart cars és guided cars. A szimuláció kezdőállapota a routine cars és smart cars elhelyezése a térképen a gyűjtött adatok alapján.

A rendszer „Police Edition” változatából készíthetnek egy saját fork-ot a hallgatók és az érdeklődő kutatók. A játék célja, hogy a rendőr ágensekkel minél több gengszter ágenst kapjunk el. A bemutatott projekt (4. fejezet) is egy ilyen forknak tekinthető.

A rendszerről további információk olvashatóak a [5] közleményben.

## 2.2. A rendszer elindítása

A továbbiakban tételezzük fel hogy a projektben a szükséges forráskódokat már lefordítottuk, így binárisan elérhetőek, továbbá jelenleg az „rcemu” mappában vagyunk, és a szülőmappában elérhető . Ehhez segítséget az OOCWC eredeti tárolóján [4] találhatunk.

Elsőnek az okos városunkat kell elindítani. Ezt a 2.1. forráskódban látható Bash parancs futtatásával tehetjük meg.

```
src/smartcity --osm=../debrecen.osm --city=Debrecen --shm=
    DebrecenSharedMemory
```

Forráskód 2.1. A SmartCity elindítása

Itt a kapcsolók a következő beállításokat jelentik:

- osm – Az OpenStreetMap térkép elérési útvonala.
- city – A szimulált város neve.
- shm – Az osztott memória elérési neve.

Amennyiben a SmartCity kimenetén megjelenik hogy „Ready”, úgy folytathatjuk a következő lépéssel, a szimulációs szerver elindításával, melynek a parancsát a 2.2. forráskódban láthatjuk.

```
src/traffic --port=10007 --shm=DebrecenSharedMemory
```

Forráskód 2.2. A szimulációs szerver elindítása

A beállítási kapcsolók jelentése a következő:

- port – A szervernek a portja, amelyen a szerver üzemel.
- shm – Az osztott memória neve.

## 2.3. Szükséges módosítások a projekt számára

### 2.3.1. Az eredeti irányítás

Eredetileg a rendőr-ágensek is C++ nyelven (3.2. fejezet) lettek megvalósítva, ezáltal hozzáférnek az OOCWC osztott memóriájához (shared memory), amiben több adat közt megtalálható az OSM [27] (3.5. fejezet) térképből felépített irányított gráf BGL (Boost Graph Library) (3.2.2. fejezet). Ezt a gráfot és a Boost, gráfokra implementált, kereső algoritmusait felhasználva a rendőr-ágens képes létrehozni a szimulációs szerver által értelmezhető útkereső (routing) parancsot, mellyel a szerver a kapott parancs alapján mozgatja a megfelelő rendőr-ágenst.

Az eredeti parancs a következőképp néz ki:

```
<route [a csomopontok szama: N] [rendor-agens ID] [N darab csomopont]>
```

Ez a parancs a 2.3. forráskódrészleten látható módon lesz feldolgozva.

```
try {
  /* ........ */

  if ( cop->set_route ( carLexer.get_route() ) )
    length += std::sprintf ( data+length, "<OK %d>", carLexer.get_id()
    );
  else
    length += std::sprintf ( data+length, "<ERR bad routing vector>" );

  /* ........ */
  boost::asio::write ( client_socket, boost::asio::buffer ( data,
    length ) );

}
catch ( std::exception& e ) {
  std::cerr << "Error: " << e.what() << std::endl;
}
```

Forráskód 2.3. Az eredeti routing forráskód részlete

### 2.3.2. Az új irányítás

Ahhoz hogy az eredeti irányításhoz hasonló mozgatást érjünk el a kívánt rendőr-ágensen, egy olyan programozási nyelvvel amelynek nincs hozzáférése az OOCWC által felépített osztott memóriához, egy új parancs bevezetésére volt szükség, amely helyettesíti az eredeti útkereső parancsot, a valódi routingot (az út megtalálását kettő csomópont között) pedig a szerverre kellett bízni.

