



Eötvös Loránd Tudományegyetem
Informatikai Kar
Programozási Nyelvek és
Fordítóprogramok Tanszék

Egység- és csapatmozgás algoritmusok

Pataki Norbert
Adjunktus

Szabó Zsolt
Programtervező Informatikus BSc

Budapest, 2014

Tartalomjegyzék

1. Bevezetés	3
2. Felhasználói dokumentáció	5
2.1. Feladat	5
2.2. A program indítása, rendszerkövetelmények	6
2.3. A program használata	6
2.3.1. Szerkesztőfelületek	6
2.3.2. Menü	6
2.3.3. Gráftervező	9
2.3.4. Szimulátor	15
2.3.5. Hibaüzenetek, tájékoztató üzenetek	19
3. Fejlesztői dokumentáció	20
3.1. Megoldási terv	20
3.1.1. A Menü	20
3.1.2. Generálási beállítások	20
3.1.3. Gráftervező felhasználói kezelőfelület	21
3.1.4. Szimulátor felhasználói kezelőfelület	21
3.1.5. Gráftervező megvalósítási terv	22
3.1.6. Szimulátor megvalósítási terv	22
3.2. Megvalósítás	24
3.2.1. Felhasználói esetek	24
3.2.2. Főbb osztályok kapcsolata	24
3.2.3. Főablak	25
3.2.4. Dialógus ablakok	26
3.2.5. Gombfelületek	27
3.2.6. Gráfmodell használt osztályai	28
3.2.7. Gráfmodell	30
3.2.8. Algoritmusmodell	32

3.2.9.	Gráftervező nézete által használt osztályok	35
3.2.10.	Gráftervező nézete	36
3.2.11.	Szimulátor nézete által használt osztályok	38
3.2.12.	Szimulátor nézete	40
3.3.	Tesztelés	42
3.3.1.	Főablak, dialógus ablakok, gombfelületek	42
3.3.2.	Gráfmodell	42
3.3.3.	Algoritmusmodell	43
3.3.4.	Gráftervező és szimulátor nézete	44
3.3.5.	Egyiségek mozgása	44
4.	Összefoglalás és végszó	45
	Irodalomjegyzék	46

1. fejezet

Bevezetés

A szakdolgozat a számítógépes játékokban használt egyszerűbb egység- és csapatmozgás algoritmusok és ahhoz kapcsolódóan, a gráfokon végrehajtható általánosan ismert illetve használt keresőalgoritmusok megvalósításáról és bemutatásáról szól. A dolgozat célja, hogy a tanult algoritmusok futása tetszőleges gráfokon bemutatható legyen, illetve stratégiai játékokban való használatuknak lehetőségeibe és nehézségeibe betekintést nyerjek, továbbá későbbi felhasználásra alkalmas és újabb algoritmusok megvalósítására és bemutatására lehetőséget adó programkód szülessen.

A számítógépes játékokban használt mesterséges intelligencia fontos eleme a játékban szereplő egységek útkeresése. FPS (First Person Shooter) játékokban a számítógép által irányított ellenfelek tudnak navigálni az akadályok között, és el tudnak jutni a játéktér elérhető területeire. RTS (Real Time Strategy) játékokban a gépi ellenfelek egységein kívül az emberi játékos által irányított egységek is oda mozognak, ahova a játékos küldi őket. Tulajdonképpen minden játékban, ahol valamilyen egységnak vagy karakternek el kell jutni a játéktér egy pontjából egy másikba, valamiféle útkeresést kell alkalmazni.

Erre egy lehetőség a dolgozatban is megvalósított gráfon való útkeresés. Ennek használatához a játék elérhető területeire le kell fektetnünk egy gráfot, amin majd az egységek a keresést végrehajthatják. Érdemes megjegyezni, hogy ha egy játékban egy gépi ellenfél néha indokolatlanul megakad vagy megáll, az nagyban ronthatja a játékélményt, sőt, ha például egy FPS játékban mindig a legrövidebb utat találja meg két pont között, az is valótlanak tűnhet. Ezek orvoslásának érdekében sokszor használnak olyan algoritmusokat az útkereséshez, amik nem feltétlenül találnak legrövidebb utat, hanem csak majdnem legrövidebbet, viszont sokkal gyorsabbak, mint az optimális megoldást szolgáltató algoritmusok.

A dolgozat keretei között elkészített programban az ismert, de lassú, így já-

tékokban nem használt Dijkstra útkereső algoritmusnak, a megfelelő heurisztikus függvényel használva jóval gyorsabb A* algoritmusnak és annak egy több szereplőre alkalmazott változatának működését szimulálhatjuk és követhetjük nyomon. A két A* változathoz különböző heurisztikus függvények használhatók. A többszereplős A* egyszerre keres utat minden egységnek a gráfon, és egy foglalási táblát és várakozó műveleteket beiktatva talál olyan utakat, melyeken ha végighaladnának az egységek, sosem kerülnének azonos időben azonos pontra. A többi algoritmust egyszerre csak egy egységre futtathatjuk. A gráf, amelyen az algoritmus futása történik, egy grafikus gráftervező felülettel generálható és szerkeszthető. A megvalósítás részletei és bővebb információ a [3.2](#) fejezetben található.

A keresések tényleges alkalmazásának bemutatására készült a szimulátorfelület, amelyen egy régi rácsos térkép-reprezentációt használó stratégiai játék játékteréhez hasonló területet alakíthatunk ki. Ezen területen automatikusan lefektetésre kerül egy a gráftervezőben is generálható gráf. A tervezőben az említett játéktípusban lehelyezhető épületeket és egyéb átváratlan objektumokat reprezentáló téglalap alakú akadályok és a játékbeli egységeket reprezentáló egy gráfpontot foglaló egységek lehetők. Az egységek egérrel irányíthatóak, képesek bizonyos speciális mozgásfolyamatok elvégzésére, mint egy másik egység követése, megelőzése, elkerülése vagy adott pontok közötti járőrözés. Céljuk megkereséséhez a korábban említett A* algoritmust használják, illetve ha több egység mozog egyszerre, akkor egy a korábban említett többszereplős kereséstől különböző, úgynevezett újratervező A* keresést használnak. A megvalósítás részletei és bővebb információ a [3.2](#) fejezetben található.

2. fejezet

Felhasználói dokumentáció

2.1. Feladat

A feladat egy gráfon futtatott keresőalgoritmusok futásának és valós idejű stratégiai játékokban való néhány alkalmazási lehetőségének bemutatására alkalmas program készítése.

Az algoritmusok vizuális bemutatásához egy gráftervező felületen lehessen gráfot rajzolni, lehessen generálni különböző méretű négy- és nyolc-kapcsolt rácsgrafokat és véletlen gráfokat, majd azokat lehessen módosítani. A gráfon adhassunk meg kezdő és végpontokat, amelyek az algoritmus kiinduló- és célpontját szolgáltatják. Kerüljön megvalósításra Dijkstra algoritmusá és az A* keresés különböző heurisztikus függvényekkel, valamint egy többszereplős útkeresési algoritmus, mellyel egyszerre több kezdő és végpont közt kereshetünk utakat. Az algoritmus vizualizációjának sebessége legyen változtatható.

Az A* algoritmus egy felhasználásának bemutatására készítsünk egy szimulátor felületet, mellyel egy valós idejű stratégiai játék játékterét, egységeit és az azokat mozgásukban korlátozó akadályokat reprezentáló terület generálható véletlenszerűen, majd szerkeszthető egységek és akadályok lehelyezésével és törlésével. Szerkesztés után indíthassuk a szimulációt, melynek során az egységek elküldhetők a térkép üres területeire, illetve olyan speciális mozgásformákat végezhetnek, mint egy másik egység követése, megelőzése, elkerülése vagy adott pontok közötti járőrözés. Egyéni útkeresésükhez a gráftervezőben bemutatható A* algoritmust használják.

Legyen lehetőség a megszerkesztett gráf és szimulációs térkép elmentésére és betöltésére, és legyen egy programon belül elérhető rövid segítség a program irányításához.

2.2. A program indítása, rendszerkövetelmények

A program indításához Linux (Ubuntu) operációs rendszeren, a CD-n található GraffKereso mappát másoljuk a /home/<felhasználónév> mappába. Ezután a benne runGrafKereso.sh script futtatásával a program elindul. Ha a program nem indul el, futtassuk a platforms mappában található fixdep scriptet, majd próbálkozzunk újra, illetve ellenőrizzük a runGrafKereso.sh script és a GrafKereso állomány hozzáférési adatait (futtatható-e).

A program jó használhatóságához legalább 1366 × 768-as képernyőfelbontás és futtatáskor legalább 300MB szabad memória szükséges.

2.3. A program használata

2.3.1. Szerkesztőfelületek

A program indításakor nem jelenik meg szerkesztőfelület. Azt a következő (2.3.2) alfejezetben leírtak szerint lehet létrehozni. A gráftervező felülete világoskék háttérű, a szimulátoré világoszöld. A szerkesztőfelületeken fel-le az egér görgőjével vagy az alkalmazásablak jobb oldalán található görgetősávval mozgathatjuk a látható területet, balra-jobbra mozgatáshoz pedig ALT+egérgörgő vagy az ablak alján találjató görgetősáv áll rendelkezésre.

2.3.2. Menü

A menüből az alábbiak szerint lehet új gráftervezőt vagy szimuláltort indítani.

A Menü melletti Help gomb alatt a Show help opcióra kattintva megjelenik egy az alábbinál valamivel tömörebb útmutató.

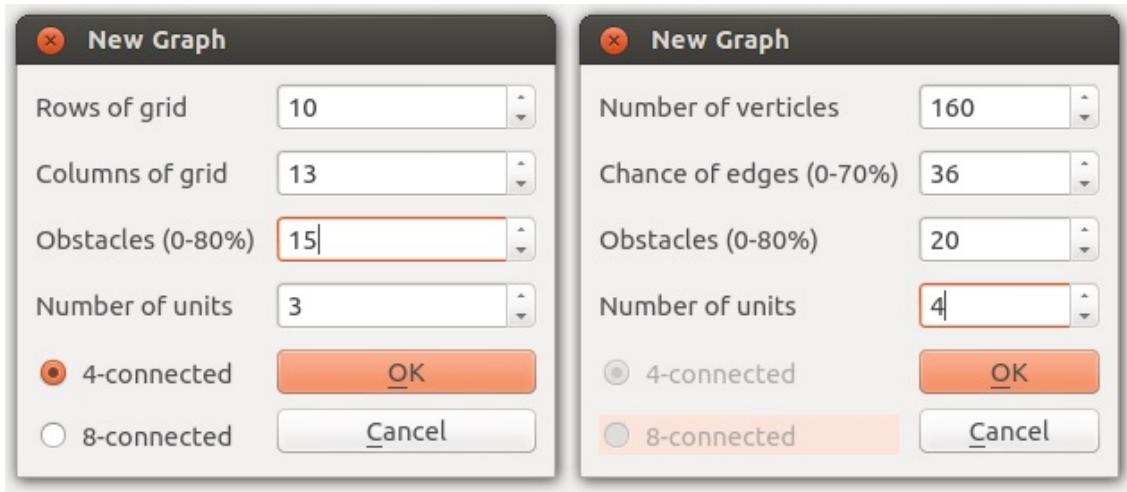
New Graph menüpont - új gráftervező nyitása

Draw: Üres szerkesztőfelület jelenik meg

Generate grid: Rácsgráfot generál az alábbi beállítható értékek szerint.

Rows of grid - rács sorainak számát adhatjuk meg ebben a számdobozban kézzel beírva, vagy az egér görgőjével beállítva. Értéke 3 és 70 között változtatható. A megrajzolt pontok mérete függ a beállított értéktől.

Columns of grid - rács oszlopainak számát adhatjuk meg ebben a számdobozban kézzel beírva, vagy az egér görgőjével beállítva. Értéke 3 és 70 között változtatható. A megrajzolt pontok mérete függ a beállított értéktől.



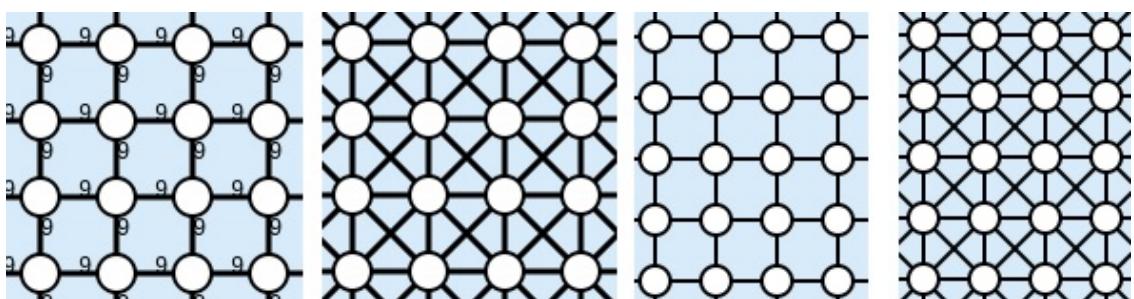
2.1. ábra. Új gráf dialógus ablakok, rácsgráfhoz (bal) és véletlen gráfhoz (jobb)

Obstacles - Akadályok aránya a generált csúcsok közül. 0 és 100% között állítható kézzel és egérgörgővel. A generált csúcsok a beállított eséllyel lesznek akadályok, azaz létező, de a keresőalgoritmus által figyelembe nem vett csúcsok.

Number of Units - Generált "egységek" (Start-cél párok) száma a gráfon. Értéke 0 és 30 között változtatható. minden generált start és cél különböző csúcsra kerül. Amennyiben nincs legalább kétszer annyi plusz egy csúcs, mint egység, akkor azok az egységek, amik nem férnek fel a gráfra nem generálódnak le.

4-connected - 4-kapcsolt rácsgráfot generál (csak sor és oszlopszomszédos csúcsok közt futnak élek)

8-connected - 8-kapcsolt rácsgráfot generál (átlós szomszékok közt is fut él)



2.2. ábra. Négy és nyolc-kapcsolt rácok közepes és nagy sor és oszlopszám mellett, megjelenített és elrejtett élsúlyokkal

Generate random: Véletlen gráfot generál az alábbi beállítható értékek szerint. (Gráfrészlet a 2.8 ábrán)

Number of verticles - Generált csúcsok száma. 10 és 2000 között állítható. A csúcsok véletlenszerűen lesznek felhelyezve a szerkesztőfelületre, méretük minden ugyanakkora.

Chance of edges - Élgenerálás valószínűsége minden lehetséges ére minden pont egy bizonyos környezetében. Hogy a gráf átláthatóbb maradjon és a keresés értelmes eredményt adhasson, ezért az így generált élek minden pontnak csak egy bizonyos környezetében levő más pontokhoz húzódhatnak. A környezet mérete a generált csúcsok számától függ.

Obstacles - Akadályok aránya a generált csúcsok közül. 0-100% közötti érték. Ugyanaz, mint rácsgráfban.

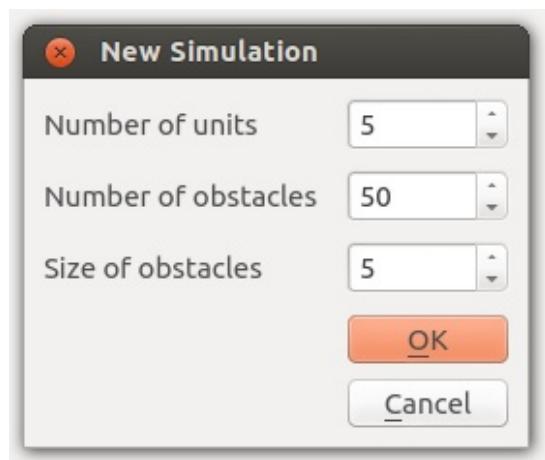
Number of Units - Generált "egységek" (Start-cél párok) száma. 0-30 közötti érték. Ugyanaz, mint rácsgráfban.

