Szakdolgozat

Miskolci Egyetem

Multiágens rendszerek koordinációs problémáinak vizsgálata

Készítette:

Mengyán Márton

Programtervező informatikus

Témavezető:

Piller Imre

Miskolc, 2022

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Intézeti Tanszék **Szám:**

Szakdolgozat Feladat

Mengyán Márton (BPWTW0) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: szimuláció, ágens

A szakdolgozat címe: Multiágens rendszerek koordinációs problémáinak vizsgálata

A feladat részletezése:

A multiágens rendszerek célja, hogy egy adott problémát egyidejűleg több ágens irányításával tudjanak megoldani.
A dolgozat azt vizsgálja, hogy ilyen esetekben az ágensek között milyen jellegű kommunikáció lehet, illetve hogy az adott ágensnek hogyan kell viselkednie a környezetből, és más ágensektől érkező információk függvényében.
Ehhez specifikálásra, megtervezésre és implementálásra kerül egy szimulációs környezet, amely valós időben követhetővé teszi az ágensek állapotát.
Az ágensek viselkedéséhez tartozó paraméterek optimalizálásra kerülnek annak érdekében, hogy az adott célt minél hatékonyabban (például gyorsabban vagy kevesebb művelet elvégzésével) el tudják érni.

Témavezető: Piller Imre (egyetemi tanársegéd)

A feladat kiadásának ideje: 2021. Szeptember 27.

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott **Mengyán Márton**; Neptun-kód: `BPWTW0` a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős Programtervező informatikus szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy *Multiágens rendszerek koordinációs problémáinak vizsgálata* című szakdolgozatom saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....

Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

A szakdolgozat egy szimulációs környezet létrehozásával fog kezdődni, amelyet felül nézetből vehetünk szemügyre. Körökre osztott, az ágenseket egy fix sorrendbe rakjuk, amely meghatározza, mikor jön melyik ágens. A szimulációs környezet véges számú különböző szobákból és folyósókból épül fel. Két szobát egy folyosó köt össze, amelyet a folyosó mindkét végén egy ajtó zár le.

A szobák és folyosók blokkokból épülnek fel. Ezeken a blokkokon helyezkedhetnek el a karakterek limitáltan, azaz egynél több karakter nem tartózkodhat rajta.

Az ágensek két nagyobb csoportra vannak osztva, amelyeket *frakció*-knak nevezünk. Minden *frakciónak* van egy teljesítendő célja, amely teljesítésénél véget ér a szimuláció. Az ágensek miközben keresik a célukat, figyelembe veszik az útközben észrevett tárgyakat, kulcsokat, ajtókat és ellenséges ágenseket és ezek alapján tervezik meg következő lépésüket.

Ha az ágensek *Inventory* mérete nem telített, akkor az észrevett tárgyakat megközelítik és felveszik.

Az *Inventory*-ba bekerült hordható tárgyakat, ha nincs semmilyen más nagyobb prioritású elintézni valójuk, akkor megvizsgálják annak statisztikáit és összehasonlítják a vele azonos típusú *typeName* változójú általuk hordott tárgy statisztikáival, ezáltal meghatározva, hogy jobb-e vagy sem.

Mivel az *Inventory* véges ezért valamilyen módon kezelniük kell a saját *Inventory* helyeiket, hogy ne teljen meg értelmetlenül.

Ha van ellenséges ágens vagy tárgy az észlelési zónájukban, akkor azokat megközelítik. Több ágens észlevételénél a legkevesebb életerővel rendelkezőt fogja támadni.

Ezen kívül megvalósításra kerül a legfontosabb része a szimulációnak, amely a *map* herusztikus felfedezése.

A program Java programozási nyelven [1] készül, a *Java Swing* csomag [4] használatával.

Az alkalmazásban használt képeket én készítettem egy $16x16$-os vásznon a Pixelart weboldalon [2].

2. fejezet

A szimulációs környezet elemei és funkciói

2.1. Több ágenses szimulációs rendszerek

Szimulációs környezetekre különféle problémák megoldása során szükség van. Ezeknek egy speciális esetei az ágens alapú szimulátorok. A dolgozat azon belül egy szűkebb területtel, a több ágens alapú szimulációkkal (*Multi Agent Based Simulation*) témakörével foglalkozik, ahhoz mutat be egy lehetséges környezetet.

A szimulációs rendszerek között találkozhatunk például olyannal, amelyik a vészhelyzetben való menekülést igyekszik modellezni, a jellemző paramétereket becsülni, az elméleti eredményeket alátámasztani [8].

Alkalmazási területeket találhatunk még a vezeték nélküli hálózatok vizsgálata kapcsán is [5].

2.2. Általános jellemzők

A szimulációs környezet inspirációt vesz már elkészített videójátékokból, mint például a *Pixel Dungeon* [7] és *Darkest Dungeon* [6] nevű játékokból ismerhető szoba és folyosó kapcsolat. Ahol minden szobát egy folyosó köt össze egy másik tetszőleges szobával. A Darkest Dungeon-hoz képest a játékmenet rugalmasabb, mivel a szobák bármikor elhagyhatóak és megközelíthetőek, ha engedi azt a pálya jelenlegi belső felépítése. Azaz, ha nincs útban valami, ami megakadályozza.

A fent említett két játék mind körökre osztott, ahogyan a szimulációs környezetünk is az lesz.

A szimulációs környezetünk viszont egyedi lesz abból a szempontból, hogy az ágensek csoportokra vannak osztva és közös céllal rendelkezve próbálják legyőzni egymást és az ellenséges ágenseket.

Az alkalmazás a *Pela* fantázia nevet kapta.

2.3. Az ágens szimulációs felület elemei

Az ágensek és környezetük szimulációjához, a rajtuk elvégzendő mérésekhez célszerű összerakni egy grafikus felületet. A következő szakaszokban ezeknek a két fő elemét, a menüt és az UI-t mutatja be a dolgozat.

2.1. ábra. UI and Menu képernyőtervek

2.3.1. Menü

A szimulációs környezethez tartozik egy menü ablak. Ennek elemei az alábbiak.

Szimuláció indítása előtt:

- Középen felül a *Main Menu* kiírását láhatjuk, alatta pedig 2 opcióból választhatunk (2.1. ábra):
  (1) *Start*, a kiválasztása esetén elindul a szimuláció,
  (2) *Quit*, a szimulációs ablak bezárása.
- A fenn említett két menüpont között a `W` (fel) és `S` (le) billentyűk lenyomásával navigálhatunk.
- Aláhúzva jelzi a felhasználónak, hogy mely menüpont van éppen kiválasztva.
- A különböző ablakok és UI elemek egy egyedi fontot használnak [3].

A szimuláció megállítása esetén:

- A szimulációban a `P` billentyű lenyomásával képes a felhasználó megállítani a szimuláció menetét.

- A megállítást a bal felső sarokban kiírt fehér *Paused* kiírás jelzi a felhasználó számára (2.1. ábra).
- Ilyenkor az ágensek nem kapnak lehetőséget a lépésre, de a felhasználó képes a kamera mozgatására a nyilak segítségével.
- A felhasználó szintén képes az ágensek *Statistics* ablakát megnyitni/bezárni a `C` billentyű megnyomásával, és előre/hátra lapozni az ágensek között a `Q` (balra) és `E` (jobbra) billentyűk megnyomásával, ha több van a szimulációban, mint 1.
- Ugyanígy képes megnyitni/bezárni az adott ágenshez tartozó *Inventory* ablakot az `I` billentyű megnyomásával, ha látszódik az ágens statisztikája ablak.
- A `P` billentyű újonnan megnyomásával ott folytatódik a szimuláció, ahol abbamaradt (2.1. ábra).

2.4. UI

A UI (*User Interface*) foglalja magába az ágens szimulátor fő elemeit. A következőkben ennek az elemei, azok funkciói kerülnek felsorolásra.

A szimuláció figyelésére szolgáló UI ablakok, amelyek a szimuláció kezdete után a képernyőn jelennek meg, a következők:

- *Statistics*: A `C` billentyű megnyomására megjelenik az első számú ágens Statistics ablaka (2.1. ábra).
- *Inventory*: Az `I` billentyű megnyomására megjelenik az az adott ágens *Inventory* oldala (2.1. ábra).
- *Health Point*: Az ágens életerejét jeleníti meg.

2.4.1. Statistics

- Egy adott ágens adatának a kiolvasásának az eredményeit megjelenítő UI ablak.
- A szimulációban megállítástól függetlenül a `C` billentyű megnyomására jelenik meg és tűnik el.
- Megnyitásakor mindig a még szimulációban létező ágensek közül az első számú ágens adait fogja visszaadni.
- A vizsgálandó ágens elpusztulásakor a még létező ágensek közül az új első számú ágens adatait fogja visszaadni.
- Ezen az ablakon a következő ágens adatokat írja ki:
  - *Sorszám*: Az ágens sorszámát adja vissza.
  - *HP* (életerő): Az ágens aktuális életerejét adja vissza.
  - *Erő* (str) Az ágens aktuális str értékét adja vissza.
  - *Kitartás* (vit): Az ágens aktuális vit értékét adja vissza.

  - *Kitérés* (eva): Az ágens aktuális eva értékét adja vissza.
  - *Pontosság* (acc): Az ágens aktuális acc értékét adja vissza.

  Ezek a számok változhatnak az alapján, hogy milyen felszerelést hord az adott ágens.

- Új tárgy felvételénél, ha úgy ítéli, hogy jobb, mint az általa hordott azonos típusú tárgy, akkor lecseréli, és ennek megfelelően frissülnek a statisztikái.