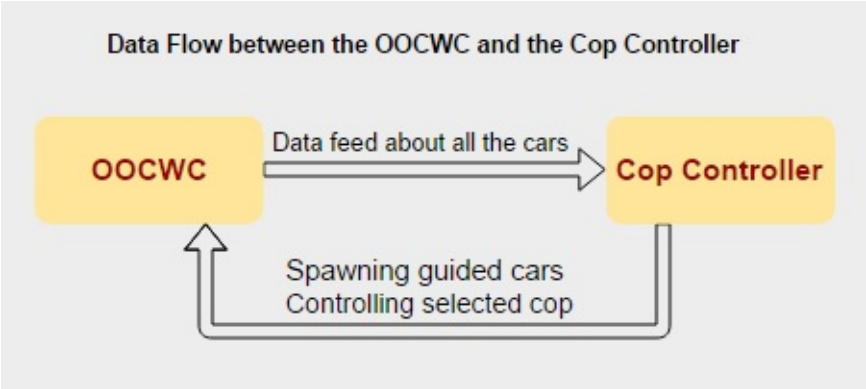
Ennek a változtatásnak a bevezetésével úgy vélem hogy egy új lehetőség nyílik meg az OOCWC rendszer számára, mégpedig hogy több programozási nyelv segítségével is meg lehessen jeleníteni az aktuális állapotot, illetve lehessen irányítani a rendőr-ágenseket. (I'm looking at you, Python)

TODO az új parancs szintaxisának bemutatása

TODO forráskód az új routingról

## 2.4. Az adat áramlása

Ahogy a 2.2. ábrán látható, ahogy folyik a szimuláció a szerveren, úgy minden egyes lépésenként a szerver elküldi az aktuális állást az alkalmazásnak. Ezt a Cop Controller feldolgozza, ezáltal valós idejű képet adva a jelenleg állásról.




2.2. ábra. Az adat áramlása az OOCWC rendszer (2. fejezet) és az alkalmazás (4. fejezet) között.

Az alkalmazás oldaláról több funkciót is lehet lehet használni. Ezekeről többet a 4.2. fejezetben olvashatsz. Ezek a funkciók azonnal feldolgozásra kerülnek a szerveren, így a változás azonnal látható lesz.

# 3. fejezet

# Felhasznált technológiák

## 3.1. Java

A Java egy általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett az 1990-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle. 2011-ben a Java 1.7-es verzióját, 2014-ben pedig az 1.8-as verzióját már az új tulajdonos gondozásában adták ki.

```
1 public class HelloVilag {
2     public static void main(String[] args) {
3         System.out.println("Hello Vilag!");
4     }
5 }
```

Forráskód 3.1. A klaszzikus „Hello World!” Javában

A nyelv 2016-ban is sikeres a szerver oldalon a servlet, a JSP és Enterprise JavaBeans, JDBC technológiákkal, integrációs lehetőségeivel, nyelvi eszközeivel, jvm nyelveivel és a nyílt forráskódú közösség tudására is építve.

A Java alkalmazásokat jellemzően bájtkód formátumra alakítják, de közvetlenül natív (gépi) kód is készíthető Java forráskódból. A bájtkód futtatása a Java virtuális géppel történik, ami vagy interpretálja a bájtkódot, vagy natív gépi kódot készít belőle, és azt futtatja az adott operációs rendszeren. Létezik közvetlenül Java bájtkódot futtató hardver is, az úgynevezett Java processzor.

A Java nyelv a szintaxisát főleg a C és a C++ (3.2. fejezet) nyelvektől örökölte, viszont sokkal egyszerűbb objektummodellel rendelkezik, mint a C++. Példaként szolgál az egyik fő különbsége a C++ nyelvtől: nincs az osztályok között többszörös öröklődés (ezt a célt az interfészek valósítják meg).

A JavaScript szintaxisa és neve hasonló ugyan a Java-éhoz, de a két nyelv nem áll olyan szoros rokonságban, mint azt ezekből a hasonlóságokból gondolhatnánk.

Négy fontos szempontot tartottak szem előtt, amikor a Javát kifejlesztették:

- objektumorientáltság;
- függetlenség az operációs rendszertől amelyen fut (platformfüggetlenség);
- olyan kódokat és könyvtárakat tartalmazzon, amelyek elősegítik a hálózati programozást;
- távoli gépeken is képes legyen biztonságosan futni.

### 3.1.1. Története

Bár a nyelv neve kezdetben Oak (tölgyfa) volt, (James Gosling, a nyelv atyja nevezte így az irodája előtt növő tölgyfáról), később kiderült, hogy ilyen elnevezésű nyelv már létezik, ezért végül Java néven vált ismertté. A Java nyelvet kávézás közben találták ki, innen ered a kávéscsésze ikon [20].