Megjegyzés: Rácsgráf és kevés pontú véletlen gráf esetén a gráf a teljes szerkesztőfelület bal felső területén generálódik, így lehetséges, hogy nem jelenik meg annak kezdetben látható részén.

New Simulation - új szimulátor nyitása

Empty: Üres szimulátor nyitása

Own settings: Saját beállításokkal véletlen generált szerkesztőt indít.



2.3. ábra. Új szimulátor dialógus ablak

Number of units - Generált egységek száma. 0-50 között állítható. Az egységek véletlenszerűen lesznek lehelyezve a 40×40 egyégyjni méretű területre.

Number of obstacles - Generált akadályok száma. 0-100 között állítható. Téglalap alakú akadályok kerülnek lehelyezésre véletlenszerű helyre, a következő pontban beállított maximális oldalhosszal.

Size of obstacles - Generált akadályok mérete. 1-10 közötti érték. A generált akadályok maximális oldalhossza a beállított érték+1 lesz, a minimális oldalhossz 1.

Save Graph/Simulation - gráf/szimulátor elmentése

Gráfnál a csúcsok, élek és egységek lesznek mentve, az algoritmus állása nem.

Szimulátoronál csak az egységek és akadályok lesznek mentve, az aktuális mozgások nem.

Megjegyzés: Az egységmozgás közben megállított szimulátor mentésének betöltése hibához vezethet, ezért szimulátort csak elindított szimuláció mellett is álló egységek esetén ajánlott menteni.

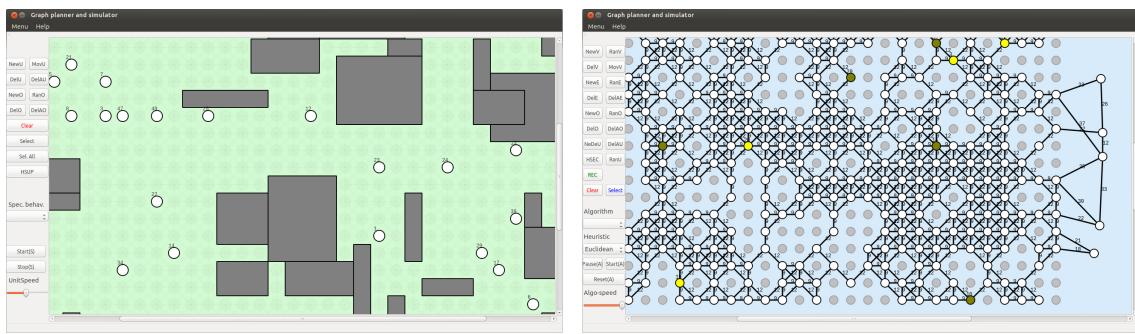
Load Graph/Simulation - gráf/szimulátor betöltése

Az elmentett állapot töltődik be. Ha a betöltendő fájl hibás, akkor arról hibaüzenet jelenik meg és nem töltődik be semmi.

Megjegyzés: A generálások és betöltések a beállított értékektől függően akár több másodpercig is tarthatnak (de fél percnél nem tovább).

Exit

Kilépés a programból.



2.4. ábra. Teljes alkalmazásablakok: szimulátor (bal) és gráftervező (jobb)

2.3.3. Gráftervező

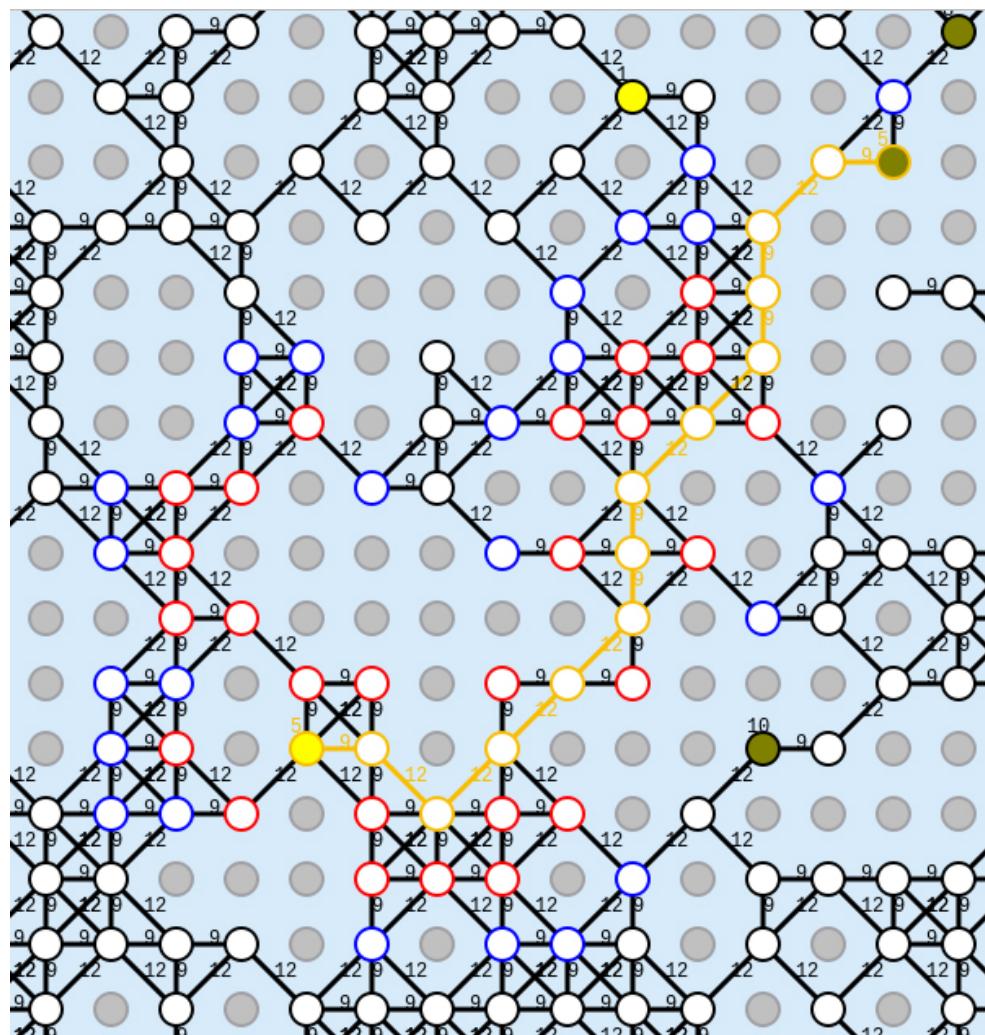
Használat, megjegyzések

A következők a megfelelő lejjebb ismertetett gomb megnyomása vagy beállítási érték megváltoztatása után értendők. Generálhatunk és szerkeszthetünk gráfot pontok letételével, élek behúzásával, akadályok és egységek lehelyezésével vagy elemek törlésével. A pontok mozgathatók, de ettől élek súlya nem változik (de az algoritmus heurisztikus értéke igen). Az élek súlya kézzel nem állítható, behúzáskor a tényleges hosszal arányos értéket kap. A HSEC gomb segítségével az élhosszak randomizálhatók, de csak egyszerre az összes. Akadály minden pontra lehelyezhető, ami nem egység kezdő vagy végpont. Egységet annak kezdő és végpontjának megadásával tehetünk le.

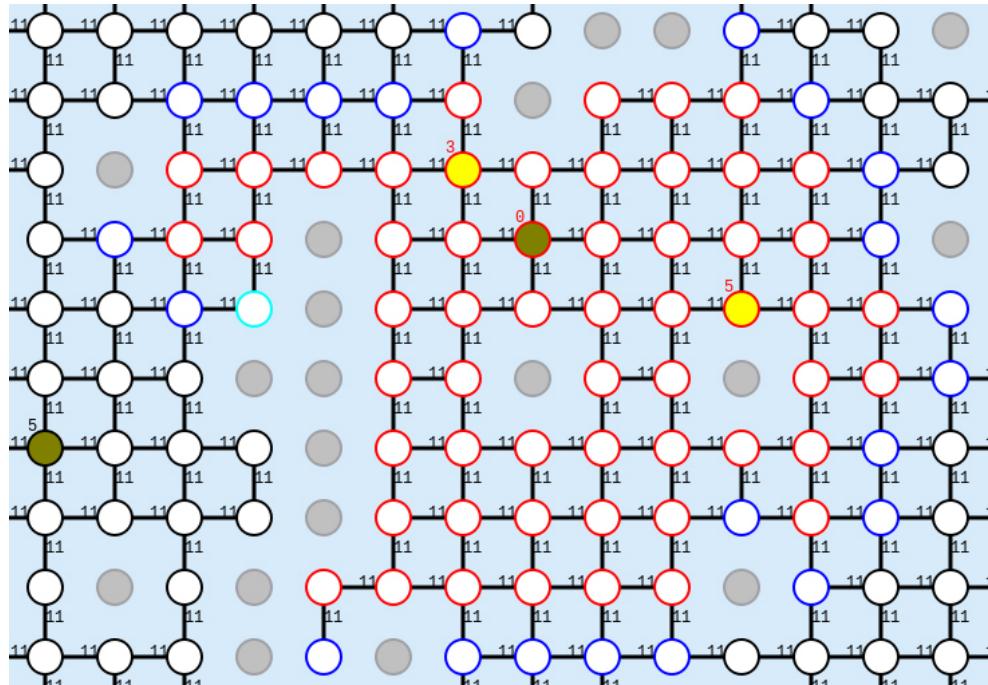
Ha van a gráfon egység akkor az Algoritmusválasztó-doboz segítségével megadhatjuk milyen algoritmust használjon a céljának megkereséséhez. A* (AStar) és

Többszereplős A* (AStarMAPF) esetén a heurisztikaválasztó-dobozbeli heurisztikával dolgozik az algoritmus. Az algoritmus-vizualizáció sebessége a csúszkával állítható, de a 0 érték "végtelen" sebességet jelent. A Többszereplős A* nagy gráfban sok egységgel még végtelen sebességen is futhat sokáig.

A Start(A) gomb megnyomásával elindíthatjuk az algoritmust, ami a beállított sebességgel fog futni (körülbelül 1-10 lépés/mp). Ha a sebesség 0 (végtelen), akkor rövid várakozás után, rögtön az algoritmus végállapotát, és végső üzenetét láthatjuk. Ha a sebesség nem 0 akkor végigkövethetjük, hogyan fut az algoritmus, milyen sorrendben ellenőrzi a csúcsokat. Az algoritmus szüneteltethető és kezdőhelyzetbe állítható. Bármely szerkesztőgomb megnyomása leállítja és alaphelyzetbe helyezi az algoritmust. A gráf a menü "Save Graph/Simulation" menüpontjának segítségével elmenthető.



2.5. ábra. Gráftervező egy A* algoritmus lefutása után



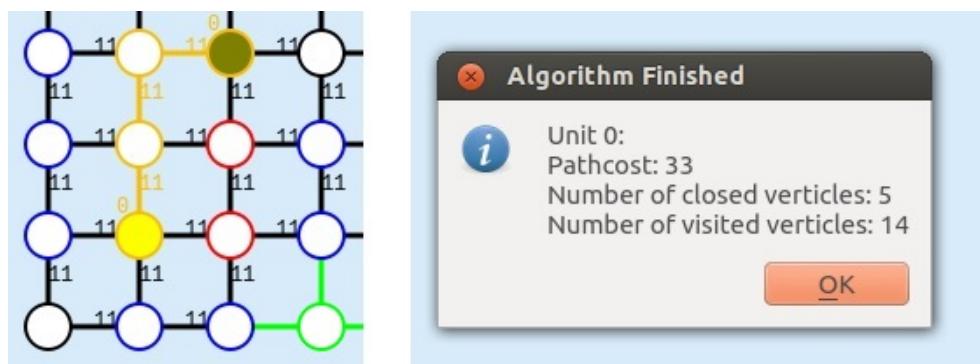
2.6. ábra. Gráftervező egy A* algoritmus lefutása közben (5-ös egységhez)

Grafikus elemek

Pontok

Fekete keretű fehérrel töltött kör - Kijelöletlen, szabad, algoritmus által még nemelfedezett pont.

Zöld keret - Az adott pont ki van jelölve. Akkor hordoz hasznos információt, ha elvégpontokat vagy egység kezdő- és végpontot jelölünk ki.



2.7. ábra. Algoritmus futásának eredménye gráfon, annak kiírása és pontszínezése, és kijelölt pont és élek színezése

Citromsárga töltés - Az adott pont egy egység kezdőpontja, az egység száma a ponttól balra felfele látható.

Sötét zöldes-sárga töltés - Az adott pont egy egység végpontja, az egység száma

a ponttól balra felfele látható.

Szürke keret, szürke töltés - Az adott pont egy akadály. Az akadályok létező pontok, éleik lehetnek, de nem látszanak, viszont kijelölhetők, tehát például eltörlés esetén a láthatatlan élek is törlődnek a kijelölés területén. A hozzájuk újonnan, kézzel behúzott élek sem jelennek meg, de létrejönnek.

Kék keret - Az adott pont a kereső algoritmus által már megtalált, de meg nem vizsgált pont.

Piros keret - Az adott pont a kereső algoritmus által már megtalált és megvizsgált pont.

Narancssárga keret - Az adott pont a kereső algoritmus által megtalált legrövidebb úton van.

Világoskék keret - Az adott pont a kereső algoritmus által éppen vizsgált pont.

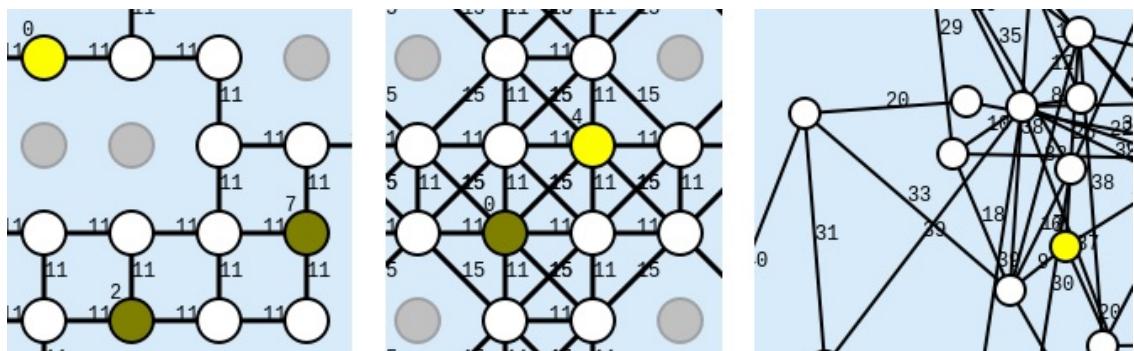
Élek

Fekete vonal - Kijelöletlen átvájható él.

Szám az él közepén - Az adott él súlya.

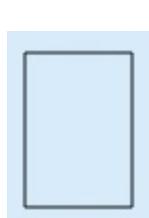
Zöld szín - Az adott él ki van jelölve.

Narancssárga szín - Az adott él a kereső algoritmus által megtalált legrövidebb úton van.



2.8. ábra. Négy-kapcsolt rács (bal), Nyolc-kapcsolt rács (közép), Véletlen gráf (jobb), és rajtuk egység kezdő- és végpontok, és akadályok

Kijelölő téglalap



Alaphelyzetben és a kijelölést lehetővé tevő műveletekkor a mozgatlan egér gombjának megnyomása után, nyomva tartás mellett mozgatott egér hatására jelenik meg. A téglalapon belüli elemek az egérgomb felengedése után kijelöltté válnak és megtörténik rajtuk a megfelelő akció a lejjebb írtak alapján, például törlődnek vagy akadállyá válnak.