2.4.2. Inventory

- Nem nyitható meg, ha nincs még megnyitva egy tetszőleges ágens *Statistics* ablaka (2.1. ábra).
- Megnyitásakor annak az ágensnek az *Inventory*-ja lesz látható, amelyiknek a *Statistics* oldala nyitva van.
- A *Statistics* oldalon az előre/hátra lépés esetén az *Inventory* az újonnan szemügyre vett ágens *Inventory*-ját fogja megjeleníteni a felhasználó számára.
- Az *Inventory*-ban a következő típusú tárgyak találhatóak meg:
  - *Armor* (páncél): Páncélként hordható tárgy, 4 alapstatisztikát tartalmaz.
  - *Helmet* (sisak): Sisakként hordható tárgy, 4 alapstatisztikát tartalmaz.
  - *Weapon* (fegyver): Fegyverként hordható tárgy, 4 alapstatisztikát tartalmaz.
  - *Key* (kulcs): Ajtó kinyitásához szükséges tárgy.
- Az *Inventory*-ban a `WASD` billentyűk lenyomásával választhatjuk ki, hogy mely *Inventory* helyet akarjuk megnézni.
- A fehér üres négyzet jelöli azt a helyett az *Inventory*-ban, amelyet éppen szemügyre vesz a felhasználó.
- Ha felszerelés típusú az adott tárgy, akkor a nevét és a 4 alapstatisztika értékét írja ki.
- Ha nem felszerelés típusú az adott tárgy, akkor pedig a nevét írja ki.

2.4.3. Health Point

- Minden ágens fölött az aktuális életcsíkja (*HP, Health Point*) jelenik meg automatikusan.

2.5. Szimulációs környezet specifikálása

A következőkben a szimulációs környezet modelljének, elemeinek a specifikálására kerül sor.

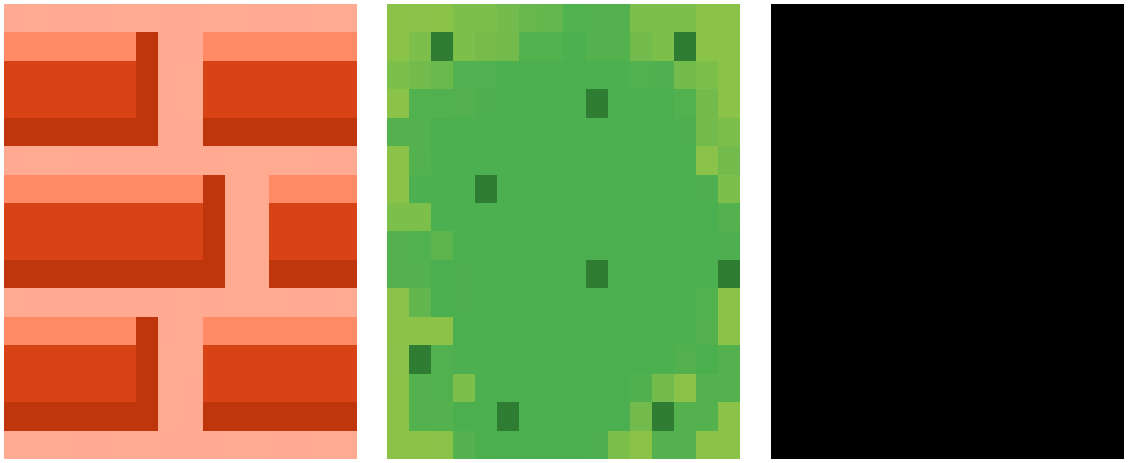
2.5.1. Map

A szimulációban az ágensek egy diszkrét négyzetrács felbontáson tudnak mozogni. Ennek a mérete rögzített, 50×50, és minden elemhez egy saját definiálású `block` típus tartozik.

Egy `block`-nak van típusa, amely egy egész számként reprezentálható. Ez a következő értékeket veheti fel:

- 0 típus: Üres, a *map*-on nem elérhető területeket jelöli.
- 1 típus: Fal, a bejárható területeket körbezáró fal, amelyen nem lehet átmenni.
- 2 típus: Füves terület, az ágensek által bejárható területek.

Kinézeteiket a 2.2 ábrán láthatjuk.



2.2. ábra. Blokkok

Az 50×50-es mátrixon minden 2-es típusú blokkra helyezhetünk el egy objektumot. Az objektumok típusai az alábbiak lehetnek.

- *Brick*: Egy mozgást korlátozó objektum. Nincs különösebb szerepe.
- *Chest*: *End screen*-t előidéző tárgy, amely a szimuláció végét jelenti.
- *Door*: Egy mozgást korlátozó objektum. Kulccsal kinyitható az ágensek által.

Az objektumok egy részhalmaza olyan, hogy az ágensek fel tudják azokat venni az *Inventory*-jukba, amelyek nem korlátozzák a mozgást. Ezek az alábbiak:

- *Key*,
- *Entity_Armor_01*,

- *Entity_ Weapon_ 01*,
- *Entity_ Helmet_ 01*,
- *Entity_ Armor_ 02*,
- *Entity_ Weapon_ 02*,
- *Entity_ Helmet_ 02*.

Kinézeteiket a 2.3. ábrán láthatjuk.

2.3. ábra. Item sets

Szimuláció kezdetekor megtörténik a *map* beolvasása. Ezt követően minden 2-es típusú blockra helyezhetünk el egy ágenst.

- *EntityFaction0*: 0-es számú csapatban lévő ágens.
- *EntityFaction1*: 1-es számú csapatban lévő ágens.

A szimulációs környezetben adott számú elemet helyezünk le. Konkrétan az alábbi számú és típusú elemek kerülnek rá:

- 18 szoba,

- 25 ajtó,
- 25 kulcs,
- 4 ágens,
- 10 felvehető tárgy,
- 2 láda,
- 12 oszlop.

2.5.2. Szobák és folyosók

A *map*-en szobák és folyosók alakíthatók ki, de nem tetszőleges módon. Az alábbi szabályrendszer írja le, hogy milyen feltételek szerint helyezheti le ezeket a program.

- A szobákat több szomszédos cellák alkotják.
- A szobákat folyosók kötik össze, amelyeket pontosan kettő cellával szomszédos cellák alkotnak.
- Minden szomszédos szobát egy ajtó választ el egymástól.
- A szobák celláin korlátozottan helyezkedhetnek el oszlopok, amely a cellát nem bejárható cellává alakítja át, és korlátozza a látást.
- Minden szoba tartalmaz legalább 1 kulcsot, amely elősegíti a *map* bejárhatóságát az ágensek által.

2.5.3. További map-re vonatkozó szabályok

A *map* szobák és folyosók összessége, ahol a szobák alakjánál törekszünk a négyzet alakú szobák elkerülésére. A folyosók általában rövidek és egy szobából akár több is nyílhat egy adott szobára. A *map* blokkokból épül fel. Blokkok típusát egy $[0, 2]$ intervallumon belüli számok jelölik. Ezek az adatok a szimuláció kezdetekor kerülnek beolvasásra a `world01.txt`-ből.

A szobákban és a folyosókban bejárható terület egy mátrixhoz hasonló, amely celláit blokkoknak nevezzük.

Minden blokkhoz tartoznak további adatok, úgy mint:

- X koordináta (Integer),
- Y koordináta (Integer).

Fontosabb kizáró feltételek:

- Ajtó csak szoba és a folyosó, vagy szoba és a szoba között létezzen.
- Folyosóból ne nyíljon ajtó folyosóra.
- Két különböző folyosó ne érjen össze.
- Nem lehet olyan cellára mozgást kizáró objektumokat letenni, amely lehetetlenné teszi a *map* egy tetszőleges pontjáról a *map* egy másik tetszőleges pontjára való eljutását.

2.6. Tárgyak

A tárgyakhoz a következő fontosabb adatok tartoznak:

- *name* (String): Az adott tárgyat bemutató név.
- *typeName* (String): A tárgyak tartalmaznak tárgy típus nevet, például: Armor, Helmet, Key... stb.
- *type* (Integer): A Key tárgy a 0 típusú, amíg minden hordható tárgy a 2 típus számot tartalmazza,
- *str* (Integer): a tárgy ereje (*strength*),
- *vit* (Integer): a tárgy kitartása (*vitality*),
- *eva* (Integer): a tárgy kitérés (*evasion*),
- *acc* (Integer): a tárgy pontosság (*accuracy*).

Az ágens által hordható tárgyaknak van valós *str*, *vit*, *eva* és *acc* értéke.

2.7. Tárgy nevek

Ágens által használható felszerelési tárgy, amely statisztikát ad, amelyek hozzáadódnak, illetve kivonódnak az ágens jelenlegi statisztikáiból. Ezek értékeit a 2.1. táblázatban láthatjuk. A tárgyak kinézetét a szimulációban pedig 2.3. ábrán láthatjuk, ahol a felső sor a 01 tárgyakat, a második sor a 02 tárgyakat foglalja össze.

2.1. táblázat. Tárgyak statisztikái

name	*str*		*vit*		*eva*		*acc*	
	min	max	min	max	min	max	min	max
Entity_Helmet_01	1	2	1	2	1	2	1	2
Entity_Helmet_02	0	3	0	3	0	3	0	3
Entity_Armor_01	1	2	1	2	1	2	1	2
Entity_Armor_02	0	3	0	3	0	3	0	3
Entity_Weapon_01	1	2	1	2	1	2	1	2
Entity_Weapon_02	0	3	0	3	0	3	0	3

2.7.1. Kulcs

Celláról szerezhetőek meg. Szerepük az ajtók kinyitása az ágens által. Kulcsot tartalmazó ágens, elpusztulásakor a legutolsó helyén hagyja a kulcsot.

2.8. Ágensek

A következő szakaszokban az ágensek tulajdonságai és a hozzájuk kapcsolódó számítási módok kerülnek részletezésre.

2.8.1. Statisztika

Az ágensek 4 alap statisztikával rendelkeznek. Ez a 4 statisztika a következők: Erő, Kitérés, Kitartás, Pontosság. Az ágensek kezdő statisztikája fix 1.

Minden ágens a következő tárgyakkal kezd a szimulációban:

- Entity_Helm_01,
- Entity_Armor_01,
- Entity_Weapon_01.

2.8.2. Sebzés számlálás

A támadásnak két végkimenetele lehet:

- *Találat*: kiszámolódik a sebzés mértéke, és levonásra kerül a megfelelő ágens életerejéből.
- *Eltévesztve*: nem történik meg a sebzés.

2.8.3. Statisztikák jelentősége

Statisztikák jelentősége két különböző *faction* változóval rendelkező ágens összetalálkozásánál látszódik meg. Két fontosabb esetben játszik szerepet, az ágens statisztikája a fent említett találkozásnál.

