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: [21].

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

rendszámtáblájukról.

A Drive.ai a 13. cég akik jogosítványt kaptak az autonóm kocsik teszteléséhez Kalifornia publikus utcáin. A cég az autonóm autók továbbfejlesztésén dolgozik úgy, hogy a mély-tanulást (deep-learning) alkalmazza robotautókra. Ez azért kiemelkedően fontos, mert az autonóm kocsik fejlesztésében a kemény rész a szélsőséges esetek (edge cases) megoldása, tehát hogy mit tegyen az autó amikor hirtelen csúszós lesz az út, vagy éppen rosszak a látásviszonyok. Jelenleg az autonóm kocsik fejlesztői különböző szabályokat programoznak be ezekre az esetekre, azonban a mély-tanulás megközelítésével a robotautó megtanulja hogyan reagáljon a különböző esetekre azáltal, hogy a kapott adat teljes egészét próbálja megérteni, és a már megtanult adat alapján reagál rá [17].

A fent említettek alapján tisztán látszik, hogy az autóipar szemléletváltás előtt áll. A nem újrahasznosítható energiaforrásokon alapuló 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: [39].

2050-re a világ lakosságának 70%-a valószínűleg városokban fog élni [47]. 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 [35]. Az informatikai fejlesztések ezen a téren már egyre jobban beférkőztek a hétköznapokba is, mint például az iCity [24], a FIWare [19], vagy a [48]. Ezeken túl egyre többen kísérleteznek azzal, hogy a város embereinek a mindennapjait hatékonyabbá és biztonságosabbá varázsolja ([34], [40]).

Sok tanulmány foglalkozik még azzal, hogy hogyan lehetne egy városnak az „okosságát" megmérni ([16], [14]). A városok „okosságáról" még egy sorrendet is készítettek [20]. Az okos városokban a vészhelyzetek megoldása is fontos szerepet kap [18].

1.1. A dolgozatban bemutatott alkalmazás

A Cop Controller a Robocar World Championship-hez készült megjelenítő, és egyben irányító program is. Azért választott ezt az alkalmazás létrehozását, mert hiányoltam a Robocar World Championship-ből a felhasználó valós-idejű beavatkozásának a lehetőségét.

A program egy Maven (3.4. fejezet) alapú Java (3.1. fejezet) projekt, melynek a grafikus felhasználói felülete a Swing (3.1.4. fejezet), és a JXMapViewer (3.1.5. fejezet) segítségével valósult meg.

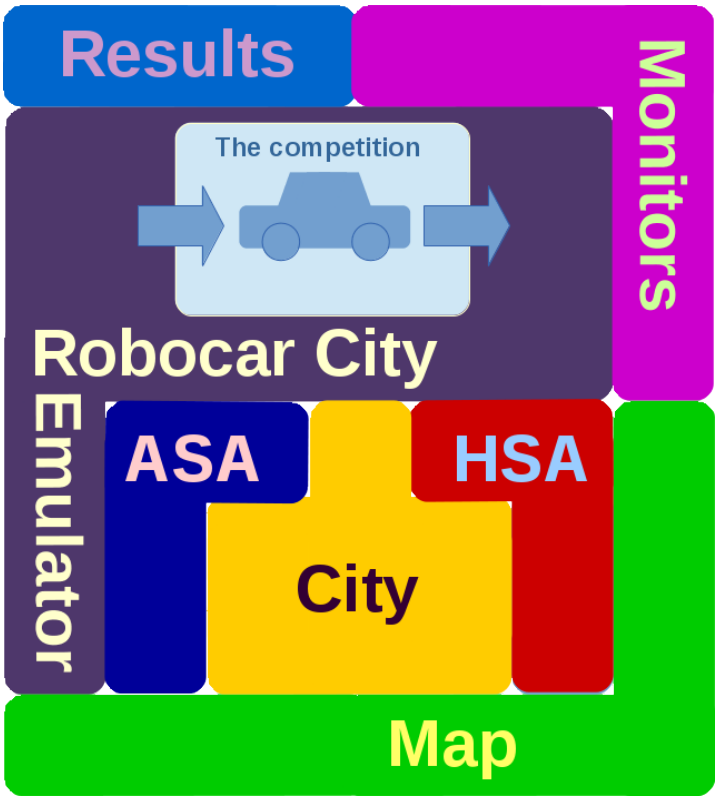
Az alkalmazás valós-időben mutatja meg a felhasználónak a verseny aktuális állapotát a szerver által küldött információk alapján. A felhasználó képes a saját rendőr-ágenseit kiválasztani, és irányítani őket a látott információk alapján.

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: [8].

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 egyrészt egy kutatási platformot kínál forgalomelemzésre, szimulációkra, másrészt pedig egy érdekes versenyzési platformot nyújt.

A verseny célja az, hogy a lehető legjobb forgalomirányító algoritmusokat találjuk meg. Az OOCWC már több sikeres versenyen is túl van a Debreceni Egyetem Informatikai Karán [36]. 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 [29].

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 [9] közleményben.

2.2. Versenyek

A Debreceni Egyetem Informatikai Karán már több Robocar World Championship, Police Edition verseny is lezajlott.

Ahhoz hogy valaki egy ilyen versenyen elindulhasson, elsőnek forkolnia kell az OOCWC repository-ját [8], majd elkészítenie a saját irányító algoritmusát a rendőr-ágensei számára. Ezután fel kell tölteni egy videót, amiben a meghirdetett poszterben foglalt kritériumoknak megfelelően elindított szimulációban 1 rendőrrel teljesítjük az adott idő alatt meghirdetett gengszter elkapási limitet. Ez a limit jelenleg 7 vagy 8 szokott lenni. Ilyen poszterre láthatunk példát a `http://justine.inf.unideb.hu/2015/Europe/Hungary/Debr` `pdf` linken, ami a „Debrecen 2” versenynek a meghirdetési posztere. Az ilyen poszteren láthatunk minden fontos információt a versennyel kapcsolatban.

Miután a videót sikeresen feltöltöttük, létre kell hoznunk egy Team Qualification Paper-t (TQP-t), amivel hivatalosan is tudunk jelentkezni a versenyre. A TQP-t a forkolt `doc/qualification` mappában könnyen el tudjuk készíteni az ott lévő példa segítségével. Ahhoz hogy hivatalosan kvalifikáljunk a versenyre, ezt a dokumentumot kell eljuttatni a szervezőnek.

A versenyen általában 2 csapat mérkőzik meg egymás ellen 10-10 rendőr-ágenst irányítva. A komolyabb irányító algoritmusok itt tudnak megmutatkozni. A csapatmérkőzéseken természetesen az nyer, aki több gengsztert tudott elkapni az előre definiált idő alatt az ellenfelénél, az egész verseny győztesét pedig a verseny előtt definiáltak alapján kapjuk meg a lejátszott mérkőzések után.

Debrecenben eddig 4 verseny volt megszervezve, amelyből 3 dokumentálva is lett az elejétől a végéig, illetve a „Debrecen 5” szervezése jelenleg is tart. Én az első 2 versenyen indultam, ahol az első versenyen 1. helyezést értem el, a 2. versenyen pedig 3. helyen végeztem.

A korábbi (és a jövőbeli) versenyek dokumentumait a következő linken érjük el: `http://justine.inf.unideb.hu/2015/Europe/Hungary/Debrecen/`

A csapatok számát tekintve míg „Debrecen 2” versenyen összesen 5-en indultak (köztük én is), a „Debrecen 3”-on már 43 csapat kvalifikált! Ebből tisztán látszik hogy a Robocar World Championship-re kifejezetten magas érdeklődés mutatható ki az elkövetkezendő évekre.

