MESTERSÉGES INTELLIGENCIA ÉS GÉPI TANULÁS VERSENY 2025.

2025. november 20.

1 Bevezetés

A városi kutatás és mentés (Urban Search and Rescue, USAR) a katasztrófavédelem fontos területe. Magában foglalja egy katasztrófa sújtotta területen található sérültek felkutatását, állapotuk stabilizálását és kimentését a területről.

Az USAR mentőegységek nehéz feladatokkal nézhetnek szembe egy rendkívül (robbanás-, omlás-) veszélyes környezetben. De a gyakorlati példák azt mutatják, hogy még hosszú órákkal vagy napokkal a katasztrófa bekövetkezte után is van esélyük túlélőkre bukkanni, így fontos, hogy működésüket megfelelően előkészítve és megtervezve végezzék. Céljuk, hogy minél több sérültet mentsenek ki a mentésen dolgozók életének minél kisebb mértékű kockáztatásával. A versenyzők feladata a robotok vezérlésének programozása.

A szimulációt és a robotágenseket működtető keretrendszer egy Java alkalmazás, ennek megfelelően az ágensek kódját is Java nyelven kell elkészíteni. A programozás megkönnyítése érdekében kiadunk mintaprogramokat, valamint a keretrendszer számos segédfüggvényt is tartalmaz.

A rendszer több, mintaként kiadott térképet is tartalmaz. A fejlesztést érdemes a “Default” térképen kezdeni.

2 Az ágensek

A keretrendszer négyféle robot vezérlését támogatja: *Rescue, Medic, Drone* és *Sensor*.

- A Rescue egy szállítórobot, alapvető célja sérültek mentése.
- A Medic is alkalmas szállításra, de elsődleges célja a sérültek életben tartása.
- A Drone egy alacsony költségű repülő robot, amely a kezdetben ismeretlen terep felderítésére alkalmas.
- A terepen előzetesen "ledobhatók" Sensor ágensek, amelyek a ledobás környezetéről nyújtanak valósidejű információkat.

2.1 Az ágensek működése

Az ágensek működtetése a szokásos észlelés - következtetés - cselekvés működési modell szerint történik.

A szimuláció során az ágensek vezérlőprogramjait a keretrendszer aktiválja a `step()` metódusuk meghívásával. A robotvezérlők ebben a metódusban határozzák meg (következtetés) a robotok elvégzendő cselekvéseit az érzékeléseik alapján.

Az ágensek csak a közvetlen környezetük érzékelésére képesek (van egy meghatározott látósugaruk, és nem látnak át falakon), így a világról mindig csak korlátozott információkkal rendelkeznek. Az egyes robotok természetesen megoszthatják egymással észleléseiket és más információkat. A keretrendszer erre a célra biztosít egy "belső" világmodellt (`internalWorldMap`), amely megkönnyíti az érzékelést és az információmegosztást. A szimulátor a világ ágensek által érzékelt részeit automatikusan leképezi a belső világmodellre. Ez a belső modell további könnyítésként tartalmaz néhány segédfüggvényt is, például útvonalkeresést a legközelebbi sérültig. Fontos azonban azt tudni, hogy a belső világmodell nem feltétlenül tükrözi a teljes aktuális valóságot (lehetnek felfedezetlen területek, illetve egyes cellák állapota futásidőben változhat), és a segédfüggvények sem feltétlenül az optimális megoldást szolgáltatják.

Az ágensek közötti együttműködés kialakítása egy fontos része a feladatnak. Az ágensprogramok mellett kiadott `AMSService` osztályban a belső világmodell mellett kialakíthatók további közös funkciók (a `moveRobotAlongPath` mintájára) és adatstruktúrák, amelyek minden ágens számára elérhetők.

Az ágensek cselekvései meghatározzák a mozgásukat (lépés előre, hátra, jobbra illetve balra), illetve a sérültekkel kapcsolatos akcióikat (szállítás, gyógyítás). A kiadott minta-ágensvezérlőkben láthatók a lehetséges cselekvések. A `step()` metódus visszatérési értéke határozza meg az ágens cselekvését.