Egy ágens egy másik ágensre mért csapásának sikeressége függ attól, hogy nagyobb-e a támadó ágens *acc*, pontosság értéke, mint a támadást elszenvedő fél *eva*, kitérés értéke. Ha nagyobb a támadó ágens *acc*, pontosság értéke, mint a támadott *eva*, kitérés értéke, akkor 100%-osan betalál a sebzése, azaz támadás betalálása következik be. Ha nem nagyobb, akkor 20% esélye van arra a támadott félnek, hogy ne kapjon semmilyen sebzést, azaz támadás eltévesztése következzen be.

Egy ágens egy másik ágensre mért csapásának a sebzése függ attól, hogy nagyobb-e a támadó ágens *str*, erő értéke, mint a támadást elszenvedő fél *vit*, kitartás értéke. Ha nagyobb a támadó ágens *str*, erő értéke, mint a támadott *vit*, kitartás értéke, akkor 20% esélye van arra a támadó félnek, hogy dupla sebzést mérjen be, ekkor a támadás nagysága nagy. Ha nem nagyobb, akkor fixen 1 sebzést okoz a támadása, ekkor a támadás nagysága normál.

2.8.4. Elsődleges tulajdonságai

Az ágens létrejöttekor kerül megadásra általunk.

- str (Integer): az ágens ereje (*strength*),
- vit (Integer): az ágens kitartás (*vitality*),

- eva (Integer): az ágens kitérés (*evasion*),
- acc (Integer): az ágens pontosság (*accuracy*),
- maxLife (Integer): az ágens maximális életereje
- faction: (Integer): Lehet 0,1. Fontos változó a combat létrejöttéhez.
- healTurn: (Integer): Változó, amely azt tárolja, hogy melyik kör végén lesz HP regeneráció.

2.8.5. Kinézetet befolyásoló tényező

- Ágens állása. Látszódik, hogy milyen irányba néz.
- Az ágens *faction* típusa.

2.8.6. Kommunikáció közöttük

Ágensek bemenete, információ szerzése:

- A hozzájuk szomszédos blokkokat vizsgálják, ha nem találnak semmit megvizsgálják az általuk érzékelhető blokkokat.
- A balra, jobbra, felfelé és lefelé irányban érzékelnek 2 blokknyi távolságra, ha nem gátolja meg valami az útjukat.

Ágensek belső memóriája a következőket tartalmazza:

- *Inventory* állapota.
- Karakter ablak állapota.
- Aktuális HP állapota.
- A blokkok *hashMap* értékei (hogy hányszor volt az adott blokkokon, segít felderíteni a teljes térképet).
- Észlelt objektumok helyzete, típusa.
- *Inventory*-ban tárolt tárgyak.
- Észlelt ágens értékei. (észlelt ágensek *faction* értéke)
- Észlelt ágens helyzete (X, Y koordináta).

2.8.7. Megfigyelési akcióik

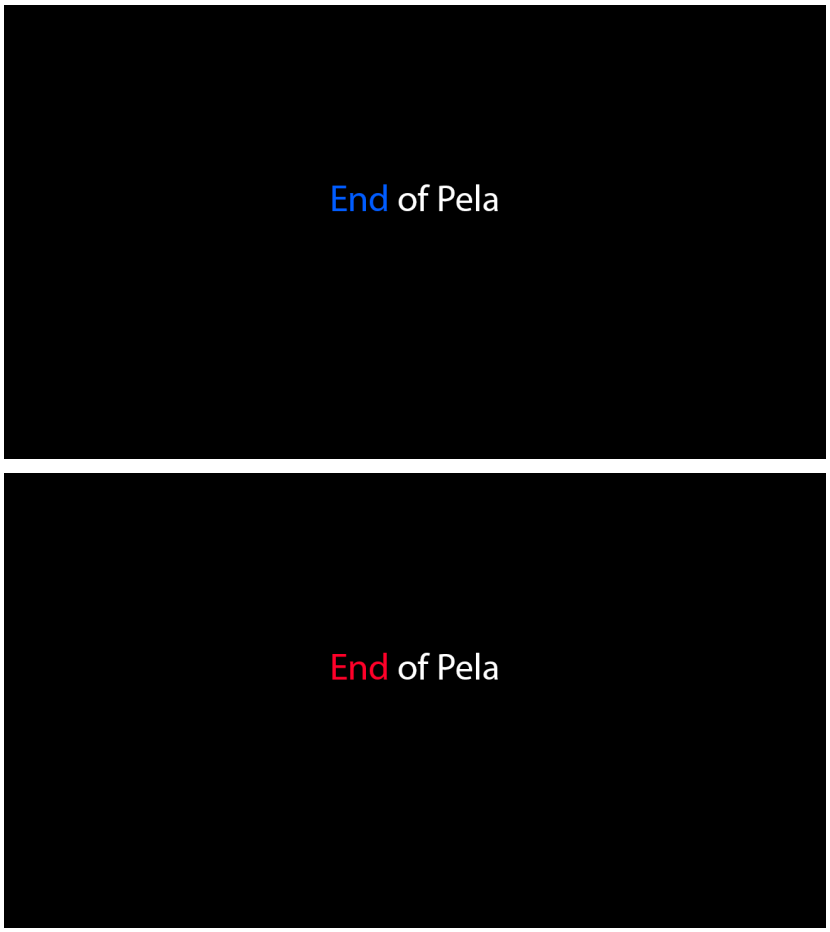
Információk feldolgozása:

- Az *Inventory* mérete és kihasználtsága. Ha elérte a maximum méretet nem képes az általa kívánt tárgyat felvenni.
- Karakter ablakban tárolt statisztikák számossága.
- Aktuális HP mennyiség. Adott Körönként fix HP visszatöltés történik meg (2.13. szakasz).
- Az adott körben lehetséges lépesek *hashMap* értékei vizsgálata, amelyek az adott blokk koordinátáit tárolják, és azt, hogy az adott ágens hányszor lépett rá az adott blokkra. (Szükséges azért, hogy idővel biztosan felfedezze az egész térképet.)
- Észlelt objektumok helyzete, típusa. Ezekből az információkból eldönti a prioritást, és ezáltal meghatározza, hogy mely irányba fog végül haladni.
- A nem használt tárgyak törlése, ha nem jobb a jelenleg hordottnál, ezáltal biztosítva, hogy ne teljen meg az *Inventory* mérete.
- Ágens értékeiből leszűrt információk számításba vétele. (ellenség-e)
- Támadható ágensek közül a legkisebb életerővel rendelkező ágens támadását kezdeményezi.

2.8.8. Ágens tevékenységeinek a köre

Az alábbiak bármely tevékenység végrehajtása, felhasználja az ágens körét.

- Lépés.
  - A lépés akkor lehetséges, hogyha az ágens közvetlen közelében, azaz vagy az X vagy az Y koordinátájával szomszédos cellán nem tartózkodik mozgást korlátozó ágens/térbeli objektum, lásd itt: Akadályok szakasz.
- Ajtó nyitás.
  - Szükséges az *Inventory*-ban egy *Key* nevű tárgy, amely a kulcs.
  - Ha az ágens közvetlen közelében létezik ajtó, a nyitás akkor lehetséges.
  - Nyitása után eltűnik az ajtó, és a kulcs az *Inventory*-ból.
- Láda nyitás.
  - Ha az ágens celláján létezik az ellenkező *faction chest*-je, akkor a nyitás lehetséges.
  - Ha nincs támadható ágens, akkor kinyitja.
  - Megjelenő képernyő a kinyitása esetén vagy piros vagy kék (2.4. ábra).
- Felvétel.



2.4. ábra. Záró képernyők (*End screens*)

  - Ha az *Inventory*-ban van hely, akkor lehetséges.
  - Ha az ágens cellájában létezik valamilyen tárgy, ekkor a tárgyat a felvétel akcióval eltünteti a celláról, és az *Inventory*-ba kerül.

- Törlés.

  - Ha az *Inventory*-ban van olyan tárgy, amely nem kulcs és nem hordott az ágens által, akkor lehetséges.
  - Az ágens *Inventory*-jából kitörli a tárgyat.

- Felszerelés.

  - A felszerelés mindig lehetséges az ágens saját körében.
  - Megvizsgálja, hogy az utoljára felvett tárgy statisztikái jobbak-e, mint az ő általa hordott azonos típusú hordható tárgy.
  - Ha igen, felveszi azt, ezáltal megváltoztatva saját statisztikáit.

- Támadás.

  - A támadás akkor lehetséges, hogyha az ágens közvetlen közelében, azaz vagy az X vagy az Y koordinátájával szomszédos cellán tartózkodik egy ellenséges ágens.
  - Több ellenséges ágens esetén, a legkisebb életerővel rendelkező fogja támadni.

2.8.9. Ágens fő céljai

Az ágensnek alapvetően két célja van:

- megtalálni az ellenséges *chest*-et,
- elpusztítani az ellenséges ágenseket.