Gombok és egyéb elemek a bal oldalon

Szerkesztőgombok

NewV - Új csúcs hozzáadása. A gomb megnyomása után, míg a gomb zöld, a többi csúcstól kellő távolságra a kék felületre kattintva tehető le új csúcs.

RanV - Véletlenszerű helyre tesz le új csúcsot egy üres területre.

DelV - Csúcs törlése. A gomb megnyomása után, míg a gomb zöld, egy csúcsra kattintva, az törlődik. Az egérgombot nyomva tartva és egy területet kijelölve, az adott területen minden csúcs törlődik.



MovV - Csúcs mozgatása. A gomb megnyomása után, míg a gomb zöld, egy csúcsot kijelölve (rákattintva), majd egy üres területre kattintva az adott csúcs áthelyezhető. Az éleinek hossza ettől nem változik.

NewE - Új él felvétele. A gomb megnyomása után, míg a gomb zöld, egymás után két különböző csúcsra kattintva azok között megjelenik egy irányítatlan él. Hurokél, többszörös él behúzására nincs lehetőség. Egy él behúzása után egy újabb él behúzásához ismét két kattintás szükséges.

RanE - Véletlenszerűen behúz egy eddig be nem húzott élt, a teljes gráfon. Tehát itt már nem csak a pontok egy környezetében húzhat be élt, mint véletlen gráf generáláskor.

DelE - Él törlése. A gomb megnyomása után, míg a gomb zöld, élekre kattintva, azok törlődnek, akár egyszerre több is. Az egérgombot nyomva tartva és egy területet kijelölve, az adott területen minden él törlődik, az akadályok rejtvé élei is.

DelAE - minden élet töröl a gráfból.

NewO - Új akadály felvétele. A gomb megnyomása után, míg a gomb zöld, egy létező csúcsra kattintva az akadályá válik. Élei nem törlődnek, csak rejtvé lesznek. Az akadályokhoz behúzott új élek sem jelennek meg, de létrejönnek. Az egérgombot nyomva tartva és egy területet kijelölve, az adott területen minden csúcs akadályá válik.

RanO - Véletlenszerűen akadályá tesz egy csúcsot ami nem akadály és nem egység kezdő- vagy végpont. *DelO* - Akadály törlése. A gomb megnyomása után, míg a gomb zöld, egy akadályra kattintva, az a csúcs nem lesz többé akadály, élei, ha vannak, újra megjelennek. Az egérgombot nyomva tartva és egy területet kijelölve, az adott területen minden akadály eltűnik.

DelAO - minden akadályt eltűntet.

NeDeU - Új egység hozzáadása/egység törlése. A gomb megnyomása után, míg a gomb zöld, egymás után két különböző csúcsra kattintva azok rendre egy új egység kezdő és végpontja lesznek. Ha a kijelölt két csúcs megegyezik, akkor nem jön létre új egység. Akadály is kiválasztható, ekkor az eltűnik. Ha valamelyik kiválasztott csúcs már egy egység kezdő- vagy végpontja, akkor az az egység törlődik.

DelAU - Töröl minden egységet.

RanU - Véletlenszerűen felhelyez egy egységet (azaz egy kezdő és egy végpontot), olyan helyre ahol még nincs egység kezdő- vagy végpont.

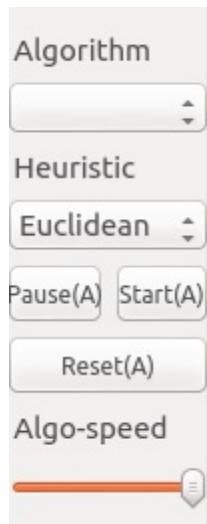
HSEC - Élsúlyok elrejtése/megjelenítése. Az élsúlyok az élek közepén jelennek meg.

REC - minden él súlyát egy 1-50 közti véletlen értékre állítja.

Clear - Töröl minden felületen.

Select - Kijelölő módba vált, például lenyomott gombok felengedéséhez használható, a korábban írt kijelölésekhez nem kell használni. Ilyen módban a kijelöléseknek az elemek zöldre színezésén kívül nincsen hatása.

Algoritmus beállítások



Felső választódoboz - Keresési algoritmus választása (Dijkstra, Dijkstra2, A*, többszereplős A*). A két Dijkstra ugyanazt az algoritmust futtatja.

Alsó választódoboz - A* és többszereplős A* algoritmusokhoz heurisztika választása. Az Euklideszi és Manhattan heuristikák széles körben ismertek. A harmadik és negyedik az aktuális pont és a cél pont közötti csúcsokat, illetve nem akadály csúcsokat számolja ki heurisztikus értékként.

Start(A) - Algoritmus indítása, Dijkstra, Dijkstra2 és A* esetén alapértelmezetten a legkisebb indexű egységre fut, de kattintással kijelölhető bármelyik egység. A többszereplős A* minden egységre keres, így annak nem lehet egységet kijelölni.

Pause(A) - Algoritmus szüneteltetése. Újabb Startra a futás folytatódik.

Reset(A) - Algoritmus kezdőhelyzetbe állítása. A következő Start az algoritmust előlről kezdi.

Csúszka - Az algoritmus sebességének állítása. A maximális követhető sebesség

körülbelül 10 lépés/másodperc a legkisebb 1 lépés/mp, de a csúszka 0-ra állításával "végtelen" sebességre gyorsul, ekkor a Start megnyomása és rövid várakozás után rögtön az algoritmus végállapota jelenik meg a gráfon.

Megjegyzés: minden gomb és egyéb elem fölé helyezve a kurzort, rövid időn megjelenik egy rövid magyarázat az adott elemhez.

2.3.4. Szimulátor

Használat, megjegyzések

A következők a megfelelő lejjebb ismertetett gomb megnyomása vagy beállítási érték megváltoztatása után értendők. Generálhatunk és szerkeszthetünk térképet akadályok és egységek lehelyezésével vagy törlésével. Az egységek áthelyezhetők bárhol, ahol üres terület van (azaz nincsen másik egység vagy akadály). Akadályok nem helyezhetők át.

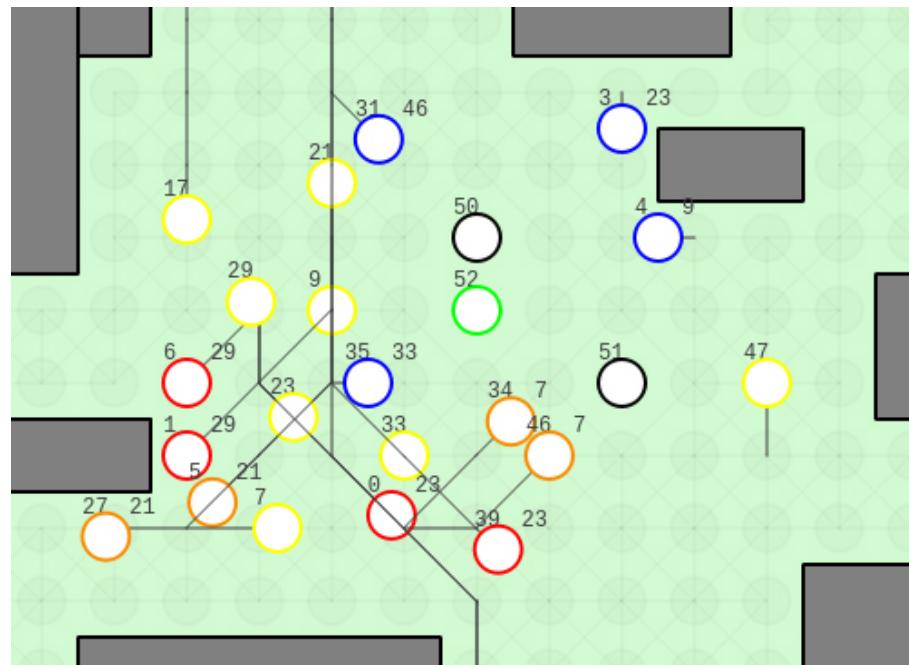
Az egységek kijelölhetők egyenként kattintással, vagy egyszerre, nyomva tartott egérgombbal egy területet kijelölve, illetve a Sel. All gombbal. Jobb egérgombbal a kijelölt egységeket irányíthatók a "Spec. behav." választódoboz aktuális beállításának megfelelően. Ha a doboz üres, az egységek egyszerűen a jobb egérkattintás helyénél található halvány pontokra mozognak. Ha ott nincs elég hely minden egységnak, előfordulhat, hogy bizonyos egységek nem indulnak el. Ha a dobozban a "Follow" szöveg látható, akkor jobb egérgombbal egy egységre kattintva a kijelölt egységek követni fogják azt. Ha a dobozban a "Intercept" szöveg látható, akkor jobb egérgombbal egy egységre kattintva a kijelölt egységek megpróbálnak közvetlen az elé vágni. Ha a dobozban a "Evade" szöveg látható, akkor jobb egérgombbal egy egységre kattintva a kijelölt egységek el fogják kerülni azt. Egy egység nem követheti, előzheti meg és kerülheti el önmagát. Ezen beállítások mellett jobb egérgombbal üres területre kattintva a kijelölt egységek abbahagyják aktuális beállított tevékenységüket és a kattintás helyére mennek (mintha a doboz üres lenne).

Ha a dobozban a "Patrol" szöveg látható, akkor jobb egérgombbal üres területre kattintva az adott hely hozzáadódik a kijelölt egységek járőrzési pontjaihoz és az egység ezen pontok között járőrözni kezd. Ha még nem járőrözik az egység, akkor az aktuális helye lesz az első járőrzési pont. Ha több egység van kijelölve, akkor minden egyik azt a pontot adja a járőrzési pontjaihoz, ahova üres doboz esetén ugyanilyen kattintás után ment volna. Mozgó egység az aktuális végcéljától indítja a járőrzést.

Egy egység áthelyezése töri a rá vonatkozó szabályokat (követés, járőrzés, stb.). Követés és megelőzés esetén előfordulhat, hogy az egységek nem mozognak céljuk felé, pedig még távol vannak attól. Ekkor a célegység mozgása újraindítja a folya-

matot. A speciális tevékenységek során megeshet, hogy az egységek abbahagyják az adott mozgásformát. Ez akkor történhet meg, ha célpontjukhoz egyáltalán nem találnak utat.

A Start(S) gombbal elindítható a szimuláció. Az egységeknek álló szimuláció mellett is kiadhatók a fenti utasítások, ekkor azok indítás után válnak érvényessé. A "Unitspeed" csúszkával állítható a kijelölt egységek sebessége. Bármely szerkesztőgomb megnyomása leállítja a szimulációt.



2.9. ábra. Szimulátor működés közben, megjelenített útvonalakkal

Grafikus elemek

Szerkesztőfelület

Zöld háttéren halvány, a gráftervezőben látott hasonló pontok és élek láthatóak. Ezek nem mozgathatók. Ez a területre lefektetett gráf, amin az egységek utat keresnek. Az egységek ezen pontok között mozognak. Új egység lehelyezésekor és egységek áthelyezésekor ezen pontok egyikére kerülnek az egységek. Akadályok is e pontok fölé helyezhetők, és az új akadály azon pontok fölé jön létre, melyek középpontját tartalmazza a kijelölt terület.

Egységek

Fekete keretű fehérrel töltött kör - Kijelöletlen, speciális mozgást nem végző egység. Az egység száma az tőle balra fent látható.

Zöld keret - Az adott egység ki van jelölve.

Kék keret - Az adott egység elkerül (evade) egy másik egységet, az elkerült egység száma az elkerülő fölött jobb oldalon látható.

Piros keret - Az adott egység megelőz (intercept) egy másik egységet, a megelőzött egység száma a megelőző fölött jobb oldalon látható.

Narancssárga keret - Az adott egység követ (follow) egy másik egységet, a követett egység száma a követő fölött jobb oldalon látható.

Citromsárga keret - Az egység járőröz (patrol), azaz adott pontok közötti mozgást ismétel.

Kijelölő téglalap

Alaphelyzetben és a kijelölést lehetővé tevő műveletekkor az álló egér gombjának megnyomása után, nyomva tartás mellett mozgatott egér hatására jelenik meg. A téglalapon belüli elemek az egérgomb felengedése után kijelölt elemek lesznek, és megtörténik rajtuk a megfelelő akció a lejjebb írtak alapján.

Akadályok

Sötétsürke keretű, szürke töltésű téglalapok. Az egységek mozgás közben kikerülik őket.

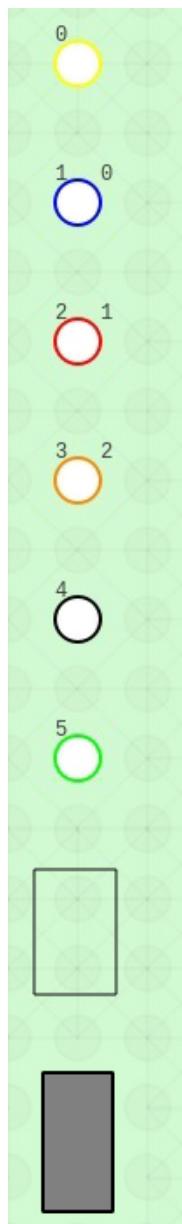
Gombok és egyéb elemek a bal oldalon

Szerkesztőgombok

NewU - Új egység felvétele. A gomb megnyomása után, míg a gomb zöld, a többi egységtől és az akadályuktól kellő távolságra a zöld felületre kattintva tehető le új egység.

MovU - Egység mozgatása. A gomb megnyomása után, míg a gomb zöld, egy egységet kijelölve (rákattintva), majd egy üres területre kattintva az adott egység áthelyezhető.

DelU - Egység törlése. A gomb megnyomása után, míg a gomb zöld, egy egységre kattintva, az törlődik. Az egérgombot nyomva tartva és egy területet kijelölve az adott területen minden egység törlődik.





DelAU - minden egység törlése.

NewO - Új akadály hozzáadása. A bal egérgombot lenyomva és nyomva tartva, egy területet kijelölve, azon a területen megjelenik egy akadály. Egységre nem tehető akadály, de más akadályra igen. Túlzottan vékony akadály nem tehető le.

RanO - Véletlenszerű helyre tesz akadályt. A letett akadály oldalhosszai legfeljebb a generáláskor beállított "Size of obstacles" érték+1 hosszúak lesznek. Betöltés és üres szimulátor generálása után az a maximális oldalhosszak olyanok, mintha a "Size of obstacles" érték 7 lenne.

DelO - Akadály törlése. A gomb megnyomása után, míg a gomb zöld, akadályra kattintva, az adott akadály törlődik, ha több van a kurzor alatt, akkor mindegyik. Az egérgombot nyomva tartva és egy területet kijelölve, az adott területet metsző minden akadály törlődik.

DelAO - minden akadály törlése.

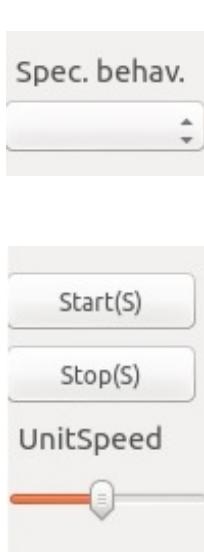
Clear - minden töröl a felületről.

Select - váltás kijelölő módba. Csak egységek jelölhetők ki.

Sel. All - váltás kijelölő módba és minden egység kijelölése.

HSUP - Egységek útvonaltervének elrejtése vagy megjelenítése, alapértelmezetten megjelennek az útvonalak.

Szimuláció beállítások



Választódoboz - Egységviselkedés kiválasztása (Követő, Megelőző, Elkerülő, Járőrző), részletek a "Használat, megjegyzések" szakasz alatt találhatók.

Start(S) - Szimuláció indítása (egységek mozoghatnak).

Stop(S) - Szimuláció megállítása (egységek nem mozoghatnak).

Csúszka - Aktuálisan kijelölt egységek sebességének állítása. Álló és elindított szimuláció mellett is érvényesül a hatása.