### 3.1.2. Objektumorientáltság

A nyelv első tulajdonsága, az objektumorientáltság („OO”), a programozási stílusra és a nyelv struktúrájára utal. Az OO fontos szempontja, hogy a szoftvert „dolgok” (objektumok) alapján csoportosítja, nem az elvégzett feladatok a fő szempont. Ennek alapja, hogy az előbbi sokkal kevesebbet változik, mint az utóbbi, így az objektumok (az adatokat tartalmazó entitások) jobb alapot biztosítanak egy szoftverrendszer megtervezéséhez. A cél az volt, hogy nagy fejlesztési projekteket könnyebben lehessen kezelni, így csökken az elhibázott projektek száma.

### 3.1.3. Platformfüggetlenség

Ez a tulajdonság azt jelenti, hogy a Java-ban írt programok a legtöbb hardveren ugyanúgy futnak. Ezt úgy érik el, hogy a Java fordítóprogram a forráskódot csak egy úgynevezett Java bájtkódra fordítja le. Léteznek továbbá szabványos könyvtárcsomagok, amelyek, - közvetítve a kód és a gép között, - egységes funkcionalitásként teszik elérhetővé az illető hardver sajátságosságait (grafika, szálak és hálózat).

Vannak olyan Java fordítóprogramok, amelyek a forráskódot natív gépi kódra fordítják le, - ilyen például a GCJ, - ezzel valamelyest felgyorsítva annak futtatását. Cserébe a lefordított program elveszíti hordozhatóságát.

Egyes cégek, mint például a Microsoft, mégis platformfüggő sajátságokat adtak a nyelvhez, amire a Sun keményen reagált: beperelte a Microsoftot (az amerikai bíróság 20 millió dollár kártérítésre és a sajátos tulajdonságok visszavonására kötelezte a céget). Válaszként a Microsoft kihagyta a Java rendszert a jövőbeli termékekből és Windows-változatokból. Ez azt jelenti, hogy az Internet Explorer webböngésző alapváltozataiból hiányzik a Java. Így abban az olyan weboldalak, amelyek Java-t használnak, nem fognak helyesen megjelenni. A Windows-felhasználók e problémáját megoldva a Sun és más cégek ingyenesen letölthetővé tették a JVM rendszert azon Windows-változatok számára, amelyekből a virtuális gép hiányzik.

A hordozhatóság megvalósítása technikailag nagyon bonyolult. E közben a Java esetében is sok vita volt. Az „írd meg egyszer, futtasd bárhol" szlogenből „írd meg egyszer, keress hibát mindenhol" lett. 2016-ra a hordozhatóság nem okoz tovább problémát, mivel maga a Java is nyílt szabványokra épül, például: openGL, vagy Open POSIX.

A Java minden jelentősebb platformon elérhető (Linux, Unix, Windows, iOS).

### 3.1.4. Swing

A Swing egy eszközkészlet a grafikus felhasználói felületek létrehozására.

A Swinget azzal a céllal fejlesztették, hogy egy szofisztikáltabb GUI komponenshalmazt szolgáltasson mint a korábbi Abstract Window Toolkit (AWT) [1]. A Swingnek a komponensei erőteljesebb és flexibilisebb az AWT-hez hasonlítva. A már ismert komponensekhez, mint például a Button-ök, check boxok és labelek, a Swing olyan további fejlett eszközökkel szolgál, mint a tabbed panel, scroll pane, fák, táblázatok és listák.

Ellentétben az Abstract Window Toolkit komponenseivel, a Swing komponensei nem platform-specifikus kódként lettek implementálva. Helyette teljesen Java nyelven írták meg, ezáltal a komponensek platform-függetlenek lettek. A „lightweight" kifejezéssel írjuk le az ilyen elemeket [22].

A Swinget a közeljövőben a JavaFX [19] fogja felváltani.

### 3.1.5. JXMapViewer

A JXMapViewer [31] egy nyílt forráskódú Java könyvtár, ami egy Swing (3.1.4. fejezet) JPanelt [18] szolgáltat, melynek feladata a térkép betöltése, és mutatása.

TODO

## 3.2. C++

A C++ egy általános célú, magas szintű programozási nyelv. Támogatja a procedurális, az objektumorientált és a generikus programozást is. Napjainkban szinte minden operációs rendszer alá létezik C++ fordító. A nyelv a C hatékonyságának megőrzése mellett törekszik a könnyebben megírható, karbantartható és újrahasznosítható kód írására, ez azonban sok kompromisszummal jár. Erre utal, hogy általánosan elterjedt a mid-level minősítése is, bár szigorú értelemben véve egyértelműen magas szintű.