2.8.10. Ágens másodlagos céljai

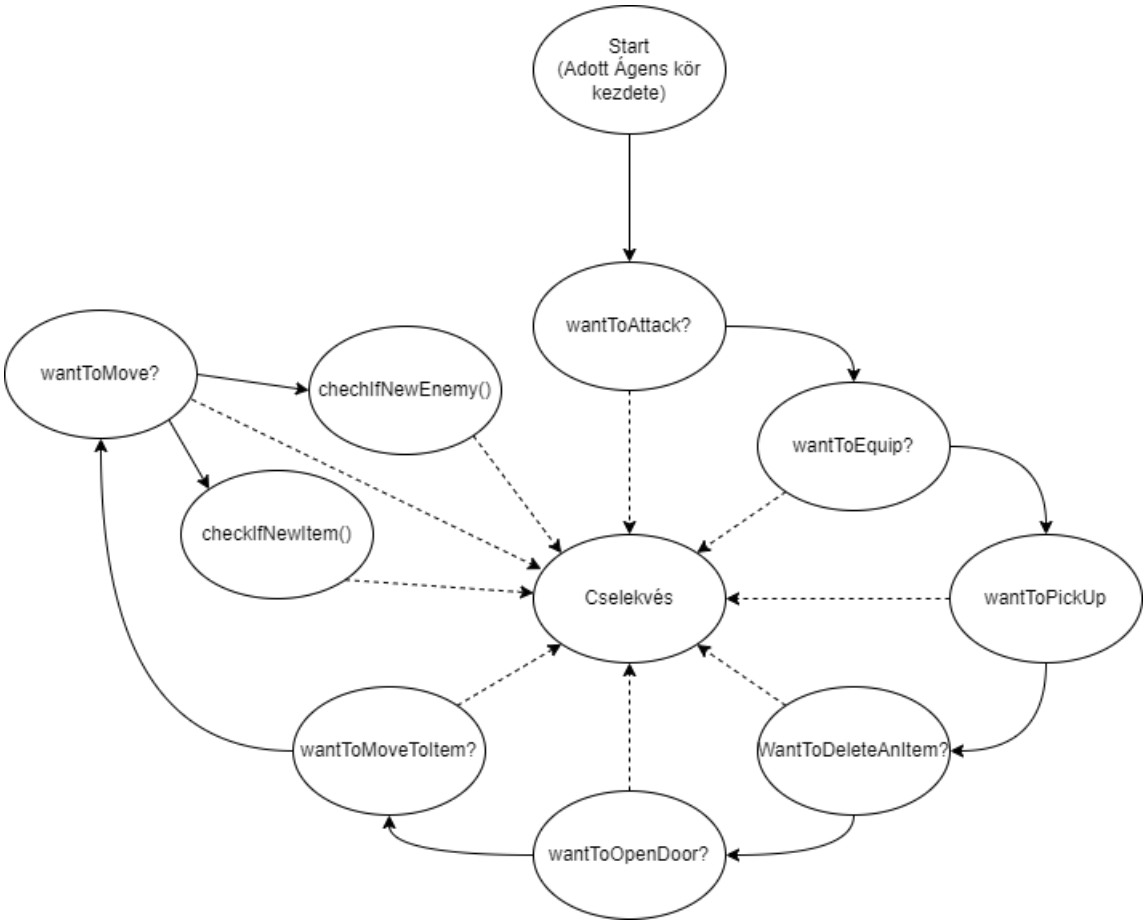
Az ágens szempontjából a következő tevékenységek végrehajtása szükségszerű, vagy előnyös lehet:

- Tárgy felvétel,
- Tárgy hordása adott esetekben,
- Ajtó kinyitás,
- *Inventory* menedzsment,
- Mindig a kevesebbszer bejárt blokkokat választani mozgáskor.

2.8.11. Ágens függvény

A 2.5. ábrán látható az ágens cselekvésének meghatározására szolgáló állapotgép gráfja, amely a szimuláció kezdetétől működik. A csomópontok fontossági sorrendben a következők.

- wantToAttack?
  - Ha közvetlen közelében van egy ellenséges ágens, mindig azt az akciót választja, ahol támadást mér be az ellenséges ágensnek.
- wantToEquip?
  - Ha a legutoljára általa felvett tárgy statisztikái több statisztikát adnak összességében, mint az általa hordott azonos típusú tárgy, akkor lecseréli.
- wantToPickUp?
  - Ha az ágenset tartalmazó blokk-on létezik valamilyen felvehető tárgy az ágens által, akkor felveszi.
- wantToDeleteAnItem?



2.5. ábra. *Agent Action Priority* állapotgép modell

  - Ha az ágens *Inventory*-ja tartalmaz olyan hordható tárgyat, amely nem jobb mint az általa hordott tárgy, akkor eltávolítja az*Inventory*-jából.

- wantToOpenDoor?

  - Ha van kulcs nála, és a közvetlen közelében van egy ajtó, akkor az ajtó kinyitását választja.

- wantToMoveToItem?

  - Ha van olyan közvetlen közeli blokk az ágens szomszédjában, amelyre lehetséges a lépés, és tartalmaz valamilyen felvehető tárgyat, akkor összeveti az ágens és a felvenni kívánt tárgy koordinátáit, és az alapján eldönti milyen irányba mozogjon az ágens.

- newEnemy?

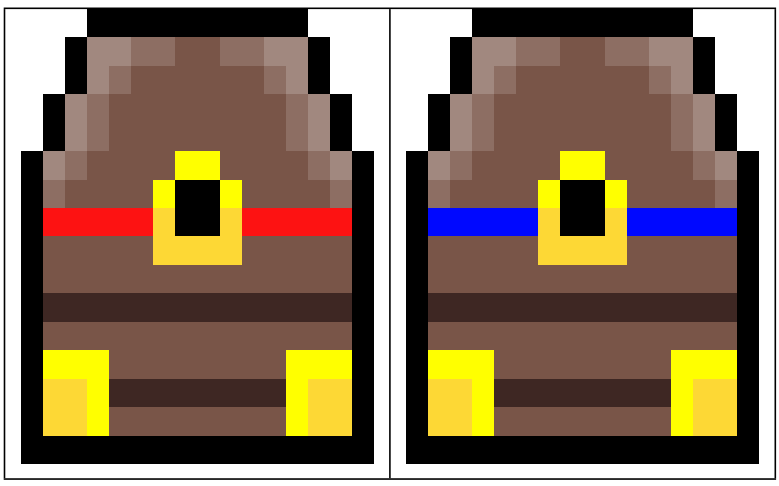
  - Ez az eset akkor lép fel, ha semmilyen más cselekvést nem akart elvégezni az adott körben az ágens és lát a *detectableBlocks*-on belül olyan blokkot, amely egy ellenséges ágenst tartalmaz. Ilyenkor a hozzá közeli blokkot választja ki következő lépésének.

- newItem?
  - Ez az eset akkor lép fel, ha semmilyen más cselekvést nem akart elvégezni az adott körben az ágens, és lát a *detectableBlocks*-on belül olyan blokkot, amely egy kívánt tárgyat tartalmaz, ilyenkor a hozzá közeli blokkot választja ki következő lépésének.
- wantToMove?

  Ez az eset akkor lép fel, ha semmilyen más cselekvést nem akart elvégezni az adott körben az ágens.
  - A lehetséges blokkok közül kiválasztja azt a blokkot a következő lépésének, amelyen még nem volt soha.
  - Ha több olyan lehetséges blokk van, amelyre léphet, akkor véletlenszerűen választ egyet a lehetőségek közül.
  - Ha nincs egy olyan blokk se, ahol ne lett volna már, akkor a legkevesebbszer bejárt blokkot fogja választani.
  - Ha több legkevesebbszer bejárt blokk létezik a lehetséges blokkok közül, akkor véletlenszerűen választ egyet a lehetőségek közül.

2.9. Cél Tárgy

Egy térkép teljesítéséhez szükséges tárgy, amely egy *chest*, láda. A kék *faction*-nek a piros ládát, a piros *faction*-nek a kék ládát kell elérnie és interakcióba lépni vele. (Ezek megjelenítését a 2.6. ábrán láthatjuk.) A térképen a ládák az ellenkező *faction* típusú ágensek kezdő szobájában helyezkedik el.



2.6. ábra. Ládák (*Chests*)

2.10. Támadás

A szimulációban 1-es *faction* változójú ágens és 2-es *faction* változójú ágens közötti életerő elvétel a szomszédos cellából lehetséges. A támadás lehet betalált vagy elvétett. A támadás nagysága lehet normális vagy nagy.

2.11. Felhasználói felület

A felhasználói felületen az elvégezhető műveletek különféle előfeltételekhez (prekondíciókhoz) vannak kötve. A műveleteknek van általános működése, illetve bekövetkezhet valamilyen alternatív és kivételes eset is.

A következő alpontokban a felhasználó által végrehajtható műveletek áttekintésére kerül sor, benne felsorolva azok prekondícióit, posztkondícióit, továbbá az eseteket.

2.11.1. Pause

- prekondíció: A szimulációs ablakban vagyunk.
- általános működés: Megnyomjuk a `P` gombot.
- alternatív esetek: Rossz tevékenység következik be.
- posztkondíció: Megjelent a *Paused* kiírás bal fent.
- kivételes esetek: Nem állt meg a játékmenet.

2.11.2. Inventory

- prekondíció: A szimulációs ablakban vagyunk, és meg van már nyitva egy tetszőleges *Statistics* ablak.
- általános működés: Megnyomjuk az `I` gombot.
- alternatív esetek: Rossz ablak nyílik meg.
- posztkondíció: Megnyílt az *Inventory* ablak.
- kivételes esetek: Nem nyílt meg az ablak.

2.11.3. Statistics

- prekondíció: A szimulációs ablakban vagyunk.
- általános működés: Megnyomjuk a `C` gombot.
- alternatív esetek: Rossz ablak nyílik meg.
- posztkondíció: Megnyílt a létező ágensek közül a legelső *Statistics* ablak.
- kivételes esetek: Nem nyílt meg az ablak.

2.12. Akadályok

A következőkben a térképen található mozgást kizáró objektumok kerülnek bemutatásra.

2.12.1. Brick

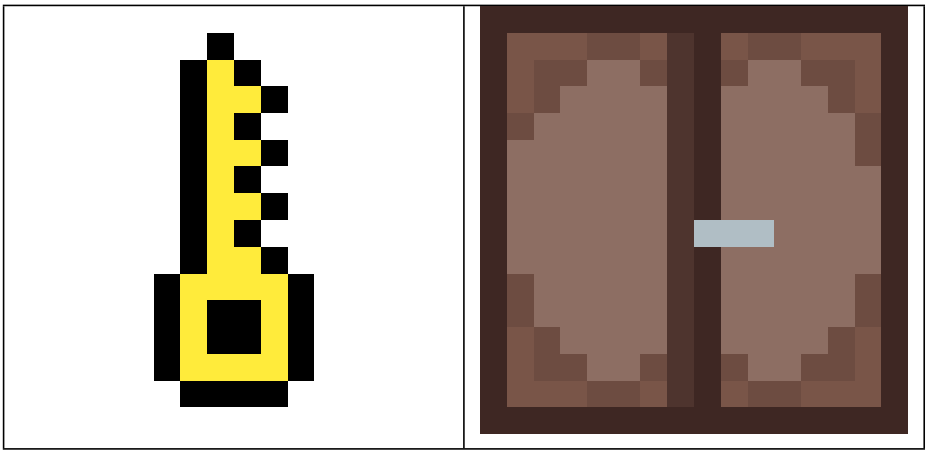
A *Brick* egy oszlop, amely a térkép beolvasása után kerül bizonyos cellákra. A kinézetéhez felhasznált kép a 2.7. ábrán látható.