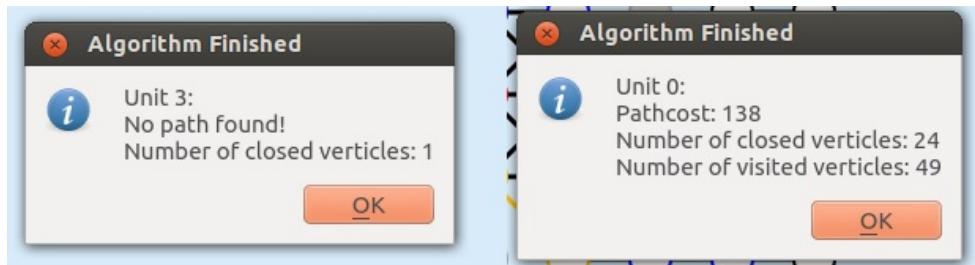
Megjegyzés: minden gomb és egyéb elem fölé helyezve a kurzort, rövid időn belül megjelenik egy rövid magyarázat az adott elemhez.

2.3.5. Hibaüzenetek, tájékoztató üzenetek

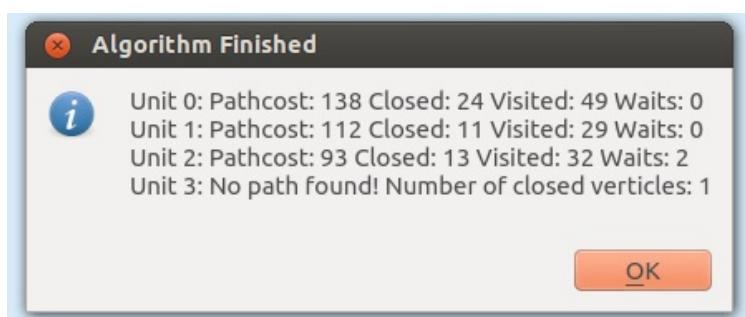
Érvénytelen területre helyezett pontok, élek, egységek és akadályok esetén nem jelenik meg hibaüzenet. Hasonlóan a szimulátorban érvénytelen helyre (például akadályba) küldött egységek esetén sincsen arról tájékoztatás.

Tájékoztató üzenetek

A gráftervezőben lefuttatott és végállapotba ért algoritmus tájékoztató üzenetet küld a megtalált vagy meg nem talált útról, utakról, leírva a használt egység számát, a megtalált út hosszát és az algoritmus során lezárt illetve elérte pontok számát.



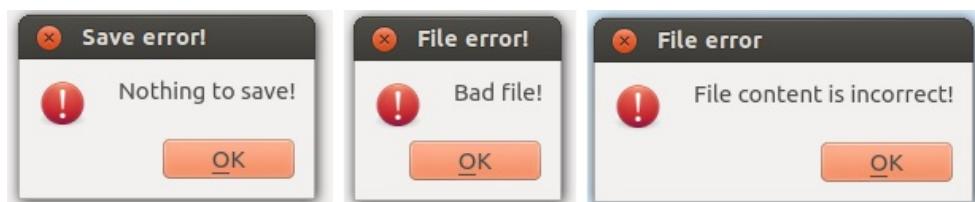
2.10. ábra. Egy egységes algoritmusok eredményét kiíró üzenetek



2.11. ábra. Több egységes algoritmus eredményét kiíró üzenet

Hibaüzenetek

Hibaüzenetet kapunk mentéskor, ha még nincsen megnyitva gráftervező vagy szimulátor, amit el lehetne menteni és betölteni, ha a betöltendő fájl hibás, azaz nem helyes gráfot vagy szimulátort ír le.



2.12. ábra. Hibaüzenetek

3. fejezet

Fejlesztői dokumentáció

3.1. Megoldási terv

A feladatot ablakos alkalmazásként valósítjuk meg C++ programozási nyelv és a Qt keretrendszer segítségével Linux operációs rendszeren. Ezen az ablakon nyílnak meg a gráf- és szimulátortervező és -kezelő felületek.

3.1.1. A Menü

Az ablak fejlécében helyezzük el a menüt, ahonnan az új szerkesztőket elindíthatjuk, illetve menthetünk és betölthetünk már elkészített gráfokat és szimulátorokat, és kiléphetünk az alkalmazásból. Szintén a fejlécben helyezzük el a rövid programbeli használati útmutató elérhetőségét.

Új gráfot lehet generáltatni a programmal vagy kézzel rajzolni. Ezeket külön menüpontokon keresztül érhetjük el. Véletlenszerű és rácsgráfot is lehet generáltatni, a kézi rajzoláshoz pedig egy üres rajzolófelületet nyitunk.

Új szimulátort nyithatunk üresen vagy véletlenszerűen elhelyezett elemekkel. Ezeket külön menüpontokon keresztül érhetjük el.

3.1.2. Generálási beállítások

Üres gráfrajzoló generálásához nem kell megadni semmi egyéb információt.

Rácsgráf esetén megadható a sorok és oszlopok száma és az akadályokkal és egységekkel kapcsolatos információ, például darabszám.

Véletlen gráf esetén megadható a pontok száma és az élekkel, akadályokkal és egységekkel kapcsolatos információk, például darabszám.

Üres szimulátorfelület generálásához nem kell megadni semmi egyéb információt.

Véletlenszerű szimulátoelemek generálásához megadhatjuk a lehelyezendő egységek és akadályok számát és az akadályok méretét valamilyen formában.

Ezeket a beállításokat külön dialógusablakban módosíthatjuk a generálás előtt, ha nem üres felületet generálunk.

3.1.3. Gráftervező felhasználói kezelőfelület

A főablakon nyílik meg. Bal oldalon egy oszlopban a tervezés során használható gombokat helyezzük el, az ablak legnagyobb részét pedig a tervező rajzfelület foglalja el.

A gráf rajzolásakor, szerkesztésekor lehetőségünk van új pontok letételére, pontok törlésére, áthelyezésére, élek behúzására és törlésére, akadályok felhelyezésére és törlésére és egységek létrehozására és törlésére.

A gráfon bemutathatóak keresőalgoritmusok. Az algoritmus neve, esetleges heurisztikus függvényének neve, az algoritmusszimuláció elindításához, szüneteltetéséhez, kezdőhelyzetbe állításához szükséges gombok szintén a bal oldali gombfelületen kapnak helyet.

A kívánt rajzolási művelet a megfelelő gomb megnyomása után a rajzterületen egér segítségével végezhető el. A kiválasztott algoritmus annak elindítása után folyamatosan fut, amíg meg nem állítjuk vagy véget nem ér.

A tényleges megvalósításhoz felhasználjuk a Qt keretrendszer előre megírt eszközöket, megjelenítőit és egyéb elemeit. Azokból való származtatással és újabb adattagok és függvények felvételével oldjuk meg a fentiek implementációját.

3.1.4. Szimulátor felhasználói kezelőfelület

A főablakon nyílik meg. Bal oldalon egy oszlopban a tervezés során használható gombokat helyezzük el, az ablak legnagyobb részét pedig a tervező rajzfelület foglalja el.

Szerkesztéskor lehelyezhetők új egységek és akadályok, illetve törölhetők azok. Az egységek áthelyezhetők. A kívánt rajzolási művelet a megfelelő gomb megnyomása után a rajzterületen egér segítségével végezhető el.

A szimuláció indításához és megállításához szükséges gombokat és az egységek mozgásformáját befolyásoló beállításokat lehetővé tevő elemeket szintén a bal oldali gombfelületre helyezzük.

A szimuláció elindítása után a rajzfelületen elhelyezett egységek egérrel irányíthatók egyesével vagy akár egyszerre. Végezhetnek speciális mozgásokat, például más egység követését vagy elkerülését.

A tényleges megvalósításhoz felhasználjuk a Qt keretrendszer előre megírt eszközeit, megjelenítőit és egyéb elemeit. Azokból való származtatással és újabb adattagok és függvények felvételével oldjuk meg a fentiek implementációját.

3.1.5. Gráftervező megvalósítási terv

A gráftervezőt modell-nézet architektúrában valósítjuk meg. A modell és nézet közötti kommunikációt szignálok segítségével oldjuk meg. A főablak csak a menünél megadott akciók elindítását végzi.

Modell

A modellben valósítjuk meg a gráfot. Mivel nem számítunk sokélű pontokra ezért éllistás reprezentációt használunk a gráf pontjainak tárolásához. Külön tárolandók az élek és egységek. Mind magát a gráfot, mind annak pontjait, éleit és egységeit külön osztályban adjuk meg. A modellben kezelni kell a gráfelemek hozzáadását és törlését, és tárolni kell az algoritmusok futásakor érdekes információkat, például az élek hosszát. A gráf azonos típusú elemeit megkülönböztetjük egy egyedi azonosítóval, és ez az azonosító köti össze a modell és nézet megfelelő elemeit is.

A keresőalgoritmusok futtatásáért is a modell felelős. Az algoritmus a rá jellemző listákat, értékeket tárolja, miközben az előbb röviden jellemzett gráfon keres. A keresés folyamata és eredménye megfigyelhető a nézeten, ehhez a megfelelő lépésekkel küldött szignálokat használunk.

A modell végzi a gráfmentést és -betöltést.

Nézet

A nézet felel a grafikus elemek megjelenítéséért, amelyek a modellben létrehozott gráf elemeinek felelnek meg. A grafikus elemek megvalósításához használjuk a Qt keretrendszer beépített grafikai elemeit és megjelenítőjét és azokból való származtatással oldjuk meg a további szükséges adattagok hozzáadását a megfelelő osztályokhoz.

3.1.6. Szimulátor megvalósítási terv

A szimulált részben modell-nézet architektúrában valósítjuk meg. Az egységek és akadályok kezelése teljesen a nézet feladata, de a területre lefektetünk egy nyolc-kapcsolt rácsgráfot, amelyen az egységek az útkeresést végzik. Ez a gráf a gráftervező modelljében is használt gráfmodell.

Modell

A modell ugyanaz, mint a gráftervezőnél, de itt a tárolt gráf közvetlenül nem módosítható a szimulátor rajzfelületén, viszont a nézetre lehelyezett akadályok és egységek hatással vannak rá. A gráf egy a teljes egységek által bejárható területre kiterített nyolc-kapcsolt rácsgraf. Ezen a gráfon keresnek utat a szimulátorban lehelyezett egységek, a gráftervező modelljében megírt algoritmussal.

Nézet

A szimulátor nézete jeleníti meg a grafikus elemeket illetve elvégzi az egységek és akadályok szerkesztése során jelentkező feladatokat, tárolja és kezeli az egységek és akadályok minden adatát, és azok változásait közvetíti a modell felé, hogy az megfelelő módon módosíthassa a lefektetett gráfot.

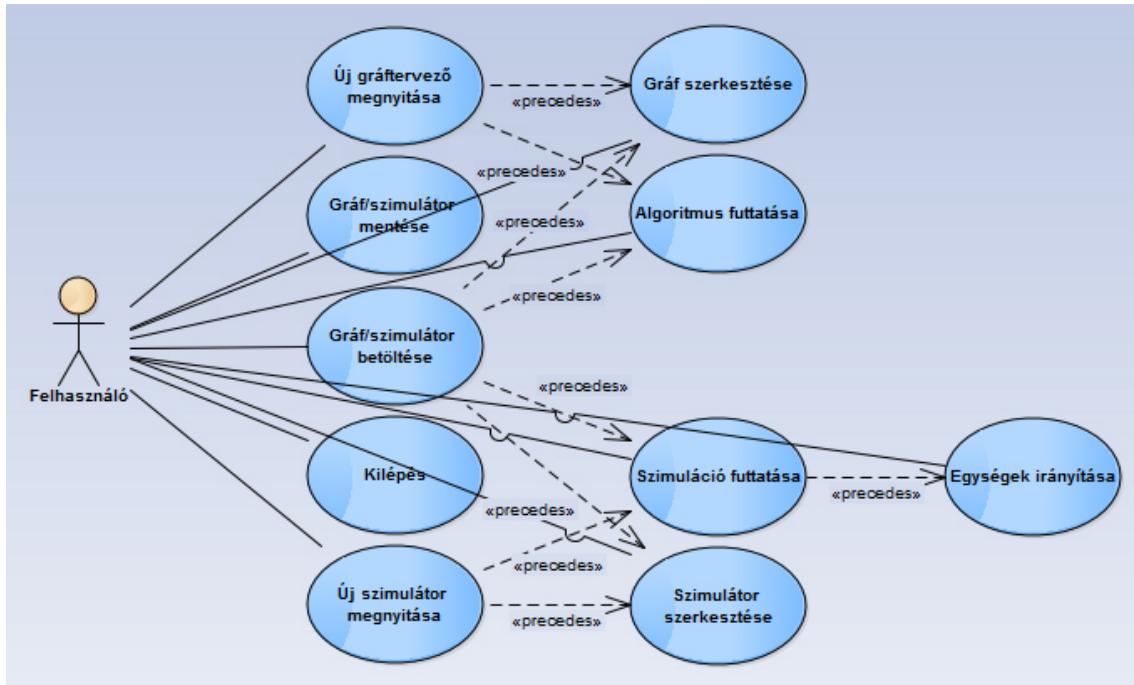
A szimulátor elemeinek megvalósítása a Qt keretrendszer beépített grafikus elemeiből való származtatással történik, újabb adattagok felvételével és a megfelelő metódusok felüldefiniálásával. Az azonos típusú elemek egyéni azonosítóval rendelkeznek. Egységeknél ez az azonosító köti össze a modellbeli gráf egységeit a megjelenítő egységeivel.

A nézet felel a szimulátormentés és -betöltés lebonyolításáért.

3.2. Megvalósítás

3.2.1. Felhasználói esetek

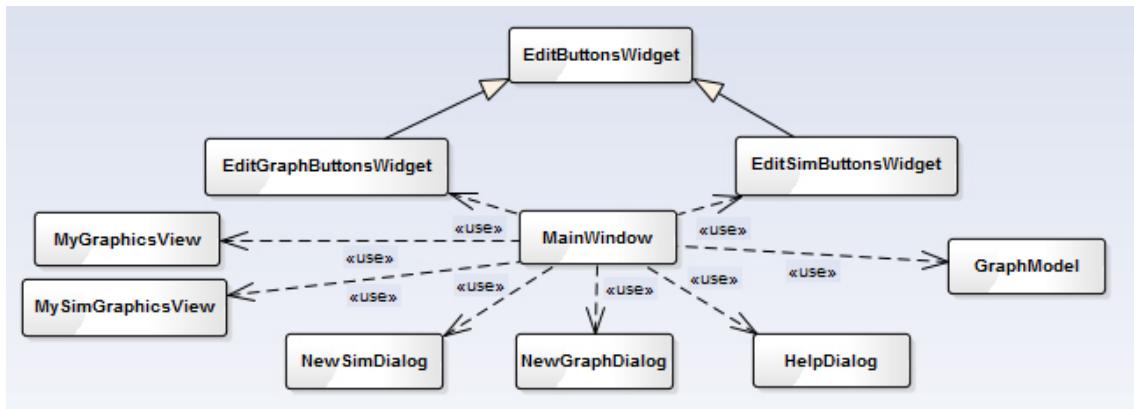
Az alábbi diagram csak az általános felhasználói eseteket mutatja, a részletektől azok mennyisége miatt eltekintettünk.