2.3. 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 [8] 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 --node2gps=debrecenNodes.txt
```

Forráskód 2.1. A SmartCity elindítása

Itt a kapcsolók a következő beállításokat jelentik:

- osm – Az OpenStreetMap térkép elérési útvonala.
  - Alapbeállítás: `../debrecen.osm`
- node2gps – Az OSM térképén belüli csomópontokhoz tartozó GPS koordinátákat ebbe a fájlba írja ki.
  - Alapbeállítás: `../lmap.txt`
- city – A szimulált város neve.
  - Alapbeállítás: `Debrecen`
- shm – Az osztott memória elérési neve.
  - Alapbeállítás: `JustineSharedMemory`

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 a következők lehetnek:

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

    - Alapbeállítás: `10007`

– nrcars – A rutin autók száma.

    - Alapbeállítás: `100`

– minutes – A szimuláció hosszának a száma percben megadva.

    - Alapbeállítás: `10`

– catchdist – A szükséges távolság méterben, amelyen belül egy rendőr-ágens elkap egy gengsztert.

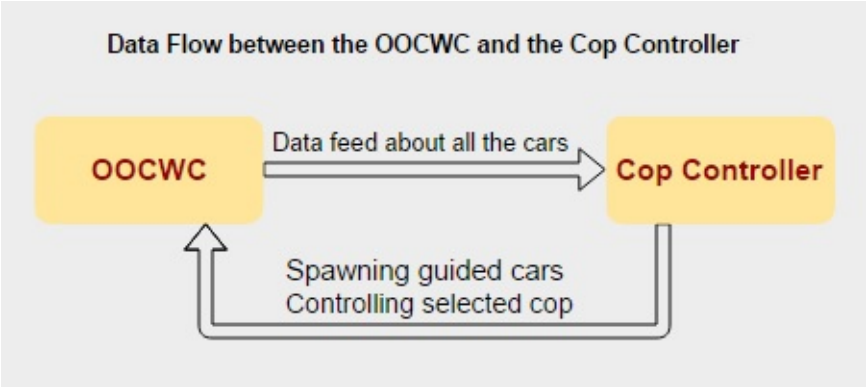
    - Alapbeállítás: `15.5`

– traffict – A rutin autók szimulációs típusa. Lehet:

    - `NORMAL` – alapbeállítás
    - `ANTS`
    - `ANTS_RND`
    - `ANTS_RERND`
    - `ANTS_MRERND`

2.4. A rendszer működése

2.4.1. Az adat áramlása



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

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 jelenlegi állásról.

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

2.5. Szükséges módosítások a projekt számára

2.5.1. 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 [37] (3.6. fejezet) térképből felépített irányított gráf (BGL (Boost Graph Library)) (3.3. 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 elsőként a CarLexer által lesz feldolgozva a 2.3. forráskódrészletben látható módon, ahol a CarLexer elmenti a parancsból kinyert adatokat.

```
/* ....... */
ROUTE "<route"
/* .......*/
{ROUTE}{WS}{INT}{WS}{INT}({WS}{INT})* {
            int size{0};
            int ss{0};
            int sn{0};

            std::sscanf(yytext, "<route %d %d%n", &size, &m_id, &sn);
            ss += sn;
            for(int i{0}; i<size; ++i)
            {
              unsigned int u{0u};
              std::sscanf(yytext+ss, "%u%n", &u, &sn);
              route.push_back(u);
              ss += sn;
            }
```

```
18            m_cmd = 101;
19          }
```

Forráskód 2.3. A CarLexer forráskód részlete az eredeti routing parancsra

Az argumentumok a következők:

- `{ROUTE}` – A forráskódban fentebb definiált értékre hivatkozik.
- `{WS}` – A szóköz karaktert jelenti.
- `{INT}` – Egy egész számot vár itt.
- `({WS}{INT})*` – Bármennyi ismétlődést vár (lehet 0 is) a szóköz és egy egész szám párosából.

Látható hogy elsőként a szerverhez beérkezett parancs validálása a CarLexer által történik, mivel ha a parancs szintaxisa nem egyezik a CarLexer által definiáltakkal, úgy a parancs fel se lesz dolgozva. A 2.3. forráskódrészletben látható továbbá hogy a CarLexer által várt szintaxisnak eleget tesz a fentebb említett route parancs szintaxisa.

Miután a CarLexer feldolgozta a parancsot az irányítás a traffic szerver kezébe kerül, ami kiolvassa a CarLexer által feldolgozott adatokat, és az alapján irányítja a megfelelő rendőrt. Ezt a 2.4. forráskódrészletben láthatjuk.

```
1  try {
2      /* ........ */
3
4      std::shared_ptr<SmartCar> cop = m_smart_cars_map[cl.get_id()];
5
6      if ( cop->set_route ( carLexer.get_route() ) )
7          length += std::sprintf ( data+length, "<OK %d>", carLexer.
       get_id() );
8      else
9          length += std::sprintf ( data+length, "<ERR bad routing vector>
       " );
10
11     /* ........ */
12     boost::asio::write ( client_socket, boost::asio::buffer ( data,
       length ) );
13
14 }
15 catch ( std::exception& e ) {
16     std::cerr << "Error: " << e.what() << std::endl;
17 }
```

Forráskód 2.4. Az eredeti routing forráskód részlete

2.5.2. 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 általuk irányítani a rendőr-ágenseket. (I'm looking at you, Python)

Az új parancs a következőképpen néz ki:

```
<innerroute [rendor-agens ID] [A cel csomopont]>
```

Mint ahogy az eredeti példánál, elsőnek itt is a CarLexer dolgozza fel a kapott parancsot, azonban észrevehető a 2.3. forráskódrészlet és a 2.5. forráskódrészlet közti különbségben hogy az új parancs már nem követeli meg a teljes utat a parancstól.

```
/* ....... */
INNERROUTE  "<innerroute"
/* ....... */
{INNERROUTE}{WS}{INT}{WS}{INT}   {
            std::sscanf(yytext, "<innerroute %d %u>", &m_id, &to);

            m_cmd = 102;
          }
```

Forráskód 2.5. A CarLexer forráskód részlete az új routing parancsra

A parancs feldolgozása után a traffic szerver gondoskodik a rendőr-ágens irányításáról. Lásd: 2.6. forráskódrészlet.

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

    std::shared_ptr<SmartCar> cop = m_smart_cars_map[cl.get_id()];
    long unsigned int from = cop->from();
    long unsigned int to = carLexer.get_to();
    if ( inner_route( cop->get_route(), from, to ) ) {
        length += std::sprintf ( data+length, "<OK %d>", carLexer.
    get_id() );
    else
        length += std::sprintf ( data+length, "<ERR bad routing vector>
    " );
```

```
11
12      /* ........ */
13      boost::asio::write ( client_socket, boost::asio::buffer ( data,
        length ) );
14
15  }
16  catch ( std::exception& e ) {
17      std::cerr << "Error: " << e.what() << std::endl;
18  }
```

Forráskód 2.6. Az új routing forráskód részlete

3. fejezet

Felhasznált technológiák

3.1. Java

A Java programozási nyelv az alapja a hálózati alkalmazások nagy részének, és világkörű szabvány a beágyazott mobil alkalmazások, játékok, webes tartalmak, és a vállalati szoftverek számára. Több mint 9 millió fejlesztővel világszerte a háta mögött, a Java az egyik leghasználtabb és legelterjedtebb programozási nyelv.

Néhány érdekes tény a Java nyelvről:

- 97%-a a vállalati számítógépeknek futtat Javát.
- 89%-a az asztali komputereknek az Amerikai Egyesül Államokban képes Javát futtatni.
- A fejlesztők első számú választása.
- Az első számú fejlesztési platform.
- 3 milliárd mobil eszközön tudnak futni Java alkalmazások.
- 125 millió TV készülékek képesek Java alkalmazásokat futtatni.