A robotoknak véges mennyiségű energia áll a rendelkezésükre a cselekvések végrehajtására (`robot.getBatteryLifeTime()` metódus). Ha elfogy, a robot nem tud több akciót végrehajtani, és a környezetét sem érzékeli a továbbiakban. Például a szenzorágensek meglehetősen kevés energiával rendelkeznek, így működésük a szimuláció közben jellemzően leáll. A Medic robot számára az energia mellett a sérültek állapotának javítására használt orvosság is véges erőforrás (`robot.hasMedicine()`).

3 A keretrendszer bemutatása

A keretrendszer egy Java alkalmazás. Feladata kettős: egyrészt az USAR környezet grafikus szimulátora, másrészt egy futtatórendszer és programozói felület az ágensekprogramok számára.

3.1 Grafikus szimulátor és pontszámítás

Az alkalmazás a RescueFramework főosztályának `main()` metódusával futtatható. A szimuláció elindítása előtt kiválasztható egy térkép, és megadható az egyes ágensek száma típusonként. Ezután elindítva a szimulációt a grafikus felületen nyomon követhetjük a robotok működését és a sérültek állapotának alakulását.

A keretrendszer egy pontszámmal folyamatosan értékeli a versenyzők által programozott ágensrendszer teljesítményét. A megmentett sérültekért pontokat ad (élve kimentett sérültenként +500 pont, a kimentett elhunytakért +200 pont jár), míg a robotok beszerzése költséggel (negatív pontszámmal) jár (rescue robot esetén -300 pont, medic esetén -400 pont, drone esetén -50 pont, és a statikus kamerák -10 pont). A robotok cselekedetei energiát használnak fel (csökkentik a pontszámot: medic vagy rescue robot mozgása: -2 pont lépésenként, `idle` vagy `heal` esetén -1 pont szimulációs lépésenként. A kamerák és drónok fenntartása szimulációnként lépésenként -1 pont.). A keretrendszer futásidőben folyamatosan kijelzi a pontszám részletes számítását.

3.2 A futtatórendszer működése

A program futása során az ágensek programjait a keretrendszer hívja meg.

A szimuláció körökre osztott, minden körben (lépésben) a keretrendszer egymás után egyesével aktiválja az ágenseket (`step()` metódus), és válaszként várja az ágens által végrehajtani kívánt cselekvést (lépés valamilyen irányban, gyógyítás, sérült felemelése stb.), amit azonnal végrehajt a szimulált világban, amennyiben az lehetséges és engedélyezett az ágens számára (a robot megpróbálhat belépni egy falba, de egy ilyen cselekvést a keretrendszer nem fog végrehajtani). Az ágens úgy is dönthet, hogy semmilyen cselekvést nem végez, nem használ fel további energiát (pl. egy drón már felfedezte a teljes terepet, így leáll).

A szimuláció akkor fejeződik be, ha egyetlen ágens sem hajt végre cselekvést, vagy valamelyik ágensprogram hibát dob.

3.3 Véletlen események

A Sensor ágensek elhelyezése véletlenszerű, a versenyzőknek csak az ágensek számára és körülbelüli helyére (lásd `generateX|YCoord()`) van hatása.

A szimuláció során a a keretrendszer véletlenszerűen megváltoztathatja a térképet azzal, hogy egyes falakat "ledönt". Ezeket a változásokat a grafikus szimulátor pirossal jelzi – ezt az információt azonban az ágensek csak akkor észlelik, ha az a látókörükben történik, vagy arra járva felfedezik. Ezek a változások a világ valóságos és az ágensek által használt belső modellje között eltéréseket eredményeznek, amíg egyetlen ágens sem észleli azokat.

Annak érdekében, hogy a véletlen események ne befolyásolják a versenyt, illetve a fejlesztést (a szimuláció lefutása reprodukálható legyen), a keretrendszer a Java beépített álvéletlen-generátorát rögzített seed értékkel inicializálja (lásd `Java.util.Random()`). A seed értéke azonban eltérhet a keretrendszer kiadott és értékelésre használt változataiban.