Az oszlopoknak semmilyen különleges funkciója nincs azonkívül, hogy a térkép komplexitását kívánja növelni azzal, hogy mozgást és látást korlátozó szerepet lát el.

2.7. ábra. Brick

2.12.2. Zárt ajtó

Az ajtók a térkép beolvasása után kerülnek bizonyos cellákra. Az ajtók mozgást korlátozó szerepet töltenek be, ha nincs az ágensnél kulcs. Ha van, akkor a szomszédos blokkból kinyithatja az ajtókat. Ahogyan az *brick*-ek, a zárt ajtók is a térkép komplexitását kívánja növelni. A kulcsról és az ajtóról láthatunk egy képet a 2.8. ábrán.



2.8. ábra. Key and Door

2.13. Idő

A szimuláció egy diszkrét idejű rendszert modellez. Minden ágens által végrehajtott művelet során egy-egy időegység telik el.

Minden 5. kör után, 1 HP-t töltenek vissza az ágensek, ha kisebb az aktuális életerejük, mint a maximális. Ha egy ágens elvégez egy akciót, akkor az irányítás átkerül egy másik ágensre.

2.14. Kamera

A szimulációban modellezett virtuális teret valahogyan láthatóvá kell tenni. Ezt egy felülnézetes kamera formájában képzelhetjük el. A kamera mozgására néhány egyszerű szabály vonatkozik.

- A kamera blokkról blokkra képes haladni 3 koordináta meglépésével.
- A kamera képes mozogni, akkor is ha a játékmenet szünetel (*Paused*).

2.15. Irányítás

A korábbiakban már érintőlegesen szó esett a szimulációs környezetben használt billentyűkről, az azokkal kiváltott eseményekről. A következő felsorolásban áttekintésképpen láthatjuk ezeket egy helyen.

- Kamera mozgatása: `WASD`,
- Ágens *Statistics* ablak megnyitása/bezárása: `C`,
- Ágensek *Statistics* ablaka közötti lépegetés: `Q` (hátra), `E` (előre),
- Adott ágens *Statistics* oldalához tartozó *Inventory* megnyitása/bezárása: `I`.

3. fejezet

A keretrendszer tervei

A tervezés során elsősorban az alkalmazás által kezelt adatok meghatározása tünt szükségesnek. A következőkben ennek a megtervezéséről olvashatunk. Ezt követi majd az osztályokhoz tartozó fontosabb műveletek.

3.1. Az adatmodell előzetes tervei

A tervezés során először az adatrészekre, a megfelelő típusokra vonatkoztak a vizsgálatok. Ennek egy előzetes relációs sémában megadott modellje látható a 3.1. ábrán.

Ez még csak egy előzetes terv, melyhez képest a további, implementációhoz egyre közelebbi lépésekben bizonyos elemek változtak.

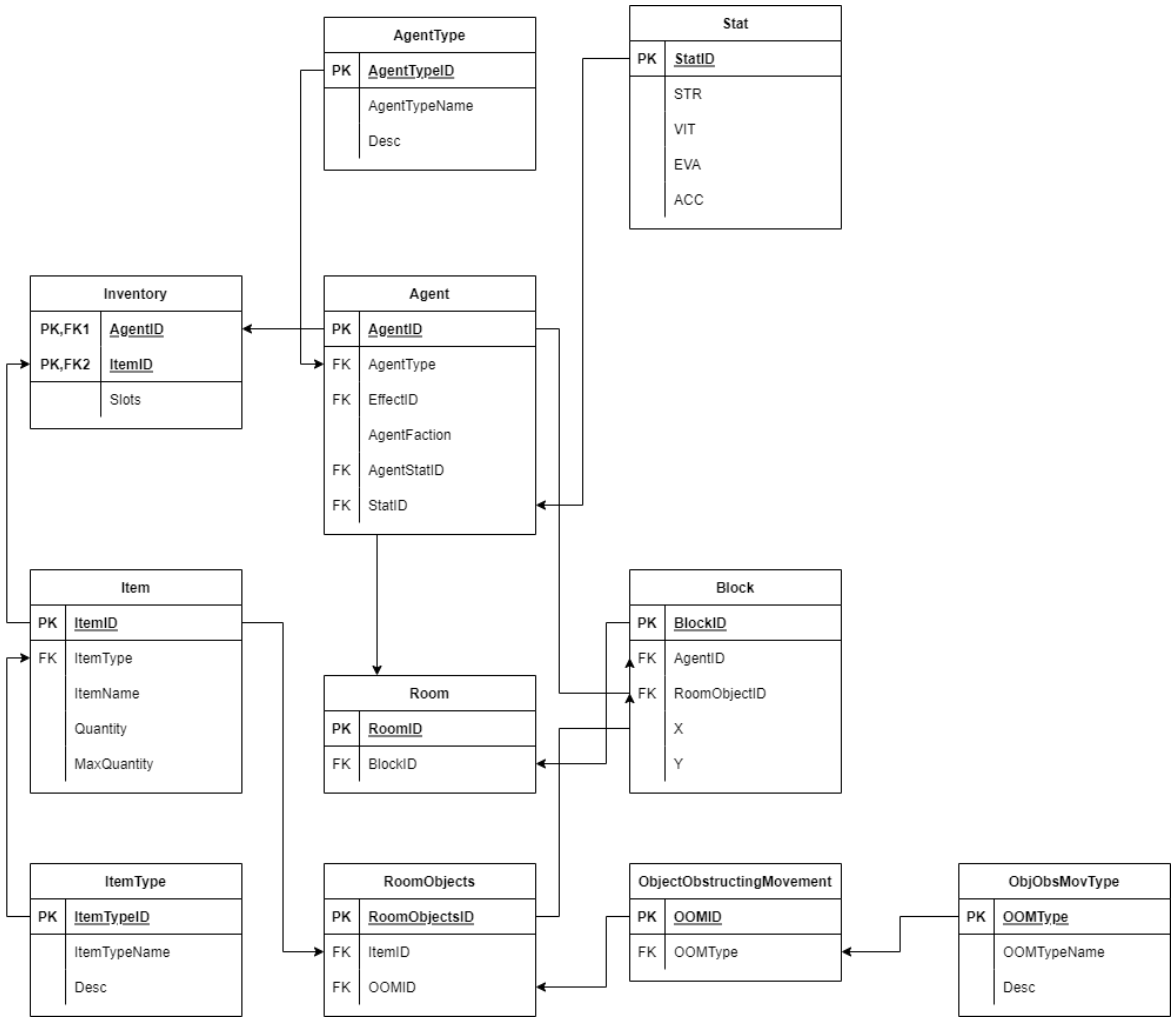
Először az ágensekhez tartozó statisztikák és az *Inventory* került bele az ágensek adataiba. Az inventory-hoz relációs séma szerint egy külön táblára volt szükség. Ez több szempontból is indokoltnak látszott. Egyrészt az *Inventory*-ban lévő elemek száma nem ismert előre (így nem volt lehetőség azokat közvetlenül, mezőként megadni), másrészt szükségesnek látszott a későbbiekre nézve felkészülni arra, hogy közvetlenül az elemekre is lehessen szűrést végezni, például hogy van-e olyan ágens, amelyik rendelkezik egy adott tárggyal.

A szobák adatainak kezelése nem ilyen formában valósult meg. A sémában szereplő változat még azt az állapotot tükrözi, amelyben minden szobához közvetlenül, egyesével történik a blokkok hozzárendelése. A modell kialakításánál a mátrixos reprezentáció jóval egyszerűbb, kézenfekvőbb megoldásnak tűnt.

A szobákban lévő objektumok végül úgy lettek megoldva, hogy a térképen vannak elhelyezve koordináta alapján, és a hozzátartozó típusok is magukba az objektumokba kerültek. Koncepcionálisan a séma ugyan megvalósítható lett volna ebben a formában is, de a térképre jellemző kialakítás előnyösebbnek bizonyult.

A relációs sémában kézenfekvőnek tűnt minden rekordhoz egyedi azonosítót rendelni. Az objektum orientált modell ehhez képest egyszerűsödött, abban már a tárolók (többségében listák) indexei elegendőek voltak.

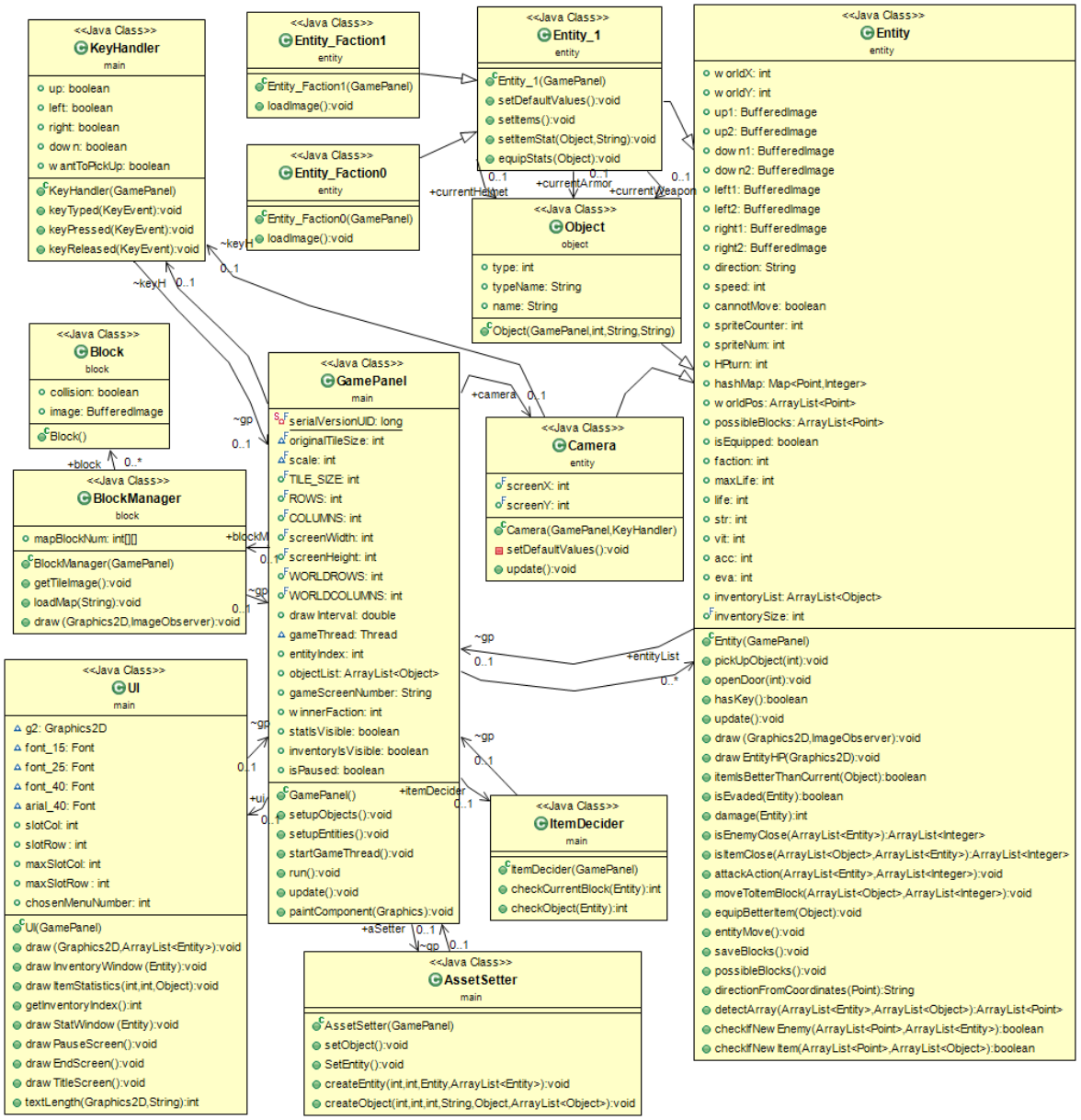
A szimulációs környezet tervezésénél nem volt cél az, hogy egy konkrét állapotot szerializálni és deszerializálni lehessen, továbbá az sem, hogy az állapot több számítógép között megoldható legyen. Mindkét eset egy reális további fejlesztési lehetőségként felmerülhet, amelyhez a relációs modell használata az adatok leírásához és tárolásához már szükséges lehet.