A nyelv tervezésénél fontos szempont volt a C-vel való kompatibilitás, ezt oly mértékben sikerült megvalósítani, hogy minden szintaktikailag helyes C program egyben egy szintaktikailag helyes C++ program is.

```
1 #include <iostream>
2
3 int main()
4 {
5     std::cout << "Hello, vilag!" << std::endl;
6     return 0;
7 }
```

Forráskód 3.2. A klaszzikus „Hello World!” C++-ban

### 3.2.1. Története

Bjarne Stroustrup kezdte el a C++ programozási nyelv fejlesztését a C programozási nyelv kiterjesztéseként, más nyelvekből véve át megoldásokat, ötleteket. A nyelv első, nem kísérleti körülmények közt való használatára 1983-ban került sor, 1987-ben pedig nyilvánvalóvá vált, hogy a C++ szabványosítása elkerülhetetlen. Ez a folyamat 1991 júniusában kezdődött el, amikor az ISO szabványosítási kezdeményezés részévé vált. A C++ programozási nyelv szabványát 1998-ban hagyták jóvá ISO/IEC 14882:1998 [6] (C++98) néven, a 2003-as változat kódjelzése pedig a ISO/IEC 14882:2003 [7] (C++03) lett. Az ezt követő 2011-es változatot ISO/IEC 14882:2011 [8] (C++11) kódjelzéssel hagyták jóvá. A jelenlegi legfrissebb változata a 2014-es ISO/IEC 14882:2014 [9] (C++14) kódjelzésű verzió.

### 3.2.2. Boost library

A Boost a C++ programozási nyelvhez ad támogatást olyan könyvtárakkal, melyek különböző területek különböző problémáit segítenek megoldani. Többek között ilyen te-


3.1. ábra. A Boost logója. Forrás: [3].

rületek a lineáris algebra, a pseudorandom számok generálása, a többszálasítás, a képfeldolgozás, a reguláris kiejezések, illetve a unittesztelés.

Az OOCWC (2. fejezet) számára az egyik legfontosabb ilyen könyvtár a BGL (Boost Graph Library) [2], melynek segítségével épül fel az OpenStreetMap [27] (3.5. fejezet) térkép alapján az OOCWC irányított gráfja.

## 3.3. Maven

Az Apache Maven (röviden Maven) egy szoftver, amelyet szoftverprojektek menedzselésére és a build folyamat automatizálására lehet használni. Jason van Zyl készítette 2002-ben. Funkcionalitásában hasonlít az Apache Ant eszközhöz. A projektet az Apache Software Foundation hosztolja, ahol korábban a Jakarta Projekt részeként működött.

A Maven bevezeti a POM, azaz a Projekt Objektummodell (angolul: Project Object Model) fogalmát. Egy POM egy buildelendő projektet ír le és annak függőségeit. Az egyes lépéseket céloknak, angolul goal-oknak nevezik. Vannak előre definiált célok a tipikus feladatokra, mint például a kód fordítása és csomagolása, de a felhasználónak lehetősége van saját célokat is definiálni a projektspecifikus lépések végrehajtására.

A Maven hálózatképes, tehát szükség esetén dinamikusan is le tud tölteni komponenseket. Repository névvel illetik a különböző hosztok fájlrendszereinek azon mappáit, ahol a letölthető komponensek találhatók. A Maven nem csak a repository-kból való letöltést támogatja, hanem a készült szoftvercsomag feltöltését is. Ezzel az automatizálható le- és feltöltési mechanizmussal a Maven de facto szabványt próbál teremteni, de elég lassan fogadja el a Java közösség.

A Maven-nek léteznek beépülő moduljai számos népszerű IDE-höz, melyek integrációt nyújtanak a Mavennal való az IDE build mechanizmusához, kódszerkesztő eszközeihez. Ezekkel lehetőség van az IDE-ből lefordítani a projekteket, és beállítani a classpath-t a kód kiegészítéshez ill. fordító hibák színezéséhez stb. Ilyen IDE például az Eclipse, illetve a NetBeans is.

## 3.4. Git

A Git egy nyílt forráskódú, elosztott verziókezelő szoftver, vagy másképpen egy szoftverforráskód-kezelő rendszer, amely a sebességre helyezi a hangsúlyt. A Gitet eredetileg Linus Torvalds fejlesztette ki a Linux kernel fejlesztéséhez. Minden Git munkamásolat egy teljes értékű repository teljes verziótörténettel és teljes revízió követési lehetőséggel, amely nem függ a hálózat elérésétől vagy központi szervertől.

Számos nagy volumenű projekt használja jelenleg a Gitet verziókezelő rendszerként; a legfontosabbak ezek közül: Linux-rendszermag, GNOME, Samba, X.org, Qt. A Git karbantartásának felügyeletét jelenleg Junio Hamano látja el.