A Java úgy lett kialakítva, hogy lehetővé váljon a hordozható, nagy teljesítményű alkalmazások fejlesztése a legszélesebb körű számítástechnikai platformok számára. Azáltal hogy az alkalmazások heterogén környezetekben elérhetővé válnak, a különböző vállalkozások képesek több szolgáltatást nyújtani, és megnöveli a a végfelhasználói produktivitást, kommunikációt és együttműködést – és drasztikusan csökkennek a fenntartási költségek mind a vállalati, mind a fogyasztói alkalmazások körében. A Java felbecsülhetetlenné vált a fejlesztők számára azáltal, hogy lehetővé tette:

– A szoftverek megírását egy platformon, és annak futtatását virtuálisan bármely másik platformon;

– Olyan programok létrehozását, melyek egy internetes böngészőn belül futnak, és hozzáférése van az elérhető internetes szolgáltatásokhoz;

– A Javában írt alkalmazások, illetve szolgáltatások, összeillesztését annak érdekében, hogy magasan személyre szabott alkalmazásokat / szolgáltatásokat fejlesszünk [1].

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

Forráskód 3.1. A klasszikus „Hello World!” Javában

3.1.1. Története

Manapság, amikor a technológia már ennyire része a mindennapi életünknek, magától értetődőnek tartjuk, hogy bármikor és bárhol elérhetővé válnak az alkalmazások és a keresett tartalmak. A Java nyelv miatt napjainkban már elvártnak tekintjük a digitális eszközeinktől hogy több funkcióval rendelkezzenek, okosabbak és jóval szórakoztatóbbak legyenek.

Az 1990-es évek elején a hálózati számítás kierjesztése a mindennapi életre egy radikális elképzelésnek számított. 1991-ben egy kis csoportja a Sun mérnökeinek, akik a „Green Team” csapatnevet viselték, James Gosling irányításával létrehoztak egy új programozási nyelvet – a Javát [22].

3.1.2. Objektumorientáltság

A Java nyelv egyik legfontosabb tulajdonsága az objektumorientáltsága, ami a nyelv felépítésére és stílusára is mutat.

A C++ nyelvvel ellentétben (lásd: 3.2. fejezet) a Java teljesen objektum-orientált: minden egyes Java programnak szüksége van legalább egy osztály jelenlétére. Továbbá a Java eredeti nyelv-definíciója magában foglalja az „objektum-orientált” kifejezést.

Az Objektum Orientált Programozás (röviden OOP) egyik fontos alapköve az egységbezárás (encapsulation). Mikor egy objektumot létrehozunk egy objektum-orientált

nyelven, el tudjuk rejteni az objektum belső működésének a komplexitását a külső világ elől, így a külső világnak csak elég csak az objektum felhasználását „tudnia”.

Másik fontos alapköve az öröklődés (inheritance). Az objektumok származtatása egy szülő-objektumból (superclass) azt a célt szolgálja, hogy a gyerek-objektumok (subclass) átveszik a superclass látható tulajdonságait és metódusait, miközben a kifejezetten csak rájuk tartozó saját tulajdonságokat, illetve metódusokat is tartalmaznak.

3.1.3. Platformfüggetlenség

„Write once, run anywhere” [33].

Általában a lefordított kód pontosan az az instrukcióhalmaz ami szükséges a processzor számára a program „futtatásához”. A Java esetében azonban ez az utasításhalmaz egy virtuális processzorhoz van lefordítva, melynek működnie kell bármely fizikai számítógépen.

A Java esetében a fizikai processzor futtatja a JVM-et (Java Virtual Machine) ami platformfüggő. Ez a virtuális gép fogja aztán futtatni a Java bájt-kódot, ami azonban már platformfüggetlen.

Az egyetlen mód arra, hogy a Java bájt-kódok valóban bármely JVM-en fussanak, az a szigorú szabvány arról, hogyan működnek a Java Virtuális Gépek. Ez azt jelenti hogy bármilyen fizikai platformot használunk, a programunknak az a része, mely a JVM-mel van kapcsolatban, garantáltan csak egy módon fog működni. Mivel mindegyik JVM pontosan ugyanúgy működik, ugyanaz a kód ugyanúgy fog működni mindenhol anélkül hogy újrafordítanánk.

Természetesen van módja a platformfüggetlenségét megtörni egy Java programnak. Ilyen eset például amikor olyan konvenciót használunk, amely csak az egyik operációs rendszerre igaz (például feltételezzük azt, hogy a „:” a könyvtárakat elválasztó szimbólum).

3.1.4. Swing

A Swing API (Application Programming Interface) rengeteg kiterjeszthető komponenst tartalmaz, melyeket felhasználva a programozó könnyen tud Java alapú, grafikus felhasználói felülettel rendelkező alkalmazást létrehozni.

A Swing fejlesztésének a céljai közt volt a korábbi AWT (Abstract Window Toolkit) leváltása, mivel szinte minden, amit a Swing implementál, megtalálható az AWT implementációjában is [4], azzal a különbséggel hogy a Swing komponensei már platformfüg-

getlenek, könnyebben testre szabhatóak, és összességében könnyebb velük dolgozni. A Swing a már az AWT-ből ismert komponensekhez olyan további új komponenseket ad a programozó kezébe, mint például a füllel ellátott panel, különböző fák, táblázatok, vagy 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, így illik ezekre az elemekre a „lightweight" kifejezés [30].

A Swinget a közeljövőben a JavaFX [26] fogja felváltani.

3.1.5. JXMapViewer

A JXMapViewer [42] egy nyílt forráskódú Java könyvtár, ami egy Swing (3.1.4. fejezet) JPanelt [25] szolgáltat, melynek feladata a térkép betöltése, és mutatása.

3.2. C++

A C++ egy nyílt, ISO-szabványosított programozási nyelv. Eleinte ez nem így volt, és a nyelvnek nem volt hivatalos szabványa, és csak egy de-facto szabványt követve volt karbantartva, fejlesztve, azonban 1998 óta [10] a nyelv szabványosítva van az ISO egy bizottsága által.

A C++ egy „compiled” (lefordított) nyelv. Ahhoz hogy egy C++ programot le tudjunk futtatni, elsőnek a C++ fordító segítségével platformfüggő futtatható bájt-kódra kell fordítanunk a kódunkat. Ezáltal a C++ a világ egyik leggyorsabb nyelve, ha a kódunk optimalizálva van.

A C++ nyelv egyik fő erőssége, hogy a programozótól függ hogy mely paradigmákat követi a probléma megoldása érdekében. A C++ támogatja a procedurális, generikus, illetve az objektum-orientált programozási paradigmákat, sok más paradigmát pedig szintén lehetővé téve ezáltal.

```
#include <iostream>

int main()
{
    std::cout << "Hello, vilag!" << std::endl;
    return 0;
}
```

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

A 3.2. forráskód lefordulását a König lookup-nak (más néven Argument Dependent Lookup (ADL)) köszönhetjük, melynek a használatával a függvények névtereit nem muszáj feltüntetni, amennyiben egy, vagy több, argumentumtípus már meg van határozva a függvény névterében. A König-lookup hiányával egy egyszerű „Hello World!” program is első ránézésre bonyolultnak látszik (lásd: 3.3. forráskód).

```
#include <iostream>

int main()
{
    std::operator<<(std::operator<<(std::cout, "Hello World!"), "\n");
    return 0;
}
```

Forráskód 3.3. A klasszikus „Hello World!” C++-ban; König-lookup használata nélkül.

3.2.1. Története

A korai ’80-as években, a Bell Laboratories cégnél, egy új programozási nyelvet kezdtek el fejleszteni, ami a C nyelvet vette alapul. Ennek az új nyelvnek a fejlesztője Bjarne Stroustrup volt, aki C++-nak keresztelte el az „új” nyelvet. Stroustrup azt állította, hogy a C++ célja az, hogy könnyítse és kellemesebbé tegye a jó programok írását. Amikor a nyelvet tervezte, belecsempészte az OOP (objektumorientált programozás) jellemzőit anélkül hogy nagyon módosításokra lett volna szükség a C kódjában. Így a C egy részhalmaza a C++-nak, tehát bármely érvényes C program egyben érvényes C++ program is.