3.1. ábra. Legelső tervezéshez tartozó relációsadatmodel

3.2. Osztályok és kapcsolataik

A keretrendszer a széles körben elterjedt objektum orientált tervezési módszereknek megfelelően készült. Az alkalmazáshoz tartozó osztályok és azok kapcsolatai a 3.2. ábrán láthatók.



3.2. ábra. A keretrendszerhez tartozó osztályok

A következő szakaszok részletesen kitérnek ezek szerepére.

3.2.1. KeyHandler

A *KeyHandler* osztály felel a kamera mozgatásáért, a játékmenet elinditásáért/megállításért, UI elemek használatáért, mint például az *Inventory* és *Statistics* oldal megnyitása/bezárása.

Az adatai között külön logikai változóként szerepelnek az irányításért felelős kurzor billentyűk aktuális állapotai. Ez ahhoz szükséges, hogy a program egészére nézve aszinkron módon elérhető legyen azok állapota, mivel egyébként csak események formájában lennének jelen a rendszerben.

3.2.2. Block

A *Block* osztály a szimulátorban szereplő négyzet alakú blokkokat reprezentálja. Aránylag kevés adattal rendelkezik, mivel azok a rendszer többi eleméhez kerültek átcsoportosításra (például az entitásokhoz).

Az osztályhoz tartozik egy *BufferedImage* típusú adattag. Ez magának a blokknak a megjelenítéséhez szükséges kirajzolható raszteres képet tárolja.

3.2.3. BlockManager

Block típusok definiálása, tulajdonságok megadása és képek beolvasása a res mappából.

Egy `txt` kiterjesztésű fájlból beolvas egy 50×50-es mátrixot, amelyben minden elem 0, 1 vagy 2 egész értéket vehet fel. Az elemek szóközzel vannak elválasztva.

A *BlockManager* osztály felelős azért, hogy megrajzolja a térképet megvalósító blokkok kinézetet az alapján, hogy hol helyezkedik el a térképen és hogy milyen blokk kép tartozik a blokk típusához. Ehhez szüksége van arra, hogy be tudja tölteni a térképeket és a képeket a *loadMap* és a *getTileImage* metódusai segítségével.

3.2.4. GamePanel

Tartalmazza a képernyő felbontásának adatait, a blokkok méretét, az ágens listát és az objektumok listáját. (Utóbbiak neve nagy betűvel szerepel, mivel konstans értéknek tekinthetők az alkalmazás szempontjából, de lényegesnek tünt a kód érthetősége és a későbbi fejlesztések szempontjából nevesíteni őket.)

A *GamePanel* osztály segítségével inicializálódik a kezdő képernyő az alkalmazás elindítása után. Tartalmazza azokat a függvényeket, amelyek kezelik az ágens és objektum létrehozását. Ezek hívódnak meg a *main* metódusban. Továbbá ez felelős a *UI* elemek, az ágensek és tárgyak megrajzolásáért.

3.2.5. Camera

Kamera kezdő helyzetét, egy lépés távolságát (blokkokban) határozza meg. Kezeli továbbá a képernyő koordinátáit is.

3.2.6. AssetSetter

Az objektumok és ágensek elhelyezésért felelős. Két fontosabb függvényt tartalmaz.

- A *setEntity* metódus szolgál minden ágens létrehozásának a tárolására.
- A *setObject* metódus szolgál minden objektum létrehozásának a tárolására.

amelyek felelősek az ágensek és a térképen minden megtalálható objektum létrehozásáért a szimuláció elején.

3.2.7. UI

A *gameScreenNumber* változó által tárolt 3 fő képernyő megjelenítését tartalmazza: kezdő képernyő (*title*), játékmenet (*normal*) és végsőképernyő (*endscreen*). Itt történnek a képernyőre való szöveg kiírások is. *Inventory* és *Statistics* ablak megjelenítését is tartalmazza.

3.2.8. Entity

A következő fontosabb ágens adatokat tárolja:

- X és Y koordináta: a térképen lévő helyzetét jelentő értékek
- *HPturn*: Azt az érték tárolja, hogy hány kör múlva lesz életerő regenerálódás.
- Alapstatisztikák: Életerő, erő, kitartás, kitérés, pontosság.
- *InventorySize*: Az *Inventory*-ban maximálisan elhelyezhető objektumok száma.
- Irány: Meghatározza, hogy melyik irányba néző képet kell betöltenie.
- Gyorsaság: Lépésenként mennyi utat tesz meg, ez egy rögzített érték, amely egy blokknyi távolság.

Itt történik az ágensek :

- életcsíkjának a megrajzolása,
- megrajzolása a képernyőre,
- által legutoljára felvett hordható tárgy statisztika vizsgálata,
- tárgy felvétele,
- támadása, kulcs használata és mozgása,
- találatának eldöntése és sebzés méretének a kiértékelődése,
- és objektumok észlelése,
- *hashMap* értékeinek frissítése.

3.2.9. ItemDecider

Két fontosabb függvényt tartalmaz, amely egy-egy indexet ad vissza az Entity osztálynak, amelyek által kiértékelődik az, hogy az ágensek akarnak-e felvenni tárgyat a saját blokkjukról vagy kinyitni egy ajtót egy nekik szomszédos blokkról. Az egyik függvény azt vizsgálja, hogy van-e valamilyen objektum az ágens koordinátáján, a másik függvény azt vizsgálja, hogy van-e az ágens koordinátája alatti, feletti, jobb oldali vagy baloldali szomszédján valamilyen objektum.

3.2.10. Object

Az *objectek* osztálya, *object* típusokat tartalmaz. Az objektumok típusát, típus nevét és nevét tárolja. Ezek a változók fontosak a tárgyak vizsgálatánál ágens szempontból. Ezek származtatásával jönnek létre az objektumok a térképen.

3.2.11. Entity1

Ágensek egyetlen típusának alap értékeinek definiálására szolgál. Tartalmazza az ágensek kezdő felszereléseit, ezek a felszereléseknek a változói itt kapnak értéket. És itt kerülnek felszerelésre az ágensre statisztikailag és kerülnek be az *Inventory*-ba.

3.2.12. EntityFaction0 és EntityFaction1

A 0-ás és 1-es *faction*-höz tartozó ágensek képeit tárolja, amelyeket láthatunk a szimulációban.

4. fejezet

Megvalósítás

4.1. Felhasznált technológiák

4.1.1. Java

A Java általános célú, objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a 90-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle [1].

4.1.2. Java Swing

A Swing osztályok kiküszöbölik a Java legnagyobb gyengeségét, a viszonylag primitív felhasználói felület eszközkészletét. A Java Swing segít a Swing osztályok teljes előnyeinek kihasználásában, részletes leírást adva minden osztályról és felületről a kulcsfontosságú Swing csomagokban [4].

Más, hasonló szerkesztő és szimulációs rendszerek esetében is népszerű választás. Ez köszönhető egyrészt a Java nyelv elterjedtségének, másrészt, hogy napjainkban is az egyik legegyszerűbben átvihető *widget toolkit*-ról van szó.

4.2. Az ágensek vezérlése

Ebben a szakaszban a ágens heurisztikus mozgását és akcióit végrehajtó biztosító függvények bemutatása történik meg röviden. Ezek leírása úgy tünt praktikusnak, hogy ha először a hozzájuk tartozó forráskód részlet kerül megadásra, majd ezt követi a hozzá tartozó magyarázó szöveges rész.

4.3. wantToAttack

```
if (wantToAttack) {
    attackAction(gp.entityList, isEnemyCloseIndexList);
}
```

Boolean típusú változó, amely akkor igaz, ha közvetlen szomszédos blockokon létező ellenséges ágensek száma nagyobb, mint 0.

Igaz érték esetén meghívódik az attackAction() függvény, amelynek paramétere az ágensekből álló lista és az isEnemyCloseIndexList, amely a közeli blockon lévő ellenséges ágensek közül a legkevesebb HP-val rendelkező ágens indexét adja vissza az ágens listából.

Az attackAction() függvény beállítja az adott ágens irányát, arra amelyik irányba hajta végre a támadását és elvégzi a támadást.

Támadás előtt először megvizsgálja betalált-e a találat, ha betalált, akkor meghívódik a damaga() függvény, amely kivonja az adott sebzést az adott ágenstől, amelyik elszenvedte a támadást.