A Git rendszerét a BitKeeper és a Monotone ihlette. Eredetileg alacsony szintű verziókövető rendszernek tervezték, azonban teljes értékű revíziókövető rendszerré szélesedett, amelyet már közvetlenül is lehet használni. Torvalds, aki jártas a nagy, szétosztott fejlesztési projektekben és a fájlrendszerekben, és gyorsan akart működő rendszert fejleszteni, ezekre az elvekre alapozta a Gitet:

- A nemlineáris fejlesztés erős támogatása. A Git támogatja az ágak (branchek) készítését, összefésülésüket, és tartalmaz eszközöket a nem-lineáris fejlesztési történet ábrázolására.

- Elosztott fejlesztés. A Git minden fejlesztő helyi munkaváltozatában rendelkezésre bocsátja a teljes addigi fejlesztési történetet, és a változtatások másolása mindig két repository között történik. Ezeket a változtatásokat, mint külön ágakat importálják és összefésülhetőek, hasonlóan a helyben létrehozott fejlesztési branchokhoz.

- Nagy objektumok hatékony használata.

- Kriptográfiailag hitelesített történet. A Git-történet olyan módon tárolódik, hogy a szóban forgó revízió neve függ a hozzá vezető fejlesztési történettől. Ha egyszer publikálták, akkor már nem lehetséges észrevétlenül megváltoztatni egy régi revíziót. A struktúra hasonlatos a hash-fához, de hozzáadott adatokkal az egyes csomópontokon és leveleken.

- Eszközkészlet-szerű megoldás. A Git maga C-ben írt programok halmaza, számos shell-szkripttel megtűzdelve, amelyek összefűzik ezeket a C-ben írt programokat. Annak ellenére, hogy a szkriptek többségét újraírták C-ben a Microsoft Windows-ra való portolás eredményeként, a struktúra megmaradt, és továbbra is könnyű az egyes komponenseket láncba rendezni az igényeknek megfelelően.

## 3.5. OpentStreetMap (OSM)

Az OpenStreetMap (OSM) csoportmunkán alapuló térképfejlesztés, melynek célja egy szabadon szerkeszthető és felhasználható térkép készítése az egész világról.

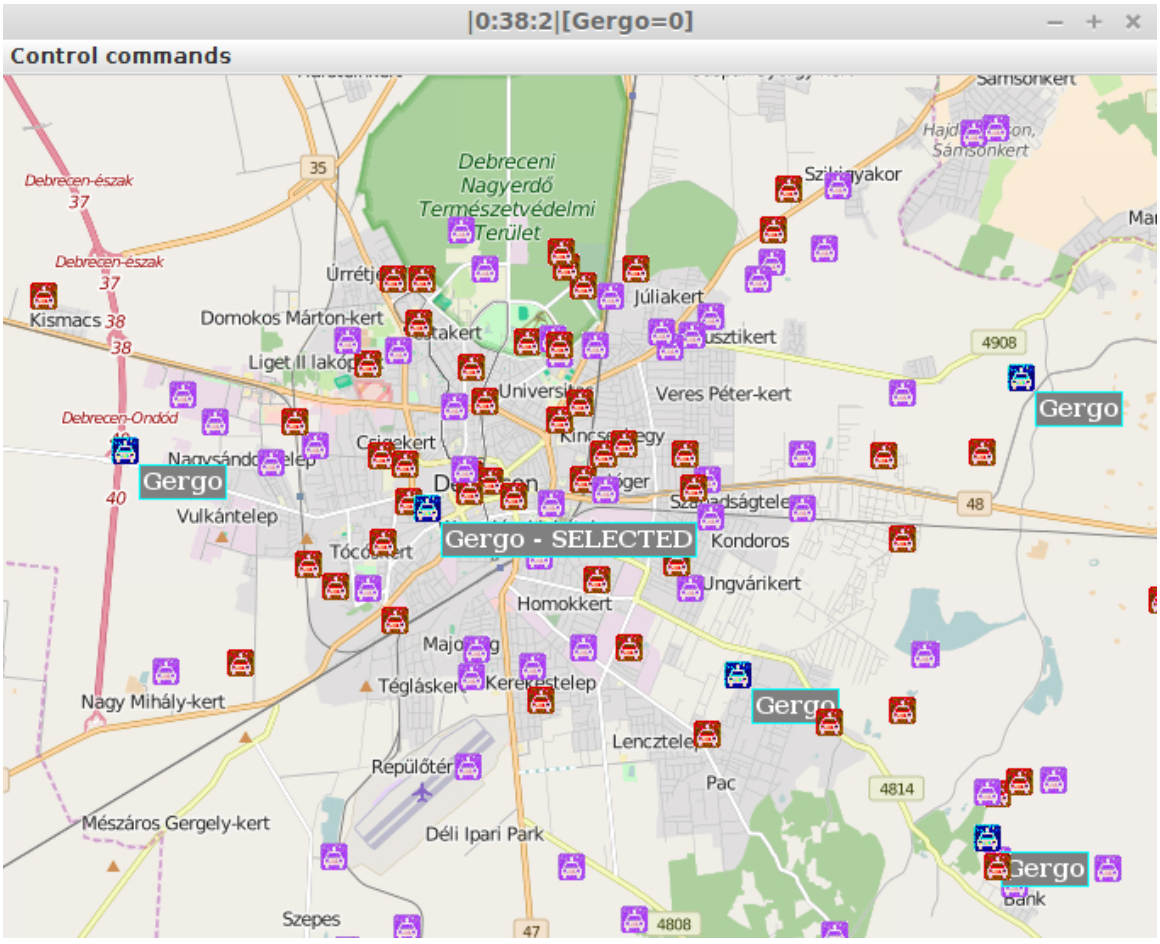
A térképek hordozható GPS eszközökből, légifotókból, szabadon használható, nemzeti kormányzati nyilvántartásokból és egyéb szabad forrásokból származó adatok, vagy egyszerű helyismeret alapján készültek. A térinformatikai adatbázis teljes tartalma Open Database License licenc alatt érhető el letöltésre [28] (a konkrétan megrajzolt térképrészletek általában változatos szabad licencek alatt attól függően, hogy azokat ki rajzolta az adatbázisból).