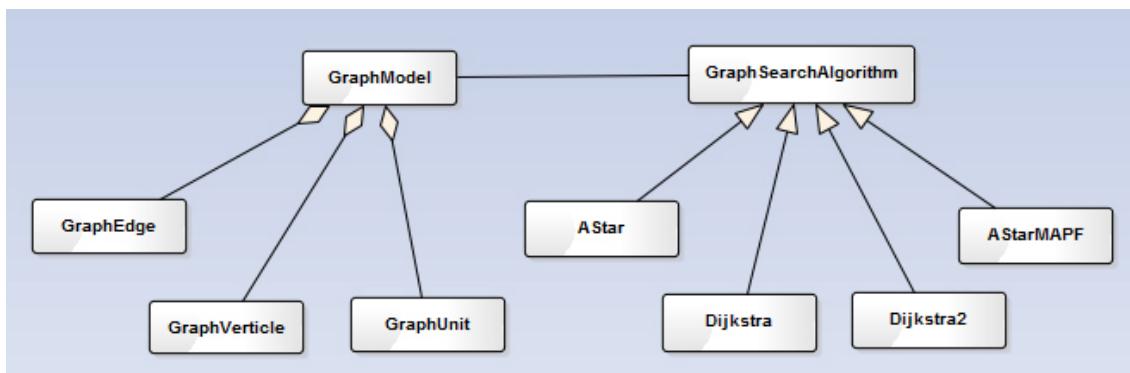
3.1. ábra. Az alkalmazás legalapvetőbb felhasználói esetei

3.2.2. Főbb osztályok kapcsolata

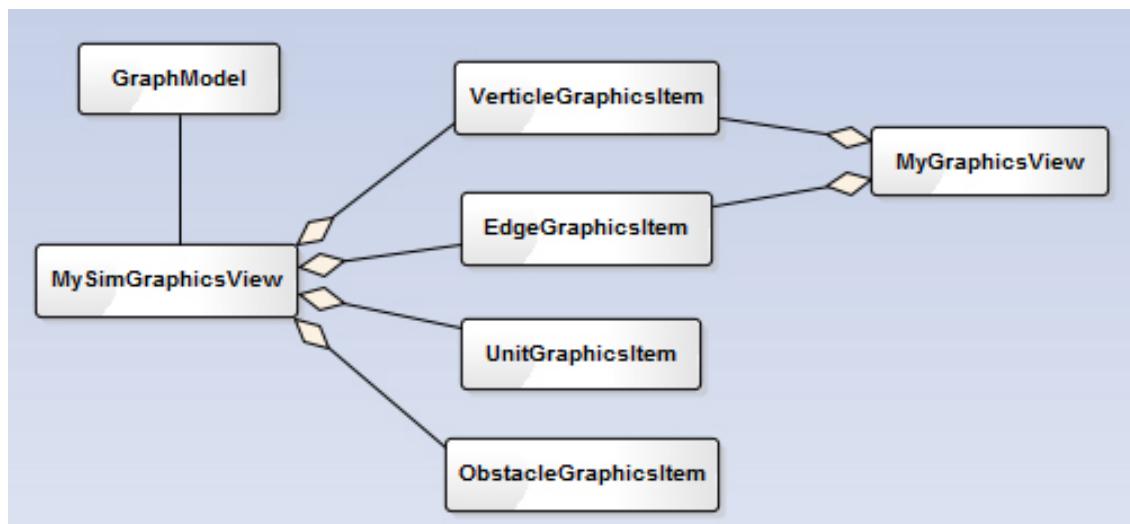
Az alábbi ábrákon csak a leglényegesebb osztályok közötti kapcsolatokat tüntet-tük fel.



3.2. ábra. A főablak kapcsolatai



3.3. ábra. A gráf- és algoritmusmodell kapcsolatai



3.4. ábra. A nézetek kapcsolatai

3.2.3. Főablak

A főablakot a `QMainWindow` osztályból leszármazó `MainWindow` osztályban implementáltam. A hozzá tartozó kód a `mainwindow.h` és `mainwindow.cpp` fájlokban található.

Adattagok

A főablak privát adattagjai a saját megjelenítését meghatározó `Ui`, a rajta megjelenő kezelők (gomb- és rajzfelületek), az új felületek beállításait megjelenítő dialógus ablakok, a használati útmutató dialógus ablakja és a gráftervező modellje.

Metódusok

A konstruktur minden adattagnak 0 értéket ad, és összeköti a menüsor által küldött szignálokat a főablak megfelelő metódusával. A destruktur törli az adattagokat.

Az alkalmazásból való kiléést az **Exit**, a mentés és betöltés kezdeményezését és a hozzájuk szükséges elenőrzéseket a **Save** és **Load** metódusok végzik. **ShowHelp** jeleníti meg az útmutatót tartalmazó ablakot.

NewGraph hozza létre az új gráftervezőt és a hozzá tartozó gombfelületet (**_GraphSideBar**), és összekapcsolja a modell, a nézet és a gombfelület megfelelő szignáljait és metodusait. Ezt hívja a **NewDrawnGraph** csupa 0 értékkel. A **NewGridGraph** metodus a **_NewGraphDialog** dialógusablak rácsgráfra értelmezett változatát jeleníti meg és az ott beállított értékekkel hív **NewGraph**-ot. **NewRandomGraph**-ban a **_NewGraphDialog** véletlen gráfhoz illő változata jelenik meg, és annak értékeivel hív **NewGraph**-ot. **Load** is 0 értékekkel hívja a **NewGraph**-ot, de átadja neki a betöltendő fájl elérési útját.

NewSim hozza létre az új szimulátort és az ahhoz tartozó **_SimSideBar** gombfelületet és összekapcsolja gombok szignáljait a szimulátor megfelelő metódusaival. A **NewEmptySim** üres szimulátort nyit a **NewSim** segítségével. **NewOwnSim** megnyitja a véletlen szimulátor beállításait megjelenítő **_NewSimDialog** dialógus ablakot és annak értékei alapján generál szimulátort a **NewSim** meghívásával. **Load** is üres szimulátort nyit **NewSim** hívással, de átadja neki a betöltendő fájl elérési útját.

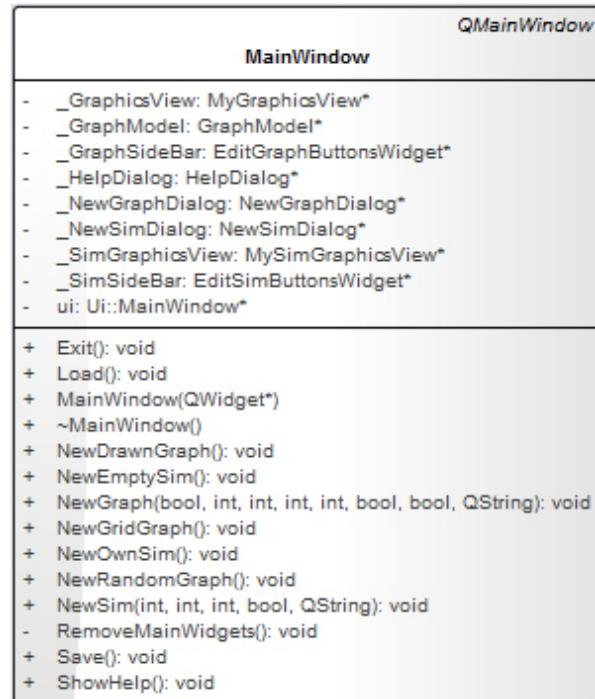
A generálások közötti kezelőeltávolítás a **RemoveMainWidgets** metodusban történik.

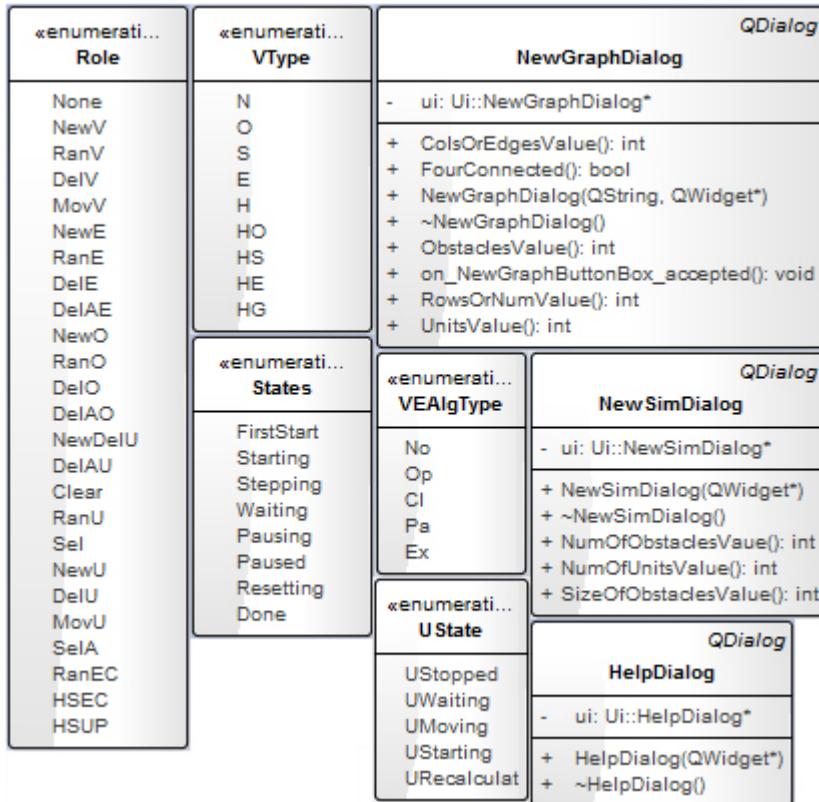
3.2.4. Dialógus ablakok

A három dialógus ablak mindegyikének őse a **QDialog** beépített osztály.

Az útmutatót a **HelpDialog** ablakban egy szövegdobozban jelenítjük meg.

Az új gráf beállításait a **NewGraphDialog**-al módosíthatjuk, az új szimulátorét pedig a **NewSimDialog**-al. Konstruktor-ukban állítjuk be felületi vezérlőket és azok





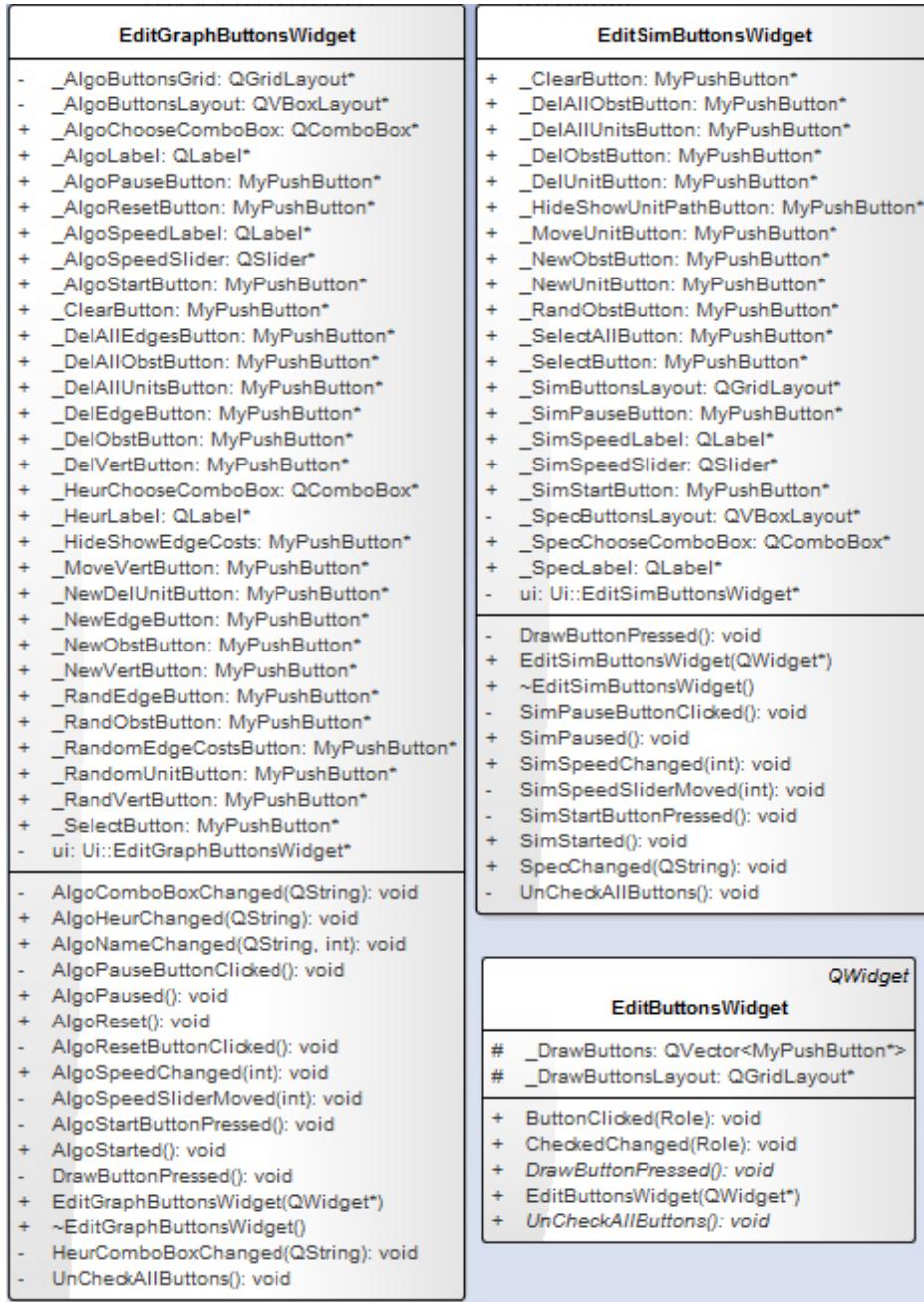
3.5. ábra. A dialógus ablakok és később használt felsorolási típusok diagramja

néhány tulajdonságát. A vezérlők értékének lekéréséhez külön metódusokat vettünk fel.

3.2.5. Gombfelületek

Az `EditButtonsWidget` osztály ōse a `QWidget` osztály. `EditButtonsWidget` az `EditGraphButtonsWidget` és `EditSimButtonsWidget` osztályok közös ōse, és a gombnyomások hatását valósítja meg (`DrawButtonPressed`).

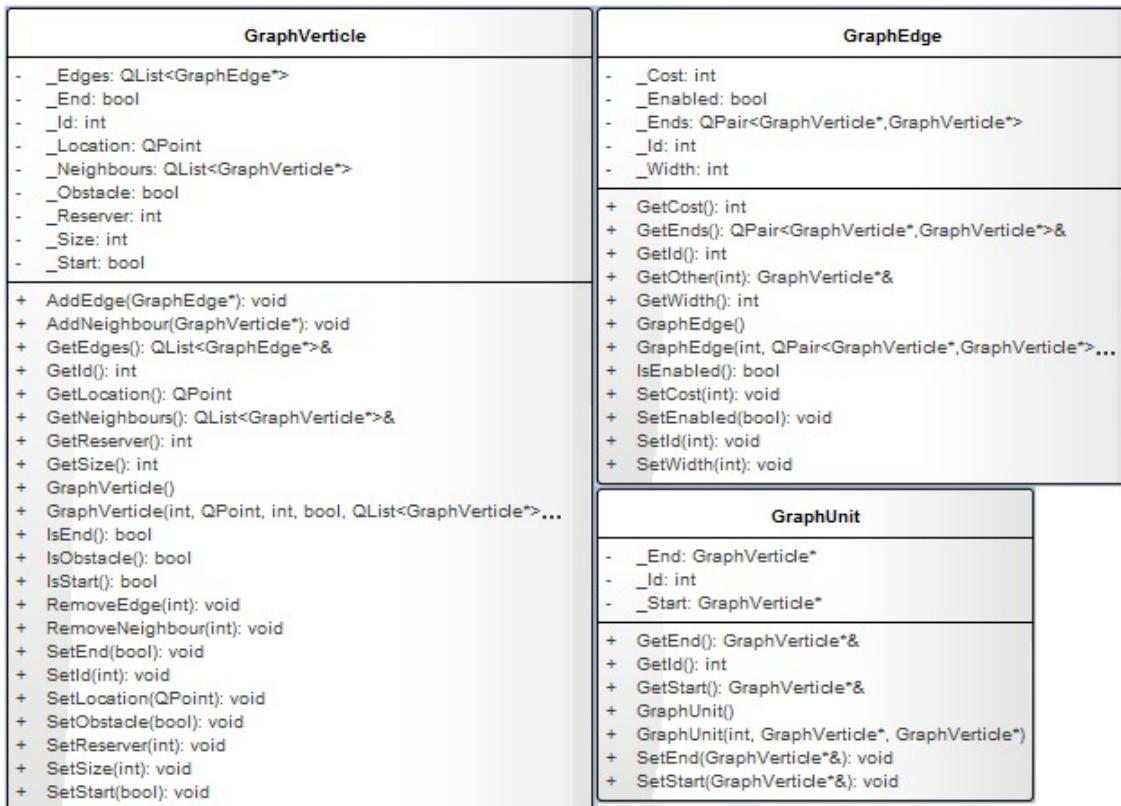
A gráftervező illetve a szimulátor gombfelülete az `EditGraphButtonsWidget` és az `EditSimButtonsWidget` kezelőkön jelenik meg, amik közül az aktuálisat a főablak bal oldalára helyezünk fel. A rajtuk levő gombok `MyPushButton` típusúak, ami a `QPushButton` osztályból származtatott, egy `_Role` adattaggal kiegészített osztály. A `_Role` típusa egy felsorolási típus `Role` (3.5 ábra) A konstruktur-okban tesszük fel az adattagként tárolt gombokat és egyéb kezelőket, beállítjuk bizonyos lényeges tulajdonságaikat, és a gombnyomások szignáljait kezelőmetódusokhoz kapcsoljuk.