4.4. wantToEquip

```
else if (wantToEquip) {
    equipBetterItem(
        inventoryList.get(inventoryList.size() - 1));
}
```

Boolean típusú változó, amely akkor igaz, ha az adott ágens inventoryjában az utoljára felvett tárgy hordható típusú és összességében több statisztikát ad, mint az ágens által jelenleg hordott azonos típusú tárgy.

Az equipBetterItem() függvénynek egy paramétere van, amely egy Integer szám, ami az adott ágens inventoryjának utolsó használt slotja, amely a legutoljára felvett tárgy. Meghívásakor leszereli az eddig hordott tárgyat, és felszereli az újonnan felvett tárgyat. Statisztikák ablakon nyomon lehet követni ezt a változást.

4.5. wantToPickUp

```
else if (wantToPickUp) {
    pickUpObject(curObjIndex);
}
```

Boolean típusú változó, amely akkor igaz, ha létezik valamilyen tárgy az ágens jelenlegi blockján.

Ha igaz értéket ad vissza, akkor meghívódik a pickUpObject() függvény, amelynek paramétere az adott blockon lévő tárgy indexe a tárgyak listájában. Meghívásakor az adott tárgy bekerül az ágens első nem használt inventory slotjába.

4.6. wantToDeleteAnItem

```
else if (wantToDeleteAnItem) {
    System.out.println("Deleted item: "
        + inventoryList.get(deleteIndex).name);
    inventoryList.remove(deleteIndex);
}
```

Boolean típusú változó, amely akkor igaz, ha létezik valamilyen nem viselt tárgy az ágens inventoryjában.

Igaz érték esetén megtörténik a deletedIndex által tárolt elsőnek talált nem használt tárgy indexének a törlése az inventoryjából, amely nem az ágens által hordott és nem kulcs.

4.7. wantToOpenDoor

```
else if (wantToOpenDoor) {
    openDoor(doorIndex);
}
```

Boolean típusú változó, amely akkor igaz, ha létezik ajtó az ágens szomszédos blockjában és van az ágens inventoryjában legalább 1 kulcs.

Az openDoor() függvénynek egy paramétere van, amely az ajtó indexe a tárgy listában. Meghívásakor az ajtó objektum eltűnik, és eltávolítja a felhasznált kulcsot az ágens inventoryjából.

4.8. wantToMoveToItem

```
else if (wantToMoveToItem) {
    if (cannotMove == false) {
        moveToItemBlock(gp.objectList,
        isItemCloseIndexList);
        entityMove();
    } else System.out.println("Skipped Turn");
}
```

Boolean típusú változó, amely akkor igaz, ha a közeli blockokon lévő tárgyak listája nagyobb, mint 0 és az ágens inventoryja nincs tele.

Valódi ágens mozgás csak akkor jön létre, ha van olyan szomszédos block, amelyre képes lépni, ezt a CannotMove false értéke biztosítja.

Majd meghívódik a moveToItemBlock(), amelynek két paramétere van, az objektum lista és a szomszédos közelében lévő tárgyak indexének a listája. Ez a függvény beállítja az ágens direction értéket.

Majd meghívódik az entityMove() függvény, amely az ágens által választott irányba lép előre egyet és vagy felrakja a hashMap-re, vagy növeli az értékét.

4.9. wantToMove

```
else if (wantToMove) {
    if (cannotMove == false) {
        ArrayList<Point> detectableBlocks =
            detectArray(gp.entityList, gp.objectList);
        if (!checkIfNewEnemy(detectableBlocks,
```

```
            gp.entityList)) {
                if (!checkIfNewItem(detectableBlocks,
                gp.objectList)) {
                    possibleBlocks();
                }
            }
            entityMove();
        } else System.out.println("Skipped Turn");
}
```

Boolean típusú változó, amely mindig igaz, ha minden fentebb sorolt boolean változú hamis, ebben a pontban összefoglalt akció fog megtörténni.

Valódi ágens mozgás csak akkor jön létre, ha van olyan szomszédos block, amelyre képes lépni, ezt a CannotMove false értéke biztosítja.

A pontokat tároló listában a detectArray() függvény minden olyan x,y párost átad, amely az ágenstől vízszintes vagy függőlegesen 2 blocknyira van és tartalmaz valamilyen kívánt tárgyat. A detectArray két paramétere az ágensek listája és az objektum lista.

A checkIfNewEnemy() függvény paramétere a pontokat tároló lista és az ágensek listája. Ha talál valamilyen ellenséges ágenst a vizsgálandó pontokon, akkor azt az irányt fogja beállítani az ágensnek, amely az ellenséges ágens felé néz.

Utána meghívódik az entityMove() függvény, amely a kiválasztott irányba lép egyet és vagy felrakja a hashMap-re, vagy növeli az értékét.

Ha a checkIfNewEnemy() hamis értéket ad vissza, akkor megvizsgálja a checkIfNewItem() függvényt, amelynek két paramétere van, a vizsgálandó pontokat tároló lista és az objektumok listája.

Ha talál valamilyen számára érdekes tárgyat a vizsgálandó pontokon, akkor azt az irányt fogja beállítani az ágensnek, amely a kívánt tárgy felé néz.

Ha a checkIfNewItem() függvény is hamis értéket ad vissza, akkor meghívódik a possibleBlocks() függvény, amely felel a map felfedezéséért.

A függvény megvizsgálja a lehetséges lépéseket, hogy van-e köztük olyan block, amelyen még egyszer sem járt az ágens. Ha több ilyen van, akkor véletlenszerűen választ egyet azok közül.

Ha már minden lehetséges blockon járt legalább egyszer, akkor a lehetséges lépések közül a legkevesebbszer bejárt blockot választja. Ha több ilyen block is van, amin ugyanannyiszor volt, akkor véletlenszerűen választ közülük egyet.

4.10. Java Swing GUI fontosabb részeinek bemutatása

JFrame: ez az alapja a Swing alkalmazásoknak.

Nehány fontosabb függvénye a JFrame-nek:

- void add (c komponens): Hozzáad egy komponenst.
- void setSize (int szélesség, int magasság): Egy komponens szélességének és magasságának a beállítása.
- void setVisible (logikai érték a): Ha *a* igaz, akkor a komponens megjelenik a képernyőn.

- void setResizable(logikai érték a): Ha *a* igaz, akkor az ablak nem újraméretezhető.
- void pack(): úgy méretezi a keretet, hogy annak minden tartalma elérje vagy meghaladja a kívánt méretet.
- void setDefaultCloseOperation(): a következő paraméterekkel a következő tevékenységek mennek végbe bezáráskor:
  1. JFrame.EXIT_ON_CLOSE: Bezárja az alkalmazást.
  2. JFrame.HIDE_ON_CLOSE: Keret elrejtése, de az alkalmazás tovább fut.
  3. JFrame.DISPOSE_ON_CLOSE: Keretobjektum eldobása, de az alkalmazás tovább fut.
  4. JFrame.DO_NOTHING_ON_CLOSE: Nem csinál semmit.

5. fejezet

Tesztelés

5.1. Az alkalmazás elindítása

Az alkalmazás elindításakor megjelenik a *Main Menu* kiírás, ezen az oldalon, közép lent lévő ként opció közül választhatunk, a fel és lefele nyilak segítségével, a kiválasztott menü pont alatt egy aláhúzás fog megjelenni, amely a felhasználó számára jelzi, hogy melyik menü pont a jelenleg kiválasztott.

A számunkra megfelelő menüpont kiválasztása utána az `Enter` lenyomásával véglegesíthetjük döntésünket. Ekkor a menüpontnak megfelelő utasítás fog végrehajtódni.

Ha a *Start* menüpontot választottuk, akkor elhagyjuk a főmenüt és elkezdődik a szimuláció. Ha a *Quit* menüpontot választottuk, akkor az alkalmazás bezáródik.

5.2. Szimuláció közben használható funkciók

5.2.1. Pause

A szimuláció közben a felhasználónak lehetősége van megállítani a szimulációt, hogyha megszeretne valami vizsgálni vagy csak megszeretné állítani ideiglenesen a szimulációt. Ezt a `P` gomb megnyomásával teheti meg.

Az alkalmazás a szimuláció megállását, a bal felső sarokban kiírt fehér *Paused* felirattal jelzi a felhasználó számára. Ha a szimuláció jelenleg meg van állítva, akkor a `P` gomb még egyszer megnyomásával újra elindíthatjuk azt.

5.2.2. Statistics ablak

A szimuláció szüneteltetésétől függetlenül bármikor megnyithatjuk egy ágens *Statistics* oldalát, ahol mindig az első ágens adait fogjuk látni először. Megnyitása és bezárása a `C` gomb lenyomásával lehetséges. Ez az ablak 6 darab sort tartalmaz, mindegyik az ágens egy adatát írja ki a felhasználó számára. Az ágensek között a `Q` és `E` gombok megnyomásával haladhatunk hátra és előre. Az alkalmazás ezek használatára az ágens ablak bal alsó és jobb alsó sarkában lévő *PRESS Q* és *PRESS E* kiírásokkal vonja fel a felhasználó figyelmét.

5.2.3. Inventory ablak

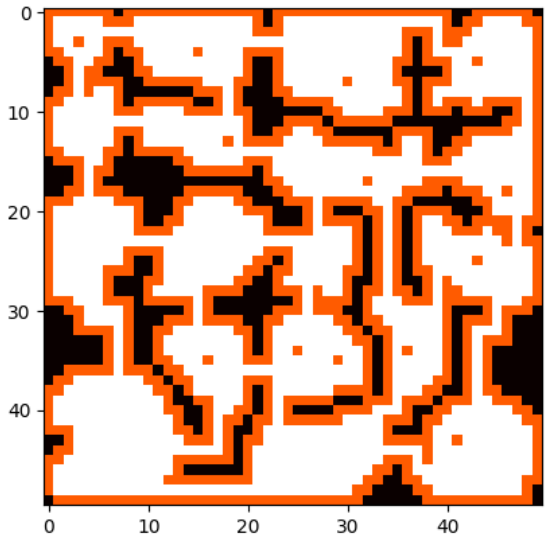
A *Statistics* ablakban megjelenített ágensnek sorszámától és a szimuláció szüneteltetésétől függetlenül bármikor megnyithatjuk a hozzá tartozó *Inventory* ablakot. Ezt az ablakot az ágens ablakkal ellentétben nem lehet léptetni, mindig a *Statistics* ablakban vizsgált ágens *Inventory* adatait fogja kiírni. Ezt az ablakot önmagában nem lehet megnyitni, csak akkor ha már a *Statistics* ablak meg van nyitva. Megnyitása és bezárása az `I` gomb lenyomásával lehetséges. Ez az ablak tartalmaz 8 négyzetet és egy rövid tárgy leírást, amely lehet 1 vagy 6 sorú, a tárgy típusától függően. Ha a tárgy hordható típusú, akkor első sorában a nevét, a további 4 sorában a statisztikáit láthatjuk, majd végül az utolsó sorában azt hogy fel van-e szerelve az ágens által. Ha a tárgy nem hordható (kulcs), akkor a tárgy leírásában csak az első sorát, a nevét fogjuk látni. Az ablak megnyitásakor mindig a bal felső négyzet van kijelölve, amelyet az alkalmazás úgy jelöl a felhasználó számára, hogy egy fehér négyzetet helyez rá. A négyzetekben tárgyak képeit láthatjuk, hogy ha a vizsgált ágens *Inventory* listája tartalmaz valamilyen tárgyat.