Az OpenStreetMapot olyan oldalak ihlették, mint a Wikipédia – a térkép-megjelenítésnél elérhető egy jellegzetes „Szerkesztés” fül, és a változtatások teljes története is rendelkezésre áll. A regisztrált felhasználók feltölthetnek GPS nyomvonalakat, és szerkeszthetik a vektoros adatokat adott szerkesztőeszközök használatával.

A `www.openstreetmap.org` [27] címen a regisztrációt követően bárki elkezdheti a szerkesztést. A GPS készülékkel vagy PDA-val személyesen gyűjtött adatokat GPX formátumban lehet a megadott oldalra feltölteni, majd a rögzített pontok alapján a terepttárgyakat megrajzolni és beazonosítani. A projektben résztvevők térképező hétvégéket szoktak szervezni egy-egy kiemelt terület felmérésére. A projektnek már 2006-ban több magyar tagja volt, pedig akkor a szerkesztők száma még csak kb. 1000 volt.

# 4. fejezet

# Az alkalmazás (Cop Controller)




4.1. ábra. Pillanatkép a rendszer „Police Edition” változatának a projekt szerinti módosításával (lásd: 4. fejezet). A térkép az OSM egy részlete, Debrecen, Hajdú-Bihar Megye, Magyarország. A megjelenítést a JXMapViewer2 [31] biztosítja. A térképen a routine car rózsaszínnel, a gengszter ágensek pirossal (smart car), a rendőrágensek (guided car) kékkel jelölve. Az éppen kiválasztott rendőr-ágens melyet éppen irányítunk pedig el van látva a „SELECTED” felirattal. Forrás: [5]

Ennek a projektnek a célja az, hogy az OOCWC (2. fejezet) kvalifikációs (illetve a bátrabbaknak a verseny) részét (is) valósidejű irányítás válthassa ki, ezáltal érdekesebbé, és kézügyességtől (is) függővé válik a gengszterek elkapása.

A projekt egy Maven (3.3. fejezet) által menedzselt Java (3.1. fejezet) alkalmazás, amihez egy Swing (3.1.4. fejezet) grafikus felhasználói felület (Graphic User Interface) tartozik. Továbbá a Maven felelős a projekt dependenciáinak (dependency) a megfelelő verziószámú letöltéséért is.

Ilyen függőség például JXMapViewer2 [31] (3.1.5. fejezet)).

## 4.1. Az alkalmazás elindítása

TODO képek importálásról + parancsok

## 4.2. Az alkalmazás funkciói

### 4.2.1. Aktuális állás megjelenítése

Az aktuális állást egyrészt a főképernyőn lehet látni, másrészt az alkalmazás fejlécében összegezve van hogy a szimuláció mennyi időnél tart, illetve hogy melyik rendőrcsapat összesen hány gengsztert kapott el eddig.