Idővel ahogy egyre jobban fejlődött a nyelv, elkerülhetetlenné vált a szabványosítás. A C++ 1991-ben indult el a szabványosítás útján, majd 1998-ban az ISO szabványosításba bekerült „ISO/IEC 14882:1998” néven [10]. A következő szabványt 2003-ban fogadták el „ISO/IEC 14882:2003” kódnévvel [11]. 8 éve elteltével a következő szabványt is elfogadták, amit „ISO/IEC 14882:2011” kódjelzéssel láttak el [12]. A legújabb C++ szabvány a röviden C++14 névre hallgató, „ISO/IEC 14882:2014” kódú szabvány [13].

3.3. Boost library

A Boost a C++ programozási nyelvhez ad támogatást olyan könyvtárakkal, melyek különböző területek különböző problémáit segítenek megoldani. Többek között ilyen területek a lineáris algebra, a pszeudorandom számok generálása, a több-szálasítás, a képfeldolgozás, a reguláris kifejezések, illetve a unittesztelés.

Az OOCWC (2. fejezet) számára az egyik legfontosabb ilyen könyvtár a BGL (Boost Graph Library) [6] (lásd: 3.3.1. fejezet), melynek segítségével épül fel az OpenStreetMap [37] (3.6. fejezet) térkép alapján az OOCWC irányított gráfja.

3.3.1. Boost Graph Library (BGL)

A gráfok olyan matematikai absztrakciók, amelyek számos számítástechnikai probléma megoldására hasznosak. Következtetésképp ezeknek az absztrakcióknak ugyanúgy helyet kell adni a számítógépes programok körében. A gráffal kapcsolatos algoritmusoknak és adatstruktúráknak a újrafelhasználásához talán az egyik legfontosabb eszköz a szabványosított generikus interfész a gráfok bejárásához.

A Boost Graph Library egy részét az ilyen generikus interfészek alkotják, amely segíti a programozót a gráf struktúrájával való műveletekkel, miközben elrejti az implementáció részleteit előle. A BGL olyan általános célú gráf osztályokkal szolgál a felhasználói felé, melyek megfelelnek a fent említett interfésznek.

A BGL több kereső algoritmussal is szolgál. A teljesség igénye nélkül ezek a következők:

- `BreadthFirstSearch`,
- `DepthFirstSearch`,
- `Dijkstra'sShortestPaths`,
- `Bellman-FordShortestPaths`

Ezek közül az OOCWC a Dijkstra's Shortest Paths, és a Bellman-Ford Shortest Paths algoritmusokat implementálja.

A Robocar World Championship és a Boost Graph Library

Az OOCWC irányított gráf struktúrája a következőképp épül fel:

```
typedef adjacency_list<listS, vecS, directedS,
        property<vertex_name_t, osmium::unsigned_object_id_type>,
        property<edge_weight_t, int>> NodeRefGraph;
```

Forráskód 3.4. Az OOCWC által használt gráf struktúrája.

A 3.4. forráskódrészletben látható generikus paraméterek jelentése a következő:

- `listS` – Az élek típusa. Ez azt jelenti hogy egy `listS` típusú konténerben tárolja a csúcsokhoz tartozó éleket, melyek iterálhatóak.
- `vecS` – A csúcsok típusa. Ez azt jelenti hogy egy `vecS` típusú konténerben tárolja a gráf csúcsainak a listáját.
- `directedS` – A gráf irányítottsága. A `directedS` típus használata az irányított gráfot jelenti.
- `property<vertex_name_t,osmium::unsigned_object_id_type>`:
  - Itt a csúcsok neveinek a típusát határozzuk meg. Esetünkben ez `osmium::unsigned_object_id_type`, ami fordítási időben a `longunsignedint`-re értékelődik ki.
- `property<edge_weight_t,int>`:
  - Itt az élek súlyainak a típusát tudjuk beállítani. Esetünkben `int`, tehát egy egész számot alkalmazunk az élek súlyainak meghatározására.

Miután az OOCWC inicializálta a gráfot az OSM-ből kinyert adatokkal, alkalmazhatjuk rá a fentebb felsorolt kereső algoritmusokat.

Az OOCWC-n a gráfon kereső algoritmusok közül a „Dijkstra's Shortest Paths" algoritmus lett használva. A főbb indokok amiért ez lett választva:

– A „távolság térképet" csak egyszer kell felépíteni, az alkalmazás elindításakor, ezáltal az algoritmus futási idejét, mint problémát, szinte el is lehet felejteni.

– A „Dijkstra's Shortest Paths" algoritmust akkor a legalkalmasabb használni, amikor:

  • A gráf súlyozott,

  • Az élek súlyai nem negatívak.

– A „Dijkstra's Shortest Paths" algoritmus számára a gráf irányítottsága nem számít.

Példa a „Dijkstra's Shortest Paths" algoritmusra az OOCWC-n belül:

```
1 dijkstra_shortest_paths(*nodeReferenceGraph, nodeReference2Vertex[from
    ],
2                         distance_map(distanceMap)
3                         .predecessor_map(predecessorMap));
```

Forráskód 3.5. Az OOCWC által használt útkereső algoritmus hívása.

A 3.5. forráskódrészletben látható paraméterek jelentése a következő:

– `*nodeReferenceGraph` – A gráf objektum, amin alkalmazva lesz az algoritmus.

– `nodeReference2Vertex[from]` – A kiinduló csúcs, ahonnan kezdve az algoritmus a legrövidebb utat számolja a többi csúcsig.

– `distance_map(distanceMap)` – A `DistanceMap` típusú „távolsági térkép".

– `.predecessor_map(predecessorMap)` – A `DistanceMap` típusú objektum által létrehozott `PredecessorMap` típusú objektum. Ezt felhasználva tudjuk megtalálni a legrövidebb utat két csúcs közt.

Végül a fentebb említettek alapján a legrövidebb utat a 3.6. forráskódrészletben látható módot találjuk meg.

```
typedef graph_traits<NodeRefGraph>::vertex_descriptor NRGVertex;
typedef graph_traits<NodeRefGraph>::edge_descriptor NRGEdge;

std::vector<unsigned int> & hasDijkstraPath ( osmium::
    unsigned_object_id_type from, osmium::unsigned_object_id_type to )
    {

    /* ....... */

    dijkstra_shortest_paths(*nodeReferenceGraph, nodeReference2Vertex[
    from],
                            distance_map(distanceMap)
                            .predecessor_map(predecessorMap));
    /* .......*/

    NRGVertex toVertex = nodeReference2Vertex[to];
    NRGVertex fromVertex = predecessorMap[toVertex];

    while(fromVertex != toVertex) {
        NRGEdge edge = edge(fromVertex, toVertex, *nodeReferenceGraph).
    first;
        path.push_back ( (unsigned int) vertexNameMap[target ( edge, *
    nodeReferenceGraph )] );

        toVertex = fromVertex;
        fromVertex = predecessorMap[toVertex];
      }
    path.push_back ( (unsigned int) from );

    std::reverse ( path.begin(), path.end() );

    return path;
}
```

Forráskód 3.6. Az OOCWC által implementált legrövidebb út keresése.

3.4. Maven

A Maven eredetileg egy kísérletként indult azzal a céllal, hogy a Jakarta Turbine projekt [43] build folyamatait egyszerűsítsék. Több projekt volt a saját Ant [3] build fájljaival, melyek alig voltak különbözőek. A Maven célja az (volt), hogy létrehozzon egy szabványosított módszert arra, hogy:

- hogyan legyenek a projektek buildelve,
- tiszta és egyértelmű leírás legyen a projekt függőségeit tekintve,
- könnyen lehessen a projekt információit megosztani.