5.3. Szimuláció vége

A szimuláció befejeződésekor a képernyő közepén láthatunk egy kiírást, amelyben a szín attól függ, hogy mely csapat nyerte a szimulációt.

5.4. Eredmények

Ez a szakasz arról fog beszámolni, hogy milyen utat jártak be az ágensek két különböző szimulációban. Az általuk bejárható területeket a az 5.1. ábrán mutatom be. Minden fehér négyzet azt a területet ábrázolja, amelyet az ágens képes bejárni, amíg a narancssárga minden olyan területet mutat, amit semmilyen esetben nem tud bejárni.



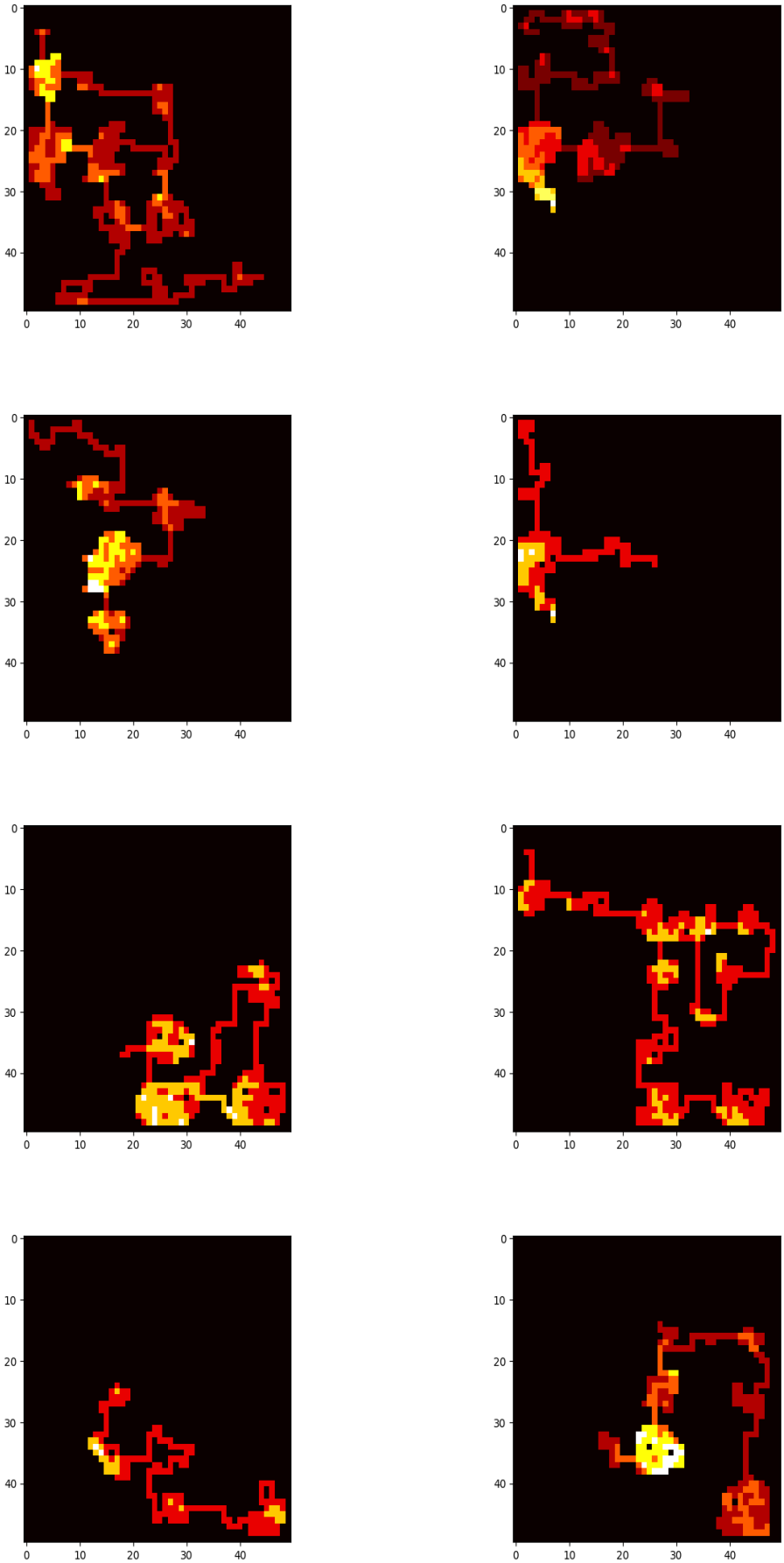
5.1. ábra. Map

Két szimulációban kapott eredményeket mutatom be a következő 5.2. ábrán. Minden oszlop egy különálló szimuláció eredményeit ábrázolja. Itt látható, hogy mely ágens mely szimulációban milyen utat járt be a mapon. Mivel az eredményeket bemutató ábrákat *heapmap*-eknek nevezzük, ezért egy adott blokk minél többször bejárt annál jobban fog kifehéredni.

Az ágensek mozgását befolyásolták általuk megtalált tárgyak, ellenséges ágensek és bezárt ajtók.

A képeken látszódnak szobák belsejében néhol egy vagy két fekete négyzet, ezek a szobákban lévő mozgást korlátozó oszlopok miatt vannak, amely megakadályozzák az ágenseket, hogy az adott négyzetekre lépjenek.

Mozgást korlátozó objektumok lehelyezése azért volt célszerű, hogy megakadályozzam néhány esetben azt, hogy az ágens néhány tárgyat könnyen észrevegyen.



5.2. ábra. Teszt eredmények

6. fejezet

Összefoglalás

A félévben a szakdolgozaton dolgozva sikerült felélénkíteni az egyetem alatt megtanult java ismereteim, és bővíteni azokat a java Swing-gel. Emellett nagy segítséget nyújtott a Szoftvertechológiák nevezetű tárgyon tanult ismeretek, amelyekben megtanultuk, hogy hogyan tervezzünk meg egy jövőben elkészülő alkalmazást. A félévben kisebb nehezségekbe ütköztem a tervezés és implementáció során, amelyek megoldása fejlesztette látásmódom, így ha legközelebb valamilyen hasonló feladatom lenne képes lennék hatékonyabban megoldani azt.

A tesztek során néhány ágens látványosan kevés utat tett meg a szimulációban, ennek oka az, ha összetalálkozott egy másik ellentétes ágenssel, akkor valamelyik ágens a számolások után elpusztult.

A szimuláció futási ideje nagyban függ attól, hogy mennyire segítjük az ágenseket azzal, hogy teszünk-e több kulcsot a pályára, mint amennyi ajtó van. Hiszen, ha ugyanannyi kulcs van a pályán, mint ajtó és feltételezzük, hogy olyan helyekre vannak eltéve, ahol nem lehetséges az, hogy úgy használják el minden kulcsukat, hogy ne férjenek hozzá további kulcsokhoz. Ebben az esetben annak az ágensnek, akinek elfogyott a potenciálisan megszerezhető kulcsok száma, akkor a másik ágens útvonalát kell megkeresni és követnie azt a tovább haladáshoz.

Illetve lehet növelni a tesztek komplexitását azzal is, hogy az ágensek másodlagos céljait többször vettjük vizsgálat alá. Azaz a szimulációba több tárgy észlelést, vizsgálatot és *combat*-ot, sebzés számolást teszünk be.

Irodalomjegyzék

[1] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[2] Bryan Ware. Pixel Art. `https://www.pixilart.com/`, 2022.

[3] codeman38. Yoster Island Font. `https://www.1001fonts.com/yoster-island-font.html/`, 2022.

[4] Robert Eckstein, Marc Loy, and Dave Wood. *Java Swing*. O'Reilly and Associates, Inc., USA, 1998.

[5] Muaz A Niazi and Amir Hussain. A novel agent-based simulation framework for sensing in complex adaptive environments. *IEEE Sensors Journal*, 11(2):404–412, 2010.

[6] Red Hook Studio. Darkest Dungeon. `https://www.darkestdungeon.com/`, 2022.

[7] watabou. Pixel Dungeon. `https://watabou.itch.io/pixel-dungeon`, 2022.

[8] Alexia Zoumpoulaki, Nikos Avradinis, and Spyros Vosinakis. A multi-agent simulation framework for emergency evacuations incorporating personality and emotions. In *Hellenic conference on artificial intelligence*, pages 423–428. Springer, 2010.

CD Használati útmutató

A CD az alábbi jegyzékeket tartalmazza:

- szakdolgozat: A szakdolgozat LaTeX forráskódját és PDF-t tartalmazza.
- Pela: Az alkalmazás jegyzékeit tartalmazza, a Jar file-t és néhány előző testet, amelyek a `test1.txt` és `test2.txt`.

A Pela jegyzék tartalma:

- `bin` jegyzék,
- `src` jegyzék: Java file-ok találhatóak meg az ebben talált jegyzékekben.
- `res` jegyzék: Használt képek, font-ok és txt az alkalmazásban.

A kód futtatása:

- a Jar file elinditásával lehetséges.
- A program a Java legújabb verziójában írodott, Java JDK 18.0.1 verzióval.