TODO

### 4.2.2. Rendőrök és gengszterek hozzáadása

Az alkalmazásban lehetőség van:

- 1 rendőr hozzáadására (4.1. forráskód);
- 10 rendőr hozzáadására (4.2. forráskód);
- 100 gengszter hozzáadására (4.3. forráskód).

```
1 echo "<init guided Gergo 1 c>" | telnet localhost 10007
```

Forráskód 4.1. 1 rendőr hozzáadása; ahol a „Gergo” a rendőr csapatnevét jelenti.

```
1 echo "<init guided Gergo 10 c>" | telnet localhost 10007
```

Forráskód 4.2. 10 rendőr hozzáadása; ahol a „Gergo” a rendőrök csapatnevét jelenti.

```
echo "<init Gergo 100 g>" | telnet localhost 10007
```

Forráskód 4.3. 100 gengszter hozzáadása; ahol a „Gergo” azt a csapatnak a nevét jelenti amely hozzáadta a gengsztereket.

Mindhárom parancs működésének az alapelve megegyezik. A telnet protokollt felhasználva elküldjük a kívánt parancsnak megfelelő üzenetet, amit a szimulációs szerver feldolgoz. Ezeknek a parancsoknak az eredményeit azonnal láthatjuk (Erről többet a 2.4. fejezetben lehet olvasni).

Ezeknek a megoldásoknak az előnye hogy a parancsok egy teljesen különböző processzként futnak mint az alkalmazás, ezáltal nem zavarják a folyamatos adatáramlást, ami a megjelenítéshez szükséges. Példa az alkalmazásból 100 gengszter inicializálására a 4.4. forráskódrészletben látható.

```
/* ..... */
try {
  String cmdPath = StaticUtils.getResourcePath() + "init_100_gangsters.
    sh";
  LOGGER.debug("Relative path: {}", cmdPath);
  Process p = Runtime.getRuntime().exec(new String[] { "/bin/sh",
    cmdPath });
  p.waitFor();

  LOGGER.debug("Added 100 gangsters.");
  } catch (IOException | InterruptedException e) {
    LOGGER.error(e.getMessage());
  }
}
/* ..... */
```

Forráskód 4.4. 100 gengszter hozzáadása a 4.3. forráskódot felhasználva Java nyelven.

### 4.2.3. Rendőrök irányítása

Miután már van saját rendőrünk a szimulációban (lásd: 4.2.2. fejezet), Tudjuk őket irányítani is.

Alaphelyzetben nincs egy rendőr se kiválasztva. Ahhoz hogy egy rendőr-ágenst irányítani tudjunk egy rendőr közelébe kell kattintanunk a bal egérgombbal, és légvonalban 2500 méteren belül kell lennie a kattintás térképre vetített pontjának a rendőr helyzetétől. Amennyiben már volt kijelölve egy rendőr, de le szeretnénk venni a jelölést, úgy egyszerűen csak egy olyan pontra kell kattintanunk, amely nem esik bele egy rendőr 2,5 kilométeres körébe sem. Ennek a megvalósítása a 4.5. forráskódrészleten látható.

```
/* ....... */
for (Waypoint w : waypoints) {
  if (w instanceof WaypointPolice) {
    Double tmpDistance = mouseClicked.getDistance(new GPSLocation((
    WaypointPolice) w));
    if (tmpDistance < distance) {
      if (selectedCop != null) {
        selectedCop.setSelected(false);
      }
      if (tmpDistance < 2500) {
        selectedCop = (WaypointPolice) w;
        selectedCop.setSelected(true);
        selected = selectedCop.getId();
        distance = tmpDistance;
      }
    }
  }
}
/* ....... */
```

Forráskód 4.5. A legközelebbi rendőr kiválasztása amennyiben az 2500 méteren belül tartózkodik.

# 5. fejezet

# Jövőbeli munka

TODO további lehetséges fejlesztések részletezése

# 6. fejezet

# Összefoglalás

TODO

# Irodalomjegyzék


[1] AWT. Abstract window toolkit, 2016. URL https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html.

[2] Boost Graph Library. Boost graph library, 2001. URL http://www.boost.org/doc/libs/1_60_0/libs/graph/doc/index.html.

[3] Boost logo. Boost logo, 2005. URL http://boost.cvs.sourceforge.net/viewvc/boost/boost/boost.png?revision=1.2&view==markup.

[4] N. Bátfai. robocar-emulator, 2015. URL https://github.com/nbatfai/robocar-emulator.