Az eredmény végül egy olyan eszköz lett, mellyel bármely Java alapú projektet könnyen tudjuk építeni és menedzselni. A Maven működéséhez egy `pom.xml` fájlt kell megadni, amely tartalmazza a projekt felépítéséhez szükséges összes információt. Ilyen információ például a menedzselt projekt függősége is, melyet miután meghatároztunk a POM (Project Object Model) fájlban, a Maven letölti a hozzá tartozó JAR fájlt, és felkerül a projekt classpath-jára, ezáltal máris használni tudjuk a projekten belül az a dependenciát.

Ezeket a függőségeket a Maven az általa üzemeltetett Maven Repository Center-ben [32] keresi, és tölti le. Van lehetőségünk további tárolókat hozzáadni a POM fájlunkhoz, illetve saját magunk is készíthetünk tárolókat.

3.5. Git

A Git egy verziókezelő rendszer, melyet széles körben használnak a szoftverfejlesztésnél és egyéb verziókövetési feladatoknál. Ez egy elosztott forráskódkezelő rendszer, melynél nagy hangsúlyt kap a sebesség [44], az adatok integritása [46], és az elosztott, nem lineáris munkafolyamatok támogatása. A Gitet eredetileg 2005-ben tervezték és fejlesztették a Linux kernel fejlesztői a kernel fejlesztésének támogatására.

Mint a legtöbb elosztott verziókövető rendszereknél, és ellentétben a legtöbb kliens-szerver rendszerekkel, minden egyes Git munkakönyvtár egy teljes értékű Git tároló, teljes előzménnyel, és teljes verzió-követési képességgel, ami független a hálózati hozzáféréstől, vagy a központi szervertől.

A Gitet elsőnek egy alacsony szintű verziókezelő rendszernek tervezték, amire aztán az emberek a saját ízlésüknek megfelelő front-endet fejleszthettek. Ilyen például a Cogito [15]. A Git mára már egy teljes verziókezelő rendszerré nőtte ki magát, és így más programok nélkül is tudjuk használni. Bár a fejlesztést erősen befolyásolta a BitKeeper [5], Torvalds szándékosan kerülte a hagyományos megközelítéseket, amelynek köszönhetően a Git egy egyedi struktúrával jött létre [45].

3.6. OpentStreetMap (OSM)

2015. novemberében a Debreceni Egyetem Informatikai Karán Dr. Jeszenszky Péter adjunktus indított egy „kampányt” a részletesebb Debrecen térkép érdekében. A megvalósítás a hallgatókat bevonva önkéntes alapon történt a HTML/XML kurzus keretében, ahol az OSM XML kapcsán került bemutatásra a projekt, mint egy XML alkalmazás, mintegy 150 fő számára. A hallgatók motiválásképp a sikeres szerkesztésekért extra pontokat kaptak a félév végi érdemjegybe.

Bár a hallgatók közül a vártnál kevesebben vettek részt a Debreceni térkép kiegészítésében, így is sok olyan bejegyzéssel bővült a térkép, mint például az utak sávjainak a száma, ami fontos lehet a további OOCWC kutatások számára.

A továbbiakban a Debreceni Egyetem Informatikai Kar Programtervező Informatikus hallgatóinak a Debrecen térkép minőségének és részletességének a növeléséhez a bevonása várhatóan hagyománnyá fogja kinőni magát.

Az OpenStreetMap egy ingyenes, a közösség által szerkesztett térkép. Célja hogy az egész világot minél részletesebben leíró ingyenes térképet szolgáltassanak a szoftverfejlesztők, illetve felhasználók számára. A térképet bárki szerkesztheti aki az oldalukra (`www.openstreetmap.org`) regisztrál.

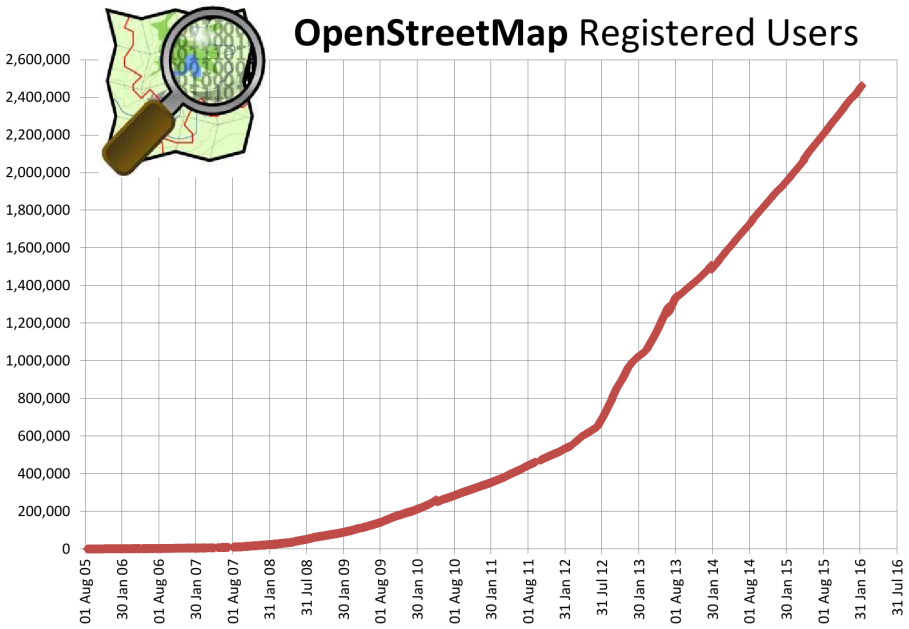
Az OSM létrejöttét Steve Coast-nak köszönhetjük, aki 2004-ben kezdte el a projektet azzal a céllal, hogy Angliát feltérképezze. 2006 áprilisában lett létrehozva az OpenStreetMap Egyesület azzal a céllal, hogy segítse a térinformatikai adatok növekedését, fejlesztését, és forgalmazását a felhasználók körében.

Az önkéntesek egy Java (3.1. fejezet) alapú appleten keresztül tudtak módosításokat felvinni a térképre, vagy valamely offline szerkesztő szoftver használatával. A mai napig az egyik legjobban használt ilyen program a JOSM [28].

2006 decemberében a Yahoo megengedte, hogy az OpenStreetMap felhasználja a légi felvételeit, ezáltal olyan helyeket is sikerült az OSM-nek ledokumentálnia, melyekhez eddig nem tudtak hozzáférni. Fél évvel később kiadták a Potlatch [38] térképszerkesztőt, ezáltal segítve a robbanásszerűen növekedő felhasználóbázis integrációját. Egy példát a Potlatch térképszerkesztő 2006-os verziójára a 3.2(a). ábrán láthatunk.