3.6. ábra. A gombfelületek diagramja

3.2.6. Gráfmodell használt osztályai

A gráf modellje 4 főbb osztályt tartalmaz. Az éleket megvalósító **GraphEdge**, a pontokat megvalósító **GraphVerticle**, az egységeket megvalósító **GraphUnit** és a magát a gráfot megvalósító **GraphModel** osztályt.



3.7. ábra. A gráftervező által használt osztályok diakgramja

Élek

A `GraphEdge` osztályban adattagként tárolom az él azonosítóját (`_Id`), végpontjait egy `GraphVerticle` mutatópárként (`_Ends`), súlyát (`_Cost`), és hogy engedélyezett él-e (`_Enabled`), azaz nincs-e akadály valamelyik végén. Az adattagokhoz felvettem lekérő és beállító metódusokat, illetve **konstruktur**-ban is beállíthatók. Az azonosító csak **konstruktur**-ban adható meg.

Pontok

A `GraphVerticle` osztályban adattagként tárolom a pont azonosítóját (`_Id`), eleit egy listában (`_Edges`), hogy akadály-e, hogy egység kezdő- vagy végpontja-e és az egyszerűbb átfedéselkerülés érdekében a megjelenítőn vett helyzetének koordinátáit (`_Location`) és a méretét (`_Size`) is, illetve egy a szimulátor egységei által használt értéket (`_Reserver`). Az adattagokhoz felvettem lekérő és beállító metódusokat, illetve bizonyosak **konstruktur**-ban is beállíthatók. Az azonosító csak **konstruktur**-ban adható meg.

Egységek

A `GraphUnit` osztályban adattagként tárolom az egység azonosítóját (`_Id`) és kezdő- és végpontját. Az adattagokhoz felvettek lekérő és beállító metódusokat, illetve `konstruktor`-ban is beállíthatók. Az azonosító csak `konstruktor`-ban adható meg.

3.2.7. Gráfmodell

A `GraphModel` osztályt a `QObject` osztályból származtatom, így használhatóak benne a Qt keretrendszer sajátosságai, például a szignálok.

A `GraphModel` külön listában tárolja a gráf pontjait (`_GraphVertices`), élét (`_GraphEdges`) és egységeit (`_GraphUnits`). Ezekhez a listákhoz fűznek hozzá újabb elemet a megfelelő ellenőrzések (átfedés, dupla él, stb.) után az `Add...` és `AddRandom...` függvények a megfelelő elemnévvel befejezve, illetve az `...Obstacle` végűek akadályá tesznek egy pontot. Mindhárom elemtípushoz fenntartunk egy adattagot, amiben a legnagyobb azonosítójú adott típusú elem azonosítójánál eggel nagyobb értéket tároljom (`_Max...Id`). minden elem a gráfban lekérdezhető azonosító alapján, az ezt végrehajtó függvények lineáris kereséssel keresnek a megfelelő listában. (Mivel az új elemek azonosítója mindenkor az aktuális legnagyobb és az új elemet a megfelelő lista végére fűzzük, ezért lehetne azonosító szerinti logaritmikus keresésre javítani a lekérdezéseket.) Az elemek törlését a `Remove...` és `RemoveAll...` függvények végezik, illetve a `ClearAll` minden töröl. Mind a hozzáadó, mind a törlő-függvények azonosító(ka)t kapnak paraméterként, ezzel egyszerűsítve a nézettel való kommunikációt.

További adattagokként jelennek meg a gráfgeneráláskor felhasznált értékek, például hogy véletlen-e a gráf (`_Rand`), a generálandó sorok vagy pontok számát tároló adattag (`_RowsOrNum`). A generálást a `Generate` függvény végzi, ami minden adattagot a megfelelő kezdőértékre állít, majd attól függően, hogy rácsot vagy véletlen gráfot generál-e sorra felveszi a pontokat, az éleket és végül az egységeket.

Adattagként tárolom a gráfszerkesztéshez kapcsolódó bizonyos információkat, például az éppen felvenni készült egység vagy él végpontjait (`_NewUnitEnds` és `_NewEdgeEnds`), az éppen mozgatott pont azonosítóját (`_MoveVertId`) és a nézeten éppen benyomva tartott gomb feladatát (`_Checked`), hogy aszerint tudjunk megfelelő műveletet végrehajtani a gráfon, ezeket főként a `View...` kezdetű függvények kezelik.

A keresőalgoritmust megvalósító osztályból is felveszünk egy példányt a gráfhoz, illetve ahhoz olyan metódusokat, amik a nézeten végrehajtott algoritmussal

<p>GraphModel</p> <ul style="list-style-type: none"> - _Algorithm: GraphSearchAlgorithm* - _Checked: Role - _ColsOrEdges: int - _EWidth: int - _GraphEdges: QList<GraphEdge*> - _GraphUnits: QList<GraphUnit*> - _GraphVertices: QList<GraphVerticle*> - _MaxEdgeld: int - _MaxUnitId: int - _MaxVertield: int - _MoveVertId: int - _Neighbourhood: int - _NewEdgeEnds: QPair<int,int> - _NewUnitEnds: QPair<int,int> - _Obstacles: int - _Rand: bool - _RowsOrNum: int - _VSize: int <ul style="list-style-type: none"> + AddEdge(int, int, int): bool + AddObstacle(int): void + AddRandomEdge(): void + AddRandomObstacle(): void + AddRandomUnit(): void + AddRandomVerticle(): void + AddUnit(int, int, int): bool + AddVerticle(QPoint, bool, int): bool + AlgoDone(int, int, int, int): void + AlgoEdgeOnPath(int): void + AlgoPartDone(int, int, int, int, int): void + AlgoReset(): void + AlgoVerticleClosed(int): void + AlgoVerticleExpanding(int): void + AlgoVerticleOnPath(int): void + AlgoVerticleOpened(int): void + ClearAll(): void + CountNonObstVerts(QPoint, QPoint): int + CountVerts(QPoint, QPoint): int + DisableEdge(int): void + EdgeAdded(int, QPoint, QPoint, bool, int): void + EdgeRemoved(int): void + EdgeUpdated(int, QPair<QPoint,QPoint>, bool, int): void + EnableEdge(int): void 	<p>QObject</p> <ul style="list-style-type: none"> + EWidthChanged(int): void + Generate(bool, int, int, int, int, int, bool): void + GenerateForSim(): void - GetEdgeByld(int, GraphEdge*&): bool + GetEWidth(): int + GetUnitByld(int, GraphUnit*&): bool + GetUnitList(): QList<GraphUnit*> + GetVerticleByld(int, GraphVerticle*&): bool + GetVerticleList(): QList<GraphVerticle*> - GetVerticesBylds(int, int, GraphVerticle*&, GraphVerticle*&): bool + GetVSize(): int + GraphModel() + ~GraphModel() + HideShowEdgeCosts(): void + Load(QString): void + MoveVerticle(int, QPoint): void + ObstacleAdded(int): void + ObstacleRemoved(int): void + RandomizeEdgeCosts(): void + RemoveAllEdges(): void + RemoveAllObstacles(): void + RemoveAllUnits(): void + RemoveEdge(int): void + RemoveObstacle(int): void + RemoveUnit(int): void + RemoveVerticle(int): void + Save(QString): void + UnitAdded(int, int, int): void + UnitRemoved(int, int): void + UpdateEdges(GraphVerticle*): void + VerticleAdded(int, QPoint, bool): void + VerticleMoved(int, QPoint): void + VerticleRemoved(int): void + ViewAlgoHeurChanged(QString): void + ViewAlgoNameChanged(QString, int): void + ViewAlgoPaused(): void + ViewAlgoReset(): void + ViewAlgoSpeedChanged(int): void + ViewAlgoStarted(): void + ViewButtonClicked(Role): void + ViewCheckedChanged(Role): void + ViewEdgeSelected(int): void + ViewMouseReleased(QPointF): void + ViewVerticleSelected(int): void + VSizeChanged(int): void
---	--

kapcsolatos változtatásokat kezelik, például az algoritmus sebesség-változását kezelő `ViewAlgoSpeedChanged` és a többi `ViewAlgo...` kezdetű metódus.

A `RandomizeEdgeCosts` függvény az élhosszakat véletlenszerűre állítja, míg az `UpdateEdges` egy csúcs változása után frissíti éleinek adattagjait. A gráf tényleges mentését és betöltését végzik a `Save` és `Load` metódusok. A mentés a megjelenítés és azonosítás szempontjából fontos információkat fájlba írja minden elemről, de az algoritmussal kapcsolatos aktuális információkat nem menti. A `Load` egy előbb generált üres felületre helyezi fel sorban az elemeket.

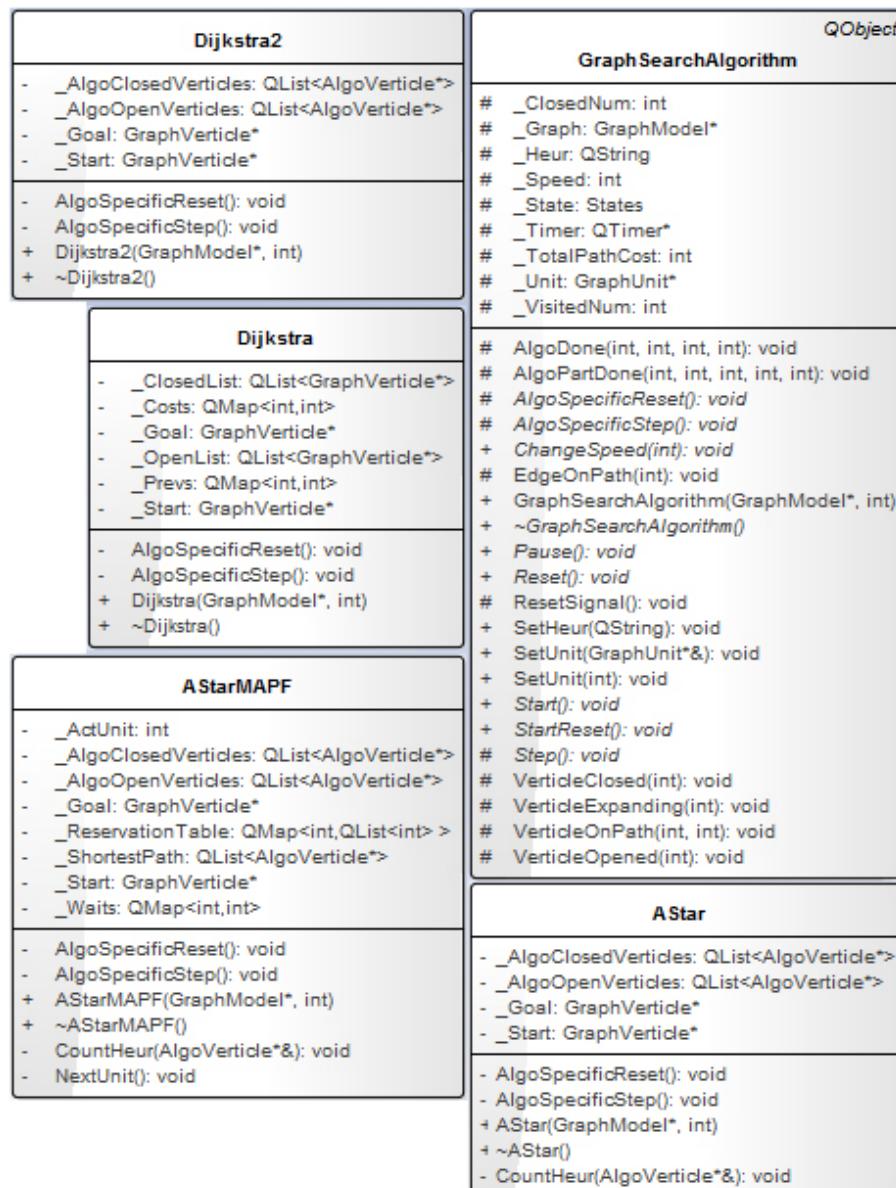
Csak az algoritmus osztálya által használt függvények a `GetUnitList`, amellyel az egységlista kérhető le, és a két különleges heurisztikaszámoló függvény, `CountVerts` és `CountNonObstVerts`, amik megszámolják mennyi (nem akadály) pont van egy adott téglalapban.

Csak a szimulátor által használt függvények a `GenerateForSim`, ami a térképre fektetett gráfot generálja le, a `GetVerticleList`, ami a pontlistát adja vissza, és az

egy adott él engedélyezéséért és tiltásáért felelős `EnableEdge` és `DisableEdge`.

A gráfmodell szignálokkal jelzi, ha felvett (...`Added`) vagy törölt (...`Removed`) egy elemet, frissített egy élt (`EdgeUpdated`), illetve saját szignálként továbbítja az algoritmus szignáljait.

3.2.8. Algoritmusmodell



3.8. ábra. Az algoritmusosztályok diakgramja

A konkrét algoritmusosztályok (`Dijkstra`, `Dijkstra2`, `AStar`, `AStarMAPF`) egy közös ősosztályból származnak, ez a `GraphSearchAlgorithm` osztály, melynek őse a `QObject` osztály, a `GraphModel`-éhez hasonló okokból.

A `GraphSearchAlgorithm` konstruktőr-ában meg kell adni a gráfot, amelyen a keresés futni fog, illetve az algoritmus kezdeti sebességét, ezeket adattagként eltárolom. Fontos adattagok továbbá a megadott sebességet tartó időzítő (`_Timer`), az algoritmus aktuális állapotát (külön `States` felsorolási típusbanban definiált: induló, várakozó, stb (3.5 ábra)) jelző `_State`, az aktuálisan használt heurisztikát tároló `_Heur`, és az aktuálisan figyelembe vett egységet tároló `_Unit`. Utóbbi kettőhöz felvettem beállító metódusokat, amiket az algoritmust futtató gráf használ.