4 Programozás

A kiadott keretrendszeren (szimulátoron) belül három csomag található: a `rescueagents` tartalmazza a robotokat vezérlő ágens osztályt, a `rescueframework` a keretrendszer forrásfájljait, míg a `world` package a világot leíró fájlokat tárolja.

4.1 Az ágensek programozása

A `rescueagents` package `*RobotControl`, illetve `AMSService` osztályaiban kell az ágensek intelligenciáját implementálni. A `RobotControl` osztályok az `AbstractRobotControl` leszármazottjai, ahol a robotok egyedi számmal történő ellátása került implementálásra. A robot a `RobotPercepcion` interfészen keresztül érzékeli a külvilágot (más módon nem férhet hozzá annak állapotához), amely számos segédfüggvényt (útvonalkeresési algoritmust) is tartalmaz az ágensek megvalósításának megkönnyítésére. Ezek az eljárások nem feltétlenül vesznek figyelembe minden tényezőt (pl. futásidejű változásokat), így nem feltétlenül optimálisak.

A robotvezérlők a robot állapotáról a RobotInterface segítségével kaphatnak információkat (pl. a robot helye, van-e sérült vagy gyógyszer nála stb.).

Az egyes ágensek egymással információkat oszthatnak meg közös adatstruktúrákon (pl. az AMSService osztályon) keresztül. A kooperáció ügyes kialakítása elengedhetetlen a jó pontszám eléréséhez.

A robotok cselekvésüket a fentiekben leírt érzékelések (külvilág, robot) és más ágensektől kapott információk alapján alakítják ki, és a step() metódus visszatérési értékeként adják meg a keretrendszer számára. A visszaadott érték alapján a szimulátor ellenőrzi, hogy az adott akció az ágens számára engedélyezet-e (lehet-e arra lépni, gyógyítani vagy sérültet felvenni), és amennyiben igen, végrehajtja az akciót. Az ágensprogramok más módon nem befolyásolhatják a világot, nem mozgathatják a robotokat, nem szállíthatnak vagy gyógyíthatnak sérülteket.

4.2 Útvonalkeresés

A feladat egyik kulcseleme a hatékony útvonalkeresés megvalósítása. A `RobotPerception` interfészek keresztül elérhető beépített útvonalkereső hívásai helyett saját algoritmusok is készíthetők. Ennek során kívánság szerint használható a `world` package `AStarSearch` statikus osztálya, mellyel A* keresést lehet megvalósítani két cella között a robotok által ismert cellákat figyelembe véve:
`public static Path AStarSearch.search(Cell start, Cell target, int maxDistance).`

Ez az algoritmus A* algoritmussal kiszámítja a két cella közötti legrövidebb, maximum `MaxDistance` hosszú utat. A `maxDistance` értékét -1-re állítva nincs korlátozás a megtalált út hosszára. Ha nem található út a két cella között, akkor a függvény `null` értékkel tér vissza. Az algoritmus megvalósítási részletei a keretrendszer kiadott forráskódjában tanulmányozhatók.

Természetesen saját útvonalkereső algoritmus is implementálható a cellák adataira támaszkodva.

4.3 Kooperáció

Bár az ágensek közötti kommunikációval a feladatban nem kell foglalkozni, a kooperáció kialakítása fontos a robotközösségben.

Egy jó megoldás elkészítéséhez az ágensek közötti feladatmegosztással is érdemes foglalkozni. Ehhez nyújt segítséget az `AMSService` osztály, amelyben további adatstruktúrák és eljárások hozhatók létre ebből a célból.

4.4 Java-programozás

Azon versenyzőknek, akik már régen foglalkoztak Java-programozással, készültünk néhány megjegyzéssel:

1. A Java egyenlőséget érték szerint csak primitív típusok esetén végez. Minden más esetben alapvetően referencia-egyenlőséget vizsgál!