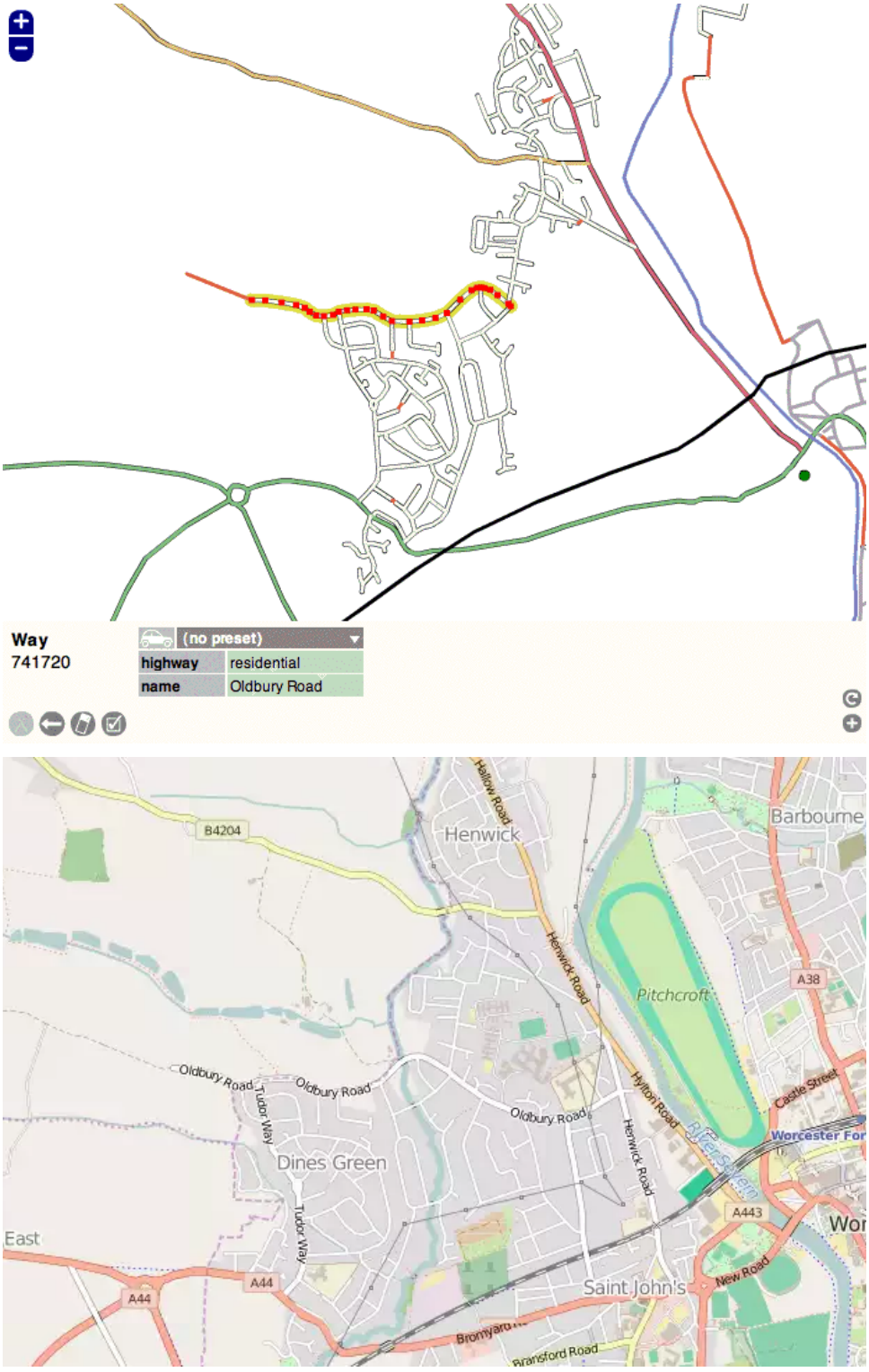
Az állandó térképszerkesztő szoftverek fejlesztéseinek köszönhetően 2009-től kezdve az OSM egyre jobban elterjedt, a felhasználó- és szerkesztőbázisa pedig exponenciálisan növekedett (lásd: 3.1. ábra), ezáltal a térkép részletessége is gyorsan javult (3.2(b). ábra).

A mai napra már annyira megbízható térképpé nőtte ki magát az OpenStreetMap – hála az egyre több és több önkéntes szerkesztőnek – hogy sok nagyobb (mobil)alkalmazás



3.1. ábra. Az OpenStreetMap regisztrált felhasználóinak száma. Forrás: [23].

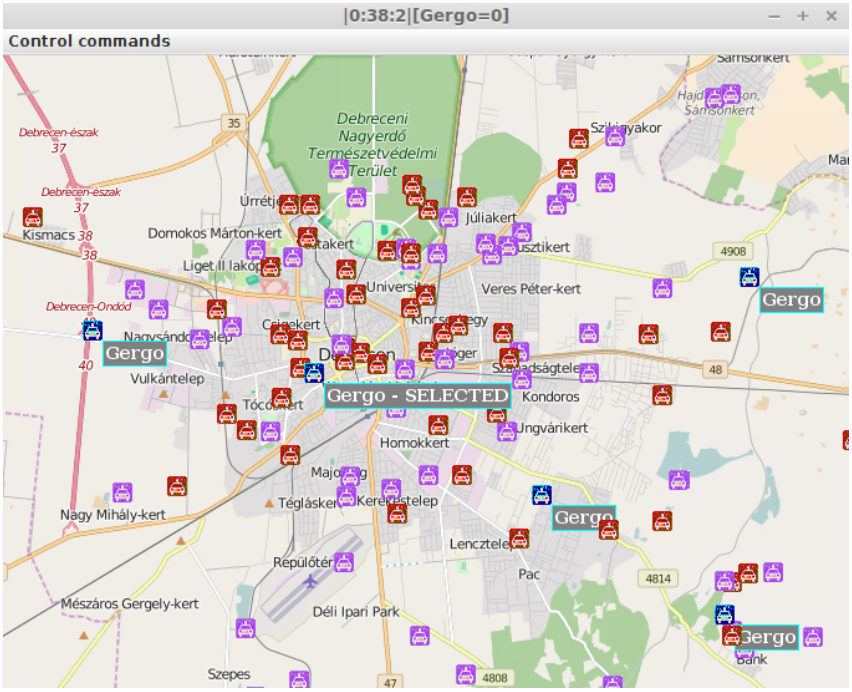
is az OSM-re épült. Ilyen például, egy a sok közül, a Maps.me [31] mobilalkalmazás.



3.2. ábra. A Potlatch-ben szerkesztett Worcester térképe 2006-ban. Forrás: [23], és ugyanaz a térképrészlet napjainkban. Forrás: [37].

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 [42] 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: [9]

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 (3.4. fejezet) által menedzselt Java (3.1. fejezet) alkalmazás, amihez egy Swing (3.1.4. fejezet) grafikus felhasználói felület (Graphic User Interface)

tartozik. A projekt függőségeit pedig a Maven tartja karban a `pom.xml` fájl alapján.

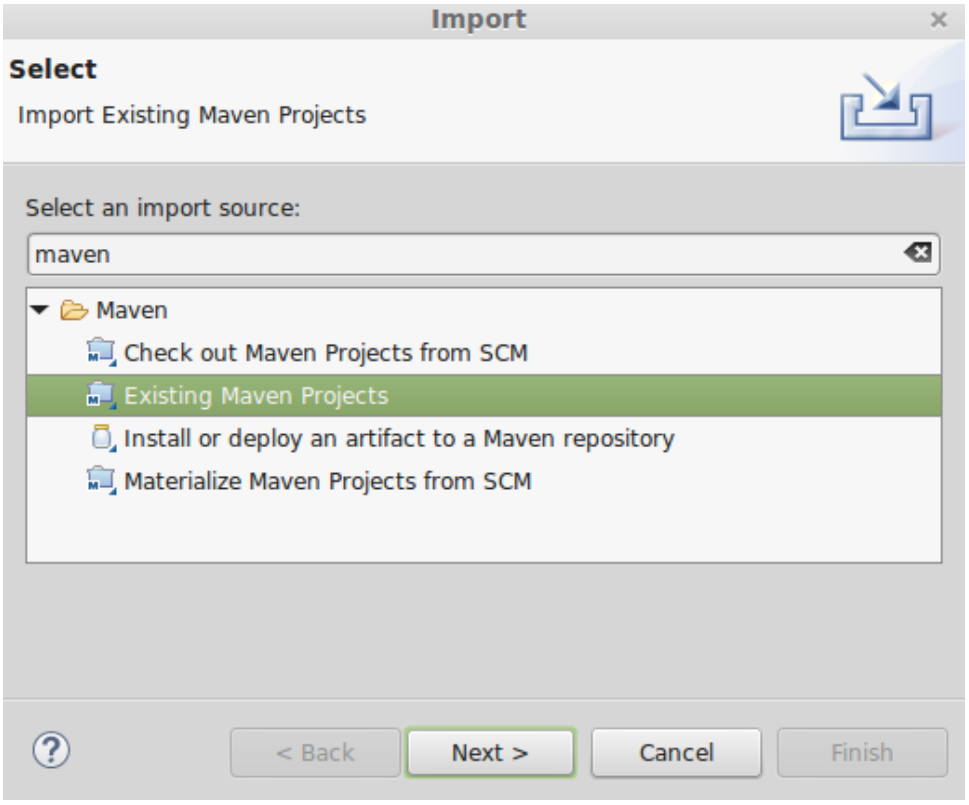
Ilyen függőség például JXMapViewer2 [42] (3.1.5. fejezet)).