Minden konkrét algoritmus ugyanolyan időzítő és helyzetállításokkal indul, áll le, és változtat sebességet, ezért ezen metódusokat (`Start`, `Pause`, `StartReset`, `ChangeSpeed`) az ősosztályban valósítjuk meg. A kezdőhelyzetbe állításban és az algoritmus egy lépésének végrehajtásában is vannak közös pontok, ezért azokra is felveszik az ősben egy-egy függvényt (`Reset` és `Step`). Az algoritmusspecifikus lépések e kettő függvényben az `AlgoSpecificReset` és `AlgoSpecificStep` metódusokban definiálják fölül. Mivel minden algoritmus azonos szignálokat küld, ezért azokat is az ősben definiáljuk. Jelezzük a pont megnyitását, bezárását, hogy éppen egy adott pontot fejt ki az algoritmus, és hogy egy adott pont a megtalált legrövidebb út része (`Verticle...` szignálok). Jelezzük ha egy él a legrövidebb úton van (`EdgeOnPath`), ha kezdőhelyzetbe állt (`ResetSignal`) vagy lefutott a algoritmus (`AlgoDone`), illetve többszereplős A* esetén, ha az algoritmus egy részfeladata befejeződött (`AlgoPartDone`).

Dijkstra

Feltételezem, hogy aki ezt a dokumentációt olvassa ismeri Dijkstra algoritmusát, ezért annak külön ismertetésétől eltekintek.

Ezt az algoritmust kétféleképpen valósítottam meg, de ezek csak a felhasznált adattagokban különböznek, mivel maga az algoritmus jól definiált.

Az első változat gráfpontlistaként tárolja a nyílt és zárt csúcsokat és külön QMap-ben tárolja az aktuális pontköltségeket és megelőző csúcsokat.

«struct»	
AlgoVerticle	
+	<code>Cost: int</code>
+	<code>Heur: int</code>
+	<code>Prev: AlgoVerticle*</code>
+	<code>Step: int</code>
+	<code>Vert: GraphVerticle*</code>
+	<code>AlgoVerticle(GraphVerticle*&, AlgoVerticle*&, int)</code>

A második változathoz felvettem egy segédstruktúrát (`AlgoVerticle`), amiben adattagként megjelenik a csúcs (`Vert`) mellett annak megelőzője (`Prev`), költsége (`Cost`), heurisztikus értéke (`Heur`), ha van, illetve egy a többszereplős A* által használt adattag (`Step`) ami azt tárolja,

hogy az adott pont a hozzá vezető legrövidebb úton hányadik lépésben következik. Ilyen `AlgoVerticle`-k listájaként tárolja a `Dijkstra2` a nyitott és zárt csúcslistát,

és QMap-eket nem használ.

Közös adattag a kezdő és végpontot tároló `_Start` és `_Goal`. Mindkét esetben az `AlgoSpecificReset` kiürít minden használt listát és beállítja a keresés kezdő- és végpontját, ha lehet, a `_Unit` értékétől függően. Az `AlgoSpecificStep` szintén csak a használt listákban különbözik, és az algoritmus ismert lépését hajtja végre. Először ellenőrzi van-e még nyitott csúcs és ha nincs véget ér megtalált út nélkül. Ha nem üres a nyitott csúcs lista, akkor megkeresi annak legkisebb költségű elemét. Ha az a célcíms, akkor végeztünk és visszakeressük a legrövidebb utat a beállított megelőző csúcsok alapján, ha pedig nem célcíms akkor kifejti a le nem zárt szomszédainak frissítésével vagy megnyitásával, majd lezárja azt. Mindeközben elküldi a megfelelő szignálokat, amik alapján a megjelenítő majd rajzolni tud.

A*

Az A* keresés lépéseihez hasonló a Dijkstra algoritmushoz, de minden csúcshoz rendel egy heurisztikus értéket, és a nyitott listában a tényleges költség és e heurisztikus érték összege szerint keresi a legkisebb költségű csúcsot. Ennek folyománya, hogy ha a heurisztikus függvény nem minden alulról becsli a céltól való tényleges távolságot, akkor nem biztos, hogy az algoritmus legrövidebb utat ad eredményül.

Az A*-ot megvalósító `AStar` osztály a `Dijkstra2`-vel azonos adattagokat használ, de felvettünk hozzá egy heuristikaszámoló metódust (`CountHeur`), ami a beállított heurisztikus függvény alapján számol euklideszi vagy Manhattan távolságot, vagy hívja a gráf `CountNonObstVerts` vagy `CountVerts` heuristikaszámoló függvényeit, amik az aktuális csúcs és a cél közötti (nem akadály) csúcsokat számolják meg.

Az `AlgoSpecificReset` a `Dijkstra`-éval lényegében azonos, ahogy az algoritmusra jellemző lépést végrehajtó `AlgoSpecificStep` is, ami csak a legkisebb költségű csúcs megkeresését máshogyan, ugyanis a nyitott csúcsokhoz megnyitáskor kiszámolt heurisztikus értéket hozzáadja a ténylege költséghez és ezen összeg szerint keres minimális nyitott csúcsot.

Többszereplős A*

A többszereplős keresések (Multi-Agent Pathfinding) feladata, hogy egyszerre több kezdő és végpont között találjanak minél rövidebb utakat úgy, hogy a megtalált utakon végighaladó egységek ne ütközzenek. Itt erre egy egyszerű megoldást valósítottam meg az `AStarMAPF` osztályban, ami csak az azonos csúcsra érkezést gátolja meg várakozás lépések beiktatásával, az egy élen szemben áthaladást és egyéb lehetséges ütközéseket nem. Az algoritmus sorban kiszámolja az egységekhez a legrövidebb útjukat, majd annak lépéseiit egy foglalási táblába írja, és ha az adott lépésnél

a célcímsúcs foglalt akkor beiktat egy várakozási lépést. A célba jutott egységeket az algoritmus nem veszi a célon várakozónak

Az adattagok közé az **AStar** adattagjain kívül bekerül a legrövidebb utat tároló **_ShortestPath**, a foglalási tábla (**_ReservationTable**), amiben az egységek lépésekkel lefoglalt csúcsait tartjuk számon, az aktuálisan számolt egység azonosítója (**_ActUnit**) és az aktuális egység várakozási lépéseinak száma. Metódusok közé pedig a következő egységre ugró **NextUnit** függvényt vettük fel az **AStar** metódusain kívül.

Az **AlgoSpecificReset** itt is az adatlistákat üríti ki és beállítja a kezdőegységet (a gráf egységlistájának első elemét, ha a lista nem üres). Az **AlgoSpecificStep** egy egységre az A* lépéseit hajtja végre, de közben nyomon követi, hogy az aktuális egység az adott kinyitott csúcsra hányadikként lépne az odavezető legrövidebb úton és ezt letárolja az **AlgoVerticle** struktúra **Step** adattagjában. A legrövidebb út visszakövetésekor feltölti fordított irányban a **ShortestPath** listát, majd azon végigfutva feltölti a foglalási táblát és beiktat várakozási lépéseket ha szükséges. Ezután ha van még egység a listában, akkor a következőre ugrik, ha pedig nincs több, akkor leáll.

3.2.9. Gráftervező nézete által használt osztályok

<i>QGraphicsEllipseItem</i>	<i>QGraphicsLineItem</i>
VerticleGraphicsItem	EdgeGraphicsItem
<ul style="list-style-type: none"> - _AlgType: VEAAlgType - _Id: int - _Pos: QPoint - _Selected: bool - _Size: int - _Type: VType - _UnitId: int <ul style="list-style-type: none"> + boundingRect(): QRectF {query} + GetAlgType(): VEAAlgType + GetId(): int + GetPos(): QPoint + GetType(): VType + IsUnit(): bool + paint(QPainter*, QStyleOptionGraphicsItem*, QWidget*): void + SetAlgType(VEAAlgType): void + SetPos(QPoint): void + SetSelect(bool): void + SetUnitId(int): void + SetType(VType): void + shape(): QPainterPath {query} + VerticleGraphicsItem(int, QPoint, bool, int, QGraphicsItem*) + ~VerticleGraphicsItem() 	<ul style="list-style-type: none"> - _AlgType: VEAAlgType - _Cost: int - _Enabled: bool - _Id: int - _Selected: bool - _Type: VType - _Width: int <ul style="list-style-type: none"> + boundingRect(): QRectF {query} + EdgeGraphicsItem(int, QPoint, QPoint, bool, int, int, QGraphicsItem*) + ~EdgeGraphicsItem() + GetAlgType(): VEAAlgType + GetId(): int + GetType(): VType + paint(QPainter*, QStyleOptionGraphicsItem*, QWidget*): void + SetAlgType(VEAAlgType): void + SetCost(int): void + SetEnabled(bool): void + SetSelect(bool): void + SetType(VType): void + shape(): QPainterPath {query}

Pontok

A nézet felelős a pontok és élek megjelenítéséért. Ennek megfelelően felhasznál kettő, a Qt keretrendszer beépített grafikai elemeiből származtatott osztályt,

a `QGraphicsEllipseItem`-ből származó `VerticleGraphicsItem`-et (pontok) és a `QGraphicsLineItem`-ből származó `EdgeGraphicsItem`-et (élék).

A `VerticleGraphicsItem` osztály adattagjai között megjelenik egy azonosító (`_Id`), amivel a modellbeli gráf pontjaihoz kapcsolható a nézetbeli pont, a pont típusa (`_Type`), amiben megadható, hogy a pont sima, akadály, egység kezdő- vagy végpont-e, a pont mérete (`_Size`), az esetlegesen hozzá tartozó egység azonosítója (`_UnitId`), a pont algoritmusbeli szerepét (sima, nyitott, zárt, stb.) tároló (`_AlgType`) és hogy a pont ki van-e jelölve (`_Selected`). Metódusként felvettettem az adattagokhoz beállító és lekérdező függvényeket. A `_Type` és `_AlgType` lehetséges értékeit egy-egy felsorolási típusban adtam meg (`VType` és `VEAlgType` (3.5 ábra)).

Az osztály konstruktor-ában az adattagoknak adunk kezdőértéket. A függvények között szerepel továbbá a kirajzoláshoz szükséges `QGraphicsEllipseItem`-beli metódusok fölüldefiniált megvalósítása (`paint`, `boundingRect` és `shape`). A `boundingRect` a grafikus elemet körülvevő téglalapot adja vissza, a `shape` egy annál pontosabb alakot (ezt használja a beépített ütközésellenőrző (`collidesWithItem`)), a `paint` pedig a kirajzolás pontos menetét definiálja. A `paint` metódusban adjuk meg a típusuktól függő színezéseket, az esetleges szövegkiírásokat, és ez rajzolja meg az adott grafikai elemet.

Élek

Az `EdgeGraphicsItem` osztály adattagként tárolja az él azonosítóját (`_Id`), súlyát (`_Cost`), megjelenítéskori szélességét (`_Width`), a pontokéhoz hasonló két típusértéket (`_Type` és `_AlgType`), és hogy az él ki van-e jelölve (`Selected`), illetve engedélyezett-e (`Enabled`). Az adattagokhoz felvettünk beállító és lekérdező függvényeket, valamint itt is fölülírtuk a `QGraphicsLineItem`-től örökölt `paint`, `boundingRect` és `shape` metódusokat.

3.2.10. Gráftervező nézete

A gráftervező nézetét a `MyGraphicsView` osztály adja, amely a `QGraphicsView`-ból származtatott megjelenítő.

Adattagok

Adattagként felvettettem a megjelenítéshez használt `QGraphicsScene` típusú `_Scene`-t, amire a tényleges rajzolás történik. Megjelenik az előző alfejezetben bemutatott grafikus pontokat és éleket tartalmazó két lista (`_Vertices` és `_Edges`), a kijelölt terület láthatóságát biztosító téglalap `_SelectionRect` és az annak kirajzolásához

használt grafikus téglalap (`_SelectionRectGItem`). A 3.2.5 alfejezetben leírt gombfelületen benyomott gombtól függően, lehetséges, hogy nem lehetséges terület kijelölése, ezt az információt a `_Selecting` adattag tárolja. A többszereplős A* algoritmus lefutása közben elküldött üzeneteket az `_EndMessage` tartalmazza, majd ezt írjuk ki az algoritmus végén. Megjegyezzük a pontok méretét (`_VSize`) és az élek szélességét (`_EWidth`), és az elemek létrehozásakor ezeket is átadjuk a konstruktorunknak.

Metódusok

A konstruktor beállítja a jelenetet (`_Scene`), és néhány adattagnak kezdőértéket ad. A `GModel...` kezdetű függvények a modell szignáljait kezelik, és végrehajtják az azoknak megfelelő műveleteket a nézeten, például létrehoznak vagy törölnek elemeket vagy elemek adatait változtatják. A `DeselectAll` minden elemet kijelöletlenné tesz, a `SideBarCheckedChanged` pedig a gombfelület változásait követi. Az egér eseményeit lekövetjük a `mouse...Event` metódusokban. A `mousePress-` és `mouseMoveEvent`-ben a kijelölő téglalapot állítjuk be, ha kell. A `mouseReleaseEvent` a tényleges kijelölést hajtjuk végre és a kijelölt elemekről szignált küldünk a modellnek.

MyGraphicsView	
-	<code>_Edges: QList<EdgeGraphicsItem*></code>
-	<code>_EndMessage: QString</code>
-	<code>_EWidth: int</code>
-	<code>_Scene: QGraphicsScene*</code>
-	<code>_Selecting: bool</code>
-	<code>_SelectionRect: QRectF</code>
-	<code>_SelectionRectGItem: QGraphicsRectItem*</code>
-	<code>_Verticles: QList<VerticleGraphicsItem*></code>
-	<code>_VSize: int</code>
+ DeselectAll(): void	
+ EdgeSelected(int): void	
+ GModelAlgoDone(int, int, int, int): void	
+ GModelAlgoEdgeOnPath(int): void	
+ GModelAlgoPartDone(int, int, int, int, int): void	
+ GModelAlgoReset(): void	
+ GModelAlgoVerticleClosed(int): void	
+ GModelAlgoVerticleExpanding(int): void	
+ GModelAlgoVerticleOnPath(int): void	
+ GModelAlgoVerticleOpened(int): void	
+ GModelEdgeAdded(int, QPoint, QPoint, bool, int): void	
+ GModelEdgeRemoved(int): void	
+ GModelEdgeUpdated(int, QPair<QPoint,QPoint>, bool, int): void	
+ GModelEWidthChanged(int): void	
+ GModelHideShowEdgeCosts(): void	
+ GModelObstacleAdded(int): void	
+ GModelObstacleRemoved(int): void	
+ GModelUnitAdded(int, int, int): void	
+ GModelUnitRemoved(int, int): void	
+ GModelVerticleAdded(int, QPoint, bool): void	
+ GModelVerticleMoved(int, QPoint): void	
+ GModelVerticleRemoved(int): void	
+ GModelVSizeChanged(int): void	
- mouseMoveEvent(QMouseEvent*): void	
+ MousePressed(QPointF): void	
- mousePressEvent(QMouseEvent*): void	
+ MouseReleased(QPointF): void	
- mouseReleaseEvent(QMouseEvent*): void	
+ MyGraphicsView(QWidget*)	
+ ~MyGraphicsView()	
+ SetEWidth(int): void	
+ SetVSize(int): void	
+ SideBarCheckedChanged(Role): void	
+ VerticleSelected(int): void	

3.2.11. Szimulátor nézete által használt osztályok

A szimulátor nézete is használja a 3.2.9 alfejezetben bemutatott pont- és élosztályokat, de emellett a csak szimulátorban megjelenő egységekhez és akadályokhoz külön osztályokat definiáltunk (`UnitGraphicsItem` és `ObstacleGraphicsItem`).