[5] N. Bátfai, R. Besenczi, A. Mamenyák, and M. Ispány. Traffic Simulation based on the Robocar World Championship Initiative. *Infocommunications Journal*, 7(3): 50–59, 2015.

[6] C++1998. ISO/IEC 14882:1998, 1998. URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=25845.

[7] C++2003. ISO/IEC 14882:2003, 2003. URL http://www.iso.org/iso/catalogue_detail.htm?csnumber=38110.

[8] C++2011. ISO/IEC 14882:2011, 2011. URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372.

[9] C++2014. ISO/IEC 14882:2014, 2014. URL http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=64029.

[10] R. Carli, M. Dotoli, R. Pellegrino, and L. Ranieri. Measuring and managing the smartness of cities: A framework for classifying performance indicators. In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, pages 1288–1293. IEEE, 2013.

[11] R. De Santis, A. Fasano, N. Mignolli, and A. Villa. Smart city: fact and fiction, 2014. URL https://mpra.ub.uni-muenchen.de/54536/.

[12] C. Du and S. Zhu. Research on urban public safety emergency management early warning system based on technologies for the internet of things. *Procedia Engineering*, 45:748–754, 2012.

[13] Fi-Ware. Fi-Ware, 2015. URL http://www.fiware.org/.

[14] Future City. Future City Glasgow, 2015. URL http://futurecity.glasgow.gov.uk/.

[15] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler-Milanovic, and E. Meijers. Smart cities-Ranking of European medium-sized cities. Technical report, Vienna University of Technology, 2007.

[16] Google Self-driving car image. Google Self-driving car image, 2015. URL http://www.newsnish.com/cars/latest-car-news/robocar--google-deploys-nextgen-cars/.

[17] iCity. iCity Project, 2015. URL http://www.icityproject.eu/.

[18] Java Swing JPanel. Java swing jpanel, 2016. URL https://docs.oracle.com/javase/7/docs/api/javax/swing/JPanel.html.

[19] JavaFX. Javafx, 2016. URL http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784.

[20] JavaName. Why is it called java?, 1991. URL http://mathbits.com/MathBits/Java/Introduction/BriefHistory.htm.

[21] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The Robot World Cup Initiative. In *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, pages 340–347. ACM, 1997.

[22] M. Loy, R. Eckstein, D. Wood, J. Elliott, and B. Cole. In *Java Swing (2 ed.)*, page 53. O'Reilly Media, Inc., 2012.

[23] MyNeighbourhood. MyNeighbourhood Initiative, 2015. URL http://my--neighbourhood.eu/.

[24] Navigant. Smart Cities Navigant Research, 2015. URL http://www.navigantresearch.com/research/smart-cities.

[25] P. Neirotti, A. De Marco, A. C. Cagliano, G. Mangano, and F. Scorrano. Current trends in Smart City initiatives: Some stylised facts. *Cities*, 38:25–36, 2014.

[26] OOCWC Competitions. Competition Result of Debrecen, 2015. URL http://justine.inf.unideb.hu/2015/Europe/Hungary/Debrecen/.

[27] OSM. OpenStreetMap, 2016. URL https://www.openstreetmap.org/about.

[28] R. Weait. Openstreetmap data license is odbl, 2012. URL https://blog.openstreetmap.org/2012/09/12/openstreetmap-data--license-is-odbl/.

[29] Smart City Logo. Smart city logo, 2016. URL https://eu--smartcities.eu/content/spanish-ministry-industry--presents-report-prospect-development-smart-cities--spain.

[30] SmartSantander. SmartSantander, 2015. URL http://www.smartsantander.eu/.

[31] M. Steiger. jxmapviewer2, 2016. URL https://github.com/msteiger/jxmapviewer2.

[32] United Nations. World Urbanization Prospects. The 2007 Revision, 2015. URL http://www.un.org/esa/population/publicationslwup2007/2007/WUP_Highlights_web.pdf.

[33] VITAL. VITAL, 2015. URL http://www.vital-iot.eu/.

[34] T. Yamauchi, M. Kutami, and T. Konishi-Nagano. Development of Quantitative Evaluation Method regarding Value and Environmental Impact of Cities. *Fujitsu Sci. Tech. J*, 50(2):112–120, 2014.

# Függelék

COPCONTROLLER is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
COPCONTROLLER is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with COPCONTROLLER. If not, see <http://www.gnu.org/licenses/>.
CSTS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
CSTS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with CSTS. If not, see <http://www.gnu.org/licenses/>.

# Köszönetnyilvánítás