4.1. Az alkalmazás elindítása

Az alkalmazás jelenlegi változata az Eclipse IDE alatt fut hibamentesen. Ennek az okát, illetve lehetséges javítására ötletet a 5. fejezetben lehet olvasni.

Az alkalmazást az Eclipse elindítása után a „File” menüponton belül az „Import...”-ot választva tudjuk maven projektként importálni.

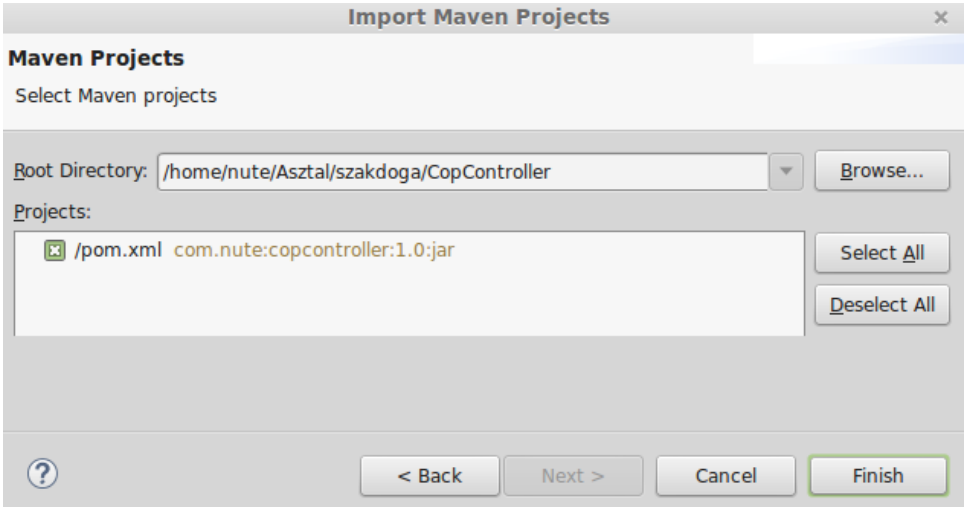
Miután felugrott az ehhez kapcsolódó panel, válasszuk ki az „Existing Maven Projects” opciót (lásd: 4.2. ábra).



4.2. ábra. Maven project importálása Eclipse IDE-be. 1. rész.

Miután „Next”-re kattintottunk meg kell adnunk a relatív elérési útját a mappának melyben `pom.xml` fájl van. Miután azt megadtuk az Eclipse be tudja importálni a projektet (lásd 4.3. ábra).

Importálás után a `CopController.launch` futtatási konfigurációban módosítani kell a `PROGRAM_ARGUMENTS` kulcs értékét a SmartCity –node2gps kapcsolója által létrehozott fájlnak a relatív elérési útjára.



4.3. ábra. Maven project importálása Eclipse IDE-be. 2. rész.

A módosítás elvégzése után futtathatjuk a konfigurációs fájlunkat, melynek tartalmát a 4.1. forráskódrészletben láthatunk.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<launchConfiguration type="org.eclipse.jdt.launching.
    localJavaApplication">
    <stringAttribute key="bad_container_name" value="/CopController/
    eclipse/run.launch"/>
    <listAttribute key="org.eclipse.debug.core.MAPPED_RESOURCE_PATHS">
        <listEntry value="/CopController/src/main/java/com/nute/
    copcontroller/ui/CopController.java"/>
    </listAttribute>
    <listAttribute key="org.eclipse.debug.core.MAPPED_RESOURCE_TYPES">
        <listEntry value="1"/>
    </listAttribute>
    <booleanAttribute key="org.eclipse.jdt.launching.
    ATTR_USE_START_ON_FIRST_THREAD" value="true"/>
    <stringAttribute key="org.eclipse.jdt.launching.CLASSPATH_PROVIDER"
     value="org.eclipse.m2e.launchconfig.classpathProvider"/>
    <stringAttribute key="org.eclipse.jdt.launching.MAIN_TYPE" value="
    com.nute.copcontroller.ui.CopController"/>
    <stringAttribute key="org.eclipse.jdt.launching.PROGRAM_ARGUMENTS"
    value="/home/nute/debrecen-lmap.txt"/>
    <stringAttribute key="org.eclipse.jdt.launching.PROJECT_ATTR" value
    ="CopController"/>
    <stringAttribute key="org.eclipse.jdt.launching.
    SOURCE_PATH_PROVIDER" value="org.eclipse.m2e.launchconfig.
    sourcepathProvider"/>
```

```
16 </launchConfiguration>
```

Forráskód 4.1. A CopController.launch futtatási konfigurációja

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 (lásd: 4.1. ábra).

Ezt az információt a traffic szervertől kapja az alkalmazás egy socketen keresztül. Ebben az információhalmazban benne van az összes autó információja – beleértve a semleges autókat – is.

Erre példát a 4.1. ábrán láthatunk.

4.2.2. Rendőrök és gengszterek hozzáadása

Az alkalmazásban lehetőség van:

- 1 rendőr hozzáadására (4.2. forráskód);
- 10 rendőr hozzáadására (4.3. forráskód);
- 100 gengszter hozzáadására (4.4. forráskód).

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

Forráskód 4.2. 1 rendőr hozzáadása; ahol a „Gergo” a rendőr csapatnevét jelenti.

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

Forráskód 4.3. 10 rendőr hozzáadása; ahol a „Gergo” a rendőrök csapatnevét jelenti.

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

Forráskód 4.4. 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 2.4.1. 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 4.5. 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 4.5. 100 gengszter hozzáadása a 4.4. 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 4.6. 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) {
```

```
10          selectedCop = (WaypointPolice) w;
11          selectedCop.setSelected(true);
12          selected = selectedCop.getId();
13          distance = tmpDistance;
14        }
15      }
16    }
17  }
18  /* ....... */
```

Forráskód 4.6. A legközelebbi rendőr kiválasztása amennyiben az 2500 méteren belül tartózkodik.

5. fejezet

Jövőbeli munka

5.1. Mesterséges intelligencia alkalmazása

Az OOCWC rendőr változatának egyik fő célja volt hogy mesterséges intelligenciát használva irányítsuk a rendőr-ágenseket. Most, hogy már lehetőség van a rendőröket más programozási nyelvvel is irányítani, ugyanúgy fenn áll a lehetősége – azon a programozási nyelven – a mesterséges intelligencia implementálásának.

5.2. Többszálasítás

Itt a többszálasítás alatt főként az adatok érkezésének, és annak feldolgozásának, illetve az adatok küldésének a szálankénti szétválasztására gondolok.

Jelenleg ez azért nem lehetséges, mert mivel csak egy traffic server fut egy ip-címen és porton, így csak egy socketet tudunk hozzá kötni. Az az egy socket jelenleg a `<disp>` parancs számára van fenntartva, hogy az aktuális állás megjelenítése folyamatos legyen.