2. A Java (ellentétben például a Python-nal) statikusan típusos nyelv. A feladat megoldásához bizonyos esetekben szükség van castolásra.

5 Mintaprogramok

Az ágensek vezérlői alapvetően egyszerű felépítéssel rendelkeznek. Például a kiadott Rescue mintaágens programjának lényegi része:

```
if (!robot.hasInjured()) { // ha nem szallitunk serultet
    if (robot.getLocation().hasInjured()) { // es egy serultnel vagyunk
        return Action.PICK_UP; // vegyuk fel
    } else { // egyebkent keressunk egyet
        path = internalWorldMap.getShortestInjuredPath(robot.getLocation());
    }
} else { // ha van nalunk serult
    if (robot.getLocation().isExit()) { // es a kijaratnal vagyunk
        return Action.PUT_DOWN; // tegyuk le
    } else { // egyebkent menjunk a kijarathoz
        path = internalWorldMap.getShortestExitPath(robot.getLocation());
    }
}
```

A Medic ágens programja:

```
public Action step() {
    if (robot.hasMedicine() // ha van gyogyszer
            && robot.getLocation().hasInjured() // es egy serultnel vagyunk
            && robot.getLocation().getInjured().isAlive() // aki eletben van,
            //es gyogyitando:
            && robot.getLocation().getInjured().getHealth() < Injured.MAXHEALTH / 3) {
        return Action.HEAL; // gyogyitunk
    }
    // egyebkent a Rescue robot programjat futtatjuk (serulteket keresunk)
    return fallbackRobotControl.step();
}
```

Természetesen ennél ötletesebb, más stratégiákat alkalmazó, és akár egymással kooperáló ágenseket célszerű készíteni.

6 Térképek

A keretrendszerrel együtt kiadunk egy térképkészletet is különféle méretű problémákkal. A versenyzők ezeket a fejlesztés során használhatják a megoldásuk előzetes értékelésére.

A kiadott szoftvercsomag tartalmaz egy térképkészítő alkalmazást is, amellyel további terepelrendezések is kialakíthatók.

A beküldött megoldásokat előre nem ismert térképeken értékeljük.

7 Értékelés

A verseny végeredménye az ismeretlen pályán a keretrendszer által adott pontok alapján alakul ki. A verseny győztese az, aki a legtöbb pontot szerzi meg, a további helyezések a pontszámok szerinti csökkenő sorrendben kerülnek kiosztásra.

Pontegyenlőség esetén az a megoldás kerül a rangsorban előrébb, mely kevesebb energiafogyasztással jár. További pontegyenlőség esetén az a versenyző végez előrébb, aki hamarabb nyújtotta be a megoldást.

A megoldásokat két tervezési időben ismeretlen tesztpályán értékeljük. Az egyik pálya bonyolultabb, és minden ágenstípusból 1-1 példányt indít el. A második pálya hasonló a `default` pályához, cserébe 2-2 példányt indítunk el minden ágenstípusból, így érdemes lehet arra is odafigyelni, hogy a robotok ne szorítsák egymást sarokba.

8 Beadás

A versenyfeladat megoldását kérjük, hogy Teams-en keresztül nyújtsák be, az `MIGT2025` csoportban, a kapcsoló `Feladatra` (angolul `Assignment`). Technikai okokból kérnénk, hogy a teljes Java-kódot töltsék fel egy `zip` állományba összecsomagolva.

Azonban a verseny értékelésének alapját a `rescueagents` csomag állományai jelentik!

- `AMSService.java`
- `FlyingDroneControl.java`
- `MedicalRobotControl.java`
- `RescueRobotControl.java`
- `StaticSensorControl.java`

Szükség esetén ebben a csomagban újabb állományokat létrehozhatnak, viszont kérjük, hogy a kiadott környezet további állományait ne módosítsák, mert az a versenyből való kizárást vonja maga után!

Ha bármilyen kérdésük merül fel a verseny során, kérjük, jelezzék azt az `MIGT2025` csoportban. Sok sikert kívánunk!