Ahhoz, hogy egy sockettel működni tudjon az alkalmazás több szálon, arra lenne szükség hogy 200 ms-ba férjenek bele a következő műveletek:

- A socket lefoglalása, hogy csak a megjelenítő szál használhassa (lock),
- A `disp` parancs elküldése,
- A kapott adatok feldolgozása,
- Az állás frissítése,
- A socket feloldása, mivel ez a szál jelenleg már nem használja a socketet,
- Amennyiben az alkalmazás funkciói közül valamelyiket használni szeretnénk:

- A socket lefoglalása a küldő szál számára,
- A parancs létrehozása, és elküldése,
- A socket feloldása.

Amennyiben olyan eset alakul ki, amikor az egyik szál erőforrásra vár – pontosabban a socket feloldására – akkor könnyen lefagyhat a várakozási idő hosszára a programunk, illetve valószínűleg nem lesz folyamatos a megjelenítés, ami úgy vélem, egy ilyen alkalmazásnál kritikus szempont.

5.3. Webalkalmazás létrehozása

Az egyik legígéretesebb fejlesztésnek a webalkalmazássá történő módosítást látom.

Mivel egy ilyen webalkalmazásnak a back-end részének a business logikája megegyezik a jelenleg is használttal az alkalmazásban, így a fejlesztések fő része a webalkalmazássá való átírásban lenne.

Technológia szempontjából úgy gondolom hogy a back-end egy Spring [41] alapú webalkalmazásból állna, ami egy Jetty [27] szerveren fut. Front-end részről mindenképp AngularJS [2] a választásom a Bootstrap [7] segítségével.

Ezt azért tartom talán a legjobb továbbhaladási iránynak, mert ezáltal akár egyszerre az összes versenyző is tudná irányítani a rendőreit valós-időben, így a versenyek még jobban a valós-idejű vetélkedésre fókuszálnának.

A kinézetét, illetve működését, úgy tudom elképzelni, hogy a versenyző megnyitja a szerverhez tartozó weboldalt, ahol be kell elsőnek jelentkeznie (megadja a csapatnevét), és bejön a térkép. Az ilyen webalkalmazás tipikusan egy SPA (Single Page Application), és az AngularJS kifejezetten az ilyen alkalmazások számára lett fejlesztve.

A térképen való navigáláson, és a saját rendőr kijelölésén nem változtatnék a mostani implementációhoz képest.

6. fejezet

Összefoglalás

Az alkalmazás célközönsége a Robocar World Championship résztvevői, akik a jelentkezéshez (vagy akár az éles meccsekhez is) tudják használni a programot. Mint ahogy azt fentebb említettem, ennek természetesen előnyei, és hátrányai is vannak. Számomra hatalmas előny, hogy a 10 perces futási idő alatt folyamatosan gondolkodnom és strategizálnom kellett, hogy az aktuális rendőrömet merre irányítsam, hogy a lehető legtöbb gengsztert elkapjam, mivel minden egyes gengszter elkapása örömmel töltött el, tudván hogy a stratégiám bevált.

Az egyik hátrány ugyanúgy abból következik hogy valós időben bízzuk a felhasználóra a rendőr irányítását. Ezzel egyértelműen minimális hátrányba kerül a többi mesterséges intelligenciával ellátott rendőr-ágenssel szemben.

A másik hátrány szintén a valós-idejűségből, és az ember reakcióideje és a mesterséges intelligencia reakcióidejének az összehasonlításából adódik. Amíg a mesterséges intelligenciával ellátott rendőr-ágens minden egyes lépésben újratervezi az aktuális adatok alapján hogy merre tudná elkapni leghamarabb a következő gengsztert, addig az embernek kell legalább 5-10 lépés, mire az adatokat a térképről megérti, a megfelelő rendőrágensét kiválasztja, és azt akarata szerint irányítja.

Ahogy láthatjuk ezek a hátrányok az ember és a mesterséges intelligencia közötti különbségek által jöttek létre, így szerintem a további versenyeket érdemes lenne két részre bontani:

1. real-time versenyek, ahol a rendőröket mindenki maga irányítja,
2. normális versenyek, ahol a cél továbbra is a jobb mesterséges intelligencia fejlesztése, mint ahogy azt eddig is láthattuk az OOCWC kapcsán.

Irodalomjegyzék

[1] About Java. About the java programming language., 2016. URL https://www.java.com/en/about/.

[2] AngularJS. Angularjs, 2016. URL https://angular.io/.

[3] Apache Ant. Apache ant, 2016. URL http://ant.apache.org/.

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

[5] BitKeeper. Bitkeeper, 2015. URL http://www.bitkeeper.com/.

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

[7] Bootstrap. Bootstrap, 2015. URL http://getbootstrap.com/.

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

[9] 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.

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

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

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

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

[14] 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.

[15] Cogito. Cogito, 2006. URL http://git.or.cz/cogito/.

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

[17] Drive.ai. Drive.ai brings deep learning to self-driving cars, 2016. URL http://spectrum.ieee.org/cars-that-think/transportation/self-driving/driveai-brings-deep-learning-to-selfdriving-cars/.

[18] 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.

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

[20] 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.

[21] 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/.

[22] History of the Java language. History of the java language, 2016. URL http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html.

[23] History of the OpenStreetMap. History of the openstreetmap, 2016. URL http://wiki.openstreetmap.org/wiki/History_of_OpenStreetMap.

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

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

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

[27] Jetty. Jetty, 2016. URL http://www.eclipse.org/jetty/.

[28] JOSM. Josm, 2016. URL https://josm.openstreetmap.de/.

[29] 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.

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

[31] Maps.me. Maps.me, 2016. URL http://maps.me/en/home.

[32] Maven Repository Center. Maven repository center, 2016. URL https://maven.apache.org/repository/index.html.

[33] S. Microsystems. Javasoft ships java 1.0. *Java's write-once-run-everywhere capability along with its easy accessibility have propelled the software and Internet communities to embrace it as the de facto standard for writing applications for complex networks*, 1996.

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

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

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

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

[38] Potlatch. Potlatch, 2015. URL http://wiki.openstreetmap.org/wiki/Potlatch_1.

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

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

[41] Spring. Spring, 2016. URL https://spring.io/.

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

[43] The Jakarta Turbine Project. The jakarta turbine project, 2016. URL https://turbine.apache.org/.

[44] L. Torvalds. "re: Kernel scm saga..". *linux-kernel (Mailing list). "So I'm writing some scripts to try to track things a whole lot faster."*, 2005.

[45] L. Torvalds. "re: Vcs comparison table". *git (Mailing list). A discussion of Git vs. BitKeeper*, 2006.

[46] L. Torvalds. "re: fatal: serious inflate inconsistency". *git (Mailing list). A brief description of Git's data integrity design goals.*, 2007.

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

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

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/>.

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőm, és egykori tanárom, dr. Bátfai Norbertnek, akinek őszinte tanácsai nélkül ezt a munkát ma nem olvashatná a kedves olvasó. Szeretném továbbá megköszönni, hogy mint akkori prog2-es tanuló, részese lehettem a Robocar World Championship létrejöttének, és fejlődésének.

Szeretnék megköszönni az újabb, reguláris oktatásban résztvevő, prog1-es és prog2-es hallgatóknak is, hogy tovább formálják, éltetik, és folyamatosan gyarapítják az OOCWC közösségét.

Szeretnék továbbá köszönetet mondani régi, jó barátomnak, és kollégámnak, Iváncza Csabának, a folyamatos támogatásáért, és a technikai beszélgetéseink által az új perspektívák megnyitásáért számomra.

Hálásan köszönöm kedvesem kitartó türelmét és a mindennapi feladatokban nyújtott segítségét.

Szeretném megköszönni szüleimnek azt a sok gondoskodást és törődést, ami végigkísért tanulmányaim során.

Végül pedig szeretném Önnek, kedves olvasó, megköszönni hogy elolvasta ezt a munkát.