<code>QGraphicsEllipseItem</code>	<code>QGraphicsRectItem</code>
<code>UnitGraphicsItem</code>	<code>ObstacleGraphicsItem</code>
<pre> - _ActAim: GraphVerticle* - _ActVert: GraphVerticle* - _AimList: QList<GraphVerticle*> + _Algo: GraphSearchAlgorithm* - _Evaded: UnitGraphicsItem* - _Followed: UnitGraphicsItem* - _Graph: GraphModel* - _Id: int - _Intercepted: UnitGraphicsItem* - _LastAim: GraphVerticle* - _NextAimBlocked: bool - _PathVisible: bool - _PatrolAimIndex: int - _Patrolling: bool - _PatrolList: QList<GraphVerticle*> - _Selected: bool - _Size: int - _Speed: int - _State: UState - _Type: VType + advance(int): void + boundingRect(): QRectF {query} + GetAim(): GraphVerticle* + GetAimList(): QList<GraphVerticle*>& + GetEvaded(): UnitGraphicsItem* + GetFollowed(): UnitGraphicsItem* + GetId(): int + GetIntercepted(): UnitGraphicsItem* + GetPatrolList(): QList<GraphVerticle*>& </pre>	<pre> + GetState(): UState + GetVert(): GraphVerticle* + IsPatrolling(): bool + paint(QPainter*, QStyleOptionGraphicsItem*, QWidget*): void + SetAim(GraphVerticle*&): void + SetEvaded(UnitGraphicsItem*): void + SetFollowed(UnitGraphicsItem*): void + SetIntercepted(UnitGraphicsItem*): void + SetPathVisibility(bool): void + SetPatrolling(bool): void + SetSelect(bool): void + SetSpeed(int): void + SetState(UState): void + SetVert(GraphVerticle*): void + SetType(VType): void + shape(): QPainterPath {query} + UnitGraphicsItem(int, QPoint, int, GraphModel*&, QGraphicsItem*) + UnitGraphicsItem() + ~UnitGraphicsItem() - _Id: int + boundingRect(): QRectF {query} + GetId(): int + ObstacleGraphicsItem(int, QRectF, QGraphicsItem*) + ~ObstacleGraphicsItem() + paint(QPainter*, QStyleOptionGraphicsItem*, QWidget*): void + shape(): QPainterPath {query} </pre>

Akadályok

A `ObstacleGraphicsItem` osztály a `QGraphicsRectItem`-ból származik, új adattagja csak az azonosító (`_Id`), a rajzoláshoz szükséges `paint`, `boundingRect` és `shape` metódusok itt is felüldefiniáltak.

Egységek

Az egységek osztálya (`UnitGraphicsItem`) összetettebb, mivel a mozgáshoz sok információra van szükség. Az `_Id`, `_Type`, `_Size` és `_Selected` adattagok funkciója a pontoknál leírtakkal megegyező és azokhoz hasonlóan őse a `QGraphicsEllipseItem` osztály. Ennek megfelelően a `paint`, `boundingRect` és `shape` metódusok is hasonlóan lettek megoldva, így azokat nem részletezem. A konstruktor az osztály adattagjainak ad kezdőértéket.

A felhasznált adattagok az egység sebessége (`_Speed`), állapota (`_State`), az aktuális gráfcsúcs ahol tartózkodik (`_ActVert`), a megtalált úton következő célja

(`_ActAim`), a végcélja (`_LastAim`) és a megtalált út listaként (`_AimList`). Különleges mozgásformákhoz tároljuk a követett (`_Followed`), megelőzött (`_Intercepted`) vagy elkerült (`_Evaded`) egységet, illetve járőrzés esetén a járőrpontok listáját (`_PatrolList`), a következő pont indexét (`_PatrolAimIndex`), hogy a következő célpont blokkolt-e (`_NextAimBlocked`), és hogy az adott egység éppen járőröz-e (`_Patrolling`). Ezen adattagokhoz felvettettem lekérdező és beállító függvényeket.

Azért, hogy az egység a többitől függetlenül tudjon majd utat keresni, adattagként átadjuk neki a területre fektetett gráfot és az átadott gráffal létrehozunk neki egy saját keresőalgoritmust (3.2.8 szakasz). Miután egy egységnek kijelöltük a célpontját (ez a szimulátor nézetében történik), az ezzel az algoritmussal, pontosabban Manhattan heurisztikát használó A* kereséssel keres oda vezető utat. Ehhez felhelyez a modellgráfra egy egységet saját azonosítójával és a megfelelő kezdő és célpontokkal. Az előbbieket és minden hozzájuk kapcsolódó szükséges ellenőrzést a `QGraphicsEllipseItem` eredetileg üres törzsű virtuális `advance` metódusának felüldefiniált változatában oldunk meg, ami tulajdonképpen az egység mozgásáért és annak előkészítéséért felel. A `_Scene` `advance` metódusa egymás után kétszer hívja a rajta levő elemek `advance` függvényét, először 0, majd 1 paraméterrel (fázis).

Az 1-es fázisban a tényleges mozgatás történik, itt ezt az aktuális pozíció (`pos`) és az aktuális cél helyét összekötő szakaszon való elmozdítással oldjuk meg, ha ez egység állapota `UMoving` vagy `URecalculate`.

A 0-s fázisban a mozgás előkészítése történik, ami jelenti például az útkeresést és a célok beállítását. Ha egy egységnek egyedül jelöltünk ki célpontot, azaz egyedül mozog, akkor keres egy utat a céljához az akadályokat és az álló egységeket kikerülve, és elindul az út mentén. Annak érdekében, hogy két egység ne lépjen egyszerre egy gráfpontra minden egység mielőtt elindulna az aktuális csúcsról (`_ActVert`), lefoglalja magának az aktuális célját, azaz beállítja annak `_Reserver` adattagját a saját azonosítójára, ha azt még nem foglalja más (azaz a `_Reserver` értéke -1).

Ha a foglalás sikeres, az aktuális csúcsát a `_Reserver` -1-re állításával elengedi, majd elindul, azaz `UMoving` állapotba vált. Ha a foglalás sikertelen, akkor az egység várakozik, azaz `UWaiting` állapotba vált és az aktuális csúcsot akadályá teszi. Ha egy egység elérte végcélját, akkor leáll, azaz `UStopped` állapotba vált és az aktuális csúcsot akadályá teszi. Ennek eredményeként a várakozó vagy álló egységet a többi kikerüli. Megtörténhet, hogy az aktuális célcímszöveg blokkolttá válik, ha valaki rajta vár vagy ráállt. Ekkor az egység újratervez, azaz új utat keres a végcélhoz, ami már ki fogja kerülni az előbbi csúcsot. Ha nem talál utat a végcélhoz, akkor leáll.

Úttervezés csak gráfcsúcsokon történik, tehát két csúcs közötti mozgáskor nem. Ha út közben küldjük új végcélhoz az egységet, akkor az `URecalculate` állapotba

vált és az aktuális céljának elérése után tervez újra. `UStopped` állapotban a különleges mozgásformát nem végző egység egyszerűen visszatér az `advance` metódusból.

Járőrzés esetén egy járőrpont elérésekor az aktuális végcélt az egység a következő járőrpontra állítja és utat keres. Ha a következő járőrpont nem elérhető, akkor az az utáni járőrpont lesz a végcél és így tovább. Ha az éppen elfoglalt járőrponton kívül egyik járőrpont sem elérhető, vagy menet közben minden járőrpont elérhetetlenné válik, akkor abba hagyja a járőrzést.

Követéskor a követett egység aktuális csúcsa lesz a végcél és a követő folyamatosan figyeli annak változását. Ha nem talál a célponthoz vezető utat, akkor abba hagyja a követést. Ha az aktuális célja akadályá válik, akkor félbehagyja a követést, amíg a követett nem mozdul el.

Megelőzéskor a megelőzött aktuális célja lesz a megelőző végcélja, ha az létezik. Egyéb tekintetben és, ha a megelőzöttnek nincs aktuális célja a megelőzés a követéssel azonos.

Elkerüléskor az elkerülő a saját közvetlen környezetében keres az elkerülttől minél távolabbi célpontot.

Egy egység egyszerre egy különleges tevékenységet végezhet, és azok csak egy egységre irányulhatnak (nem lehet egyszerre több egységet követni).

Megjegyzendő, hogy a leírt módszer nem akadályoz meg mindenfajta ütközést, például különböző, egymást keresztező éleken áthaladó egységek ütközhetnek, de egy pontra nem futnak egyszerre és egy élen sem haladnak át egyszerre egymással szemben.

3.2.12. Szimulátor nézete

A szimulátor nézete nem felel meg teljes mértékben a nézet elnevezésnek, hiszen a szimulátorfelületen adatkezelést és számolásokat is végez.

Adattagok

Megjelennek és hasonló feladatot látnak el, mint a gráftervező esetében, a `_Scene`, `_SelectionRect`, `_SelectionRectGItem`, `_Selecting`, `_Checked`, `_VSize`, `_EWidth`, `_Verticles` és `_Edges` adattagok ([3.2.10](#) szakasz). A szimulátor egységeit a `_Units`, akadályait az `_Obstacles` lista tárolja. Az azonosítók egyediségét biztosítandó, itt is felvettem gráfnál is használtakhoz hasonló `_MaxObstId` és `_MaxUnitId` számlálókat. Tárolom az egységek méretét (`_USize`), az akadályok generáláskor beállított maximális méretét (`_ObstSize`), az egységekre vonatkozó a gombfelületen beállított speciális mozgásforma nevét (`_Spec`) és hogy az egységek megtervezett útja látható-e

éppen (`_UnitPathVisible`).

A szerkesztéskor a `_UnitToMoveId` adattagot az egységek áthelyezésekor használjuk, az új kézzel lehelyezett akadály helyének kirajzolásához pedig a `_NewObstRect`-et. Tárolom az éppen kijelölt egységek listáját `_SelectedUnits` és felvettetem egy időzítőt (`_Timer`), amely a `_Scene` mozgást előidéző `advance` metódusát váltja ki. A lefektetett gráf a `_Graph` adattagban kerül megőrzésre.

Metódusok

A konstruktorban megadjuk a `_Scene` és a `_Timer` beállításait, kezdőértéket adunk bizonyos adattagoknak és létrehozzuk a lefektetett gráfot annak `GenerateForSim` függvényvel, és bizonyos szignáljait kezelőhöz kötjük. A mentést a `Save` végzi lementve az egységek és akadályok elhelyezkedését és azonosítóját (de mozgásinformációkat nem). A betöltés a `Load` meghívásával történik, ami visszaolvasa a mentett adatokat és az alapján a legenerált üres felületre helyezi a mentett helyre az elemeket. A véletlen szimulátor generálásáért a `Generate` felel, ami beállít bizonyos kezdőértékeket és létrehozza az akadályokat és egységeket.

Az egységek (`Unit`) és akadályok (`Obstacle`) létrehozását, vászonra helyezését és törlését és az ezekhez szükséges ellenőrzéket az értelemszerűen megfelelő végződésű `Add...`, `Remove...` és `RemoveAll...` metódusok végzik, illetve a `ClearAll` minden töröl. A lefekte-

MySimGraphicsView	
<i>QGraphicsView</i>	
- <code>_Checked: Role</code>	
- <code>_Edges: QList<EdgeGraphicsItem*></code>	
- <code>_EWidth: int</code>	
- <code>_Graph: GraphModel*</code>	
- <code>_MaxObstId: int</code>	
- <code>_MaxUnitId: int</code>	
- <code>_NewObstRect: QRectF</code>	
- <code>_Obstacles: QList<ObstacleGraphicsItem*></code>	
- <code>_ObstSize: int</code>	
- <code>_Scene: QGraphicsScene*</code>	
- <code>_SelectedUnits: QList<UnitGraphicsItem*></code>	
- <code>_Selecting: bool</code>	
- <code>_SelectionRect: QRectF</code>	
- <code>_SelectionRectItem: QGraphicsRectItem*</code>	
- <code>_SimGoing: bool</code>	
- <code>_Spec: QString</code>	
- <code>_Timer: QTimer*</code>	
- <code>_UnitPathVisible: bool</code>	
- <code>_Units: QList<UnitGraphicsItem*></code>	
- <code>_UnitToMoveId: int</code>	
- <code>_USize: int</code>	
- <code>_Vertices: QList<VerticleGraphicsItem*></code>	
- <code>_VSize: int</code>	
- <code>AddObstacle(QRectF, int): bool</code>	
- <code>AddRandomObstacle(): void</code>	
- <code>AddUnit(QPointF, int): bool</code>	
- <code>CheckAim(UnitGraphicsItem*&): void</code>	
- <code>CheckOverlaps(QPointF): bool</code>	
- <code>ClearAll(): void</code>	
- <code>DeselectAll(): void</code>	
+ <code>GAlgoDone(int, int, int, int): void</code>	
+ <code>GAlgoEdgeOnPath(int): void</code>	
+ <code>GAlgoVerticleOnPath(int, int): void</code>	
+ <code>GEdgeAdded(int, QPoint, QPoint, bool, int): void</code>	
+ <code>GEdgeUpdated(int, QPair<QPoint, QPoint>, bool, int): void</code>	
+ <code>generate(int, int, int): void</code>	
+ <code>GVerticleAdded(int, QPoint, bool): void</code>	
- <code>HideOrShowUnitPaths(): void</code>	
+ <code>Load(QString): void</code>	
- <code>mouseMoveEvent(QMouseEvent*): void</code>	
- <code>mousePressEvent(QMouseEvent*): void</code>	
- <code>mouseReleaseEvent(QMouseEvent*): void</code>	
+ <code>MySimGraphicsView(QWidget*)</code>	
+ <code>~MySimGraphicsView()</code>	
- <code>ObstacleSelected(ObstacleGraphicsItem*&): void</code>	
- <code>RemoveAllObstacles(): void</code>	
- <code>RemoveAllUnits(): void</code>	
- <code>RemoveObstacle(ObstacleGraphicsItem*&): void</code>	
- <code>RemoveUnit(UnitGraphicsItem*&): void</code>	
+ <code>Save(QString): void</code>	
- <code>SelectAllUnits(): void</code>	
- <code>SetAim(UnitGraphicsItem*&, QPointF): void</code>	
- <code>SetAims(QPointF): void</code>	
+ <code>SideBarButtonClicked(Role): void</code>	
+ <code>SideBarCheckedChanged(Role): void</code>	
+ <code>SimPaused(): void</code>	
+ <code>SimSpeedChanged(int): void</code>	
+ <code>SimStarted(): void</code>	
+ <code>SpecChanged(QString): void</code>	
- <code>UnitSelected(UnitGraphicsItem*&): void</code>	

tett gráf bonyos eseményeit kezelik a `G...` kezdetű függvények, a gráftervező nézetének `GModel...` függvényeihez hasonlóan. A szimuláció indítását megállítását és sebességének állítását a `Sim...` kezdetű metódusokban oldottuk meg. Ezek a gombfelület megfelelő szignáljaira hívódnak meg, hasonlóan a `SpecChanged` speciális mozgásforma-beállítás változását kezelő metódushoz és a gombnyomások hatását kezelő `SideBarCheckedChanged` és `SideBarButtonClicked`-hez.

Szimuláció során használjuk a minden egységet kijelölő `SelectAllUnits` és az egységek célpontjátmegadó `SetAims` és `SetAim` metódusokat, illetve azt, hogy az útvonalak láthatók-e a `HideOrShowUnitPaths` változtatja. A `SetAims` egyszerre kezeli a kijelölt egységeket és különböző, eymás melletti célpontokat jelöl ki a `SetAim`-t hívva. A `SetAim` beállítja a kezdeti kiinduló és végpontot egy egységhoz és `URecalculate` állapotba helyezi, aminek hatására az egység utat fog keresni. Célpontkijelöléskor és új egységek és akadályok lehelyezésekor is szükség van átfedések ellenőrzésére, amit a `CheckOverlaps` függvény végez.

Itt is kezelnünk kell az egér eseményeit. Ezekben a `_Selecting`, `_Checked` és `_Spec` adattagok aktuális értéke illetve a lenyomott egérgomb alapján hajtunk végre különböző akciókat, például utasítunk egységeket vagy rajzolunk kijelölő téglalapot.

3.3. Tesztelés

Az alkalmazás részfeladatainak tesztelése implementáció során folyamatosan történt, de osztályokhoz vagy metódusokhoz külön tesztek nem íródtak. A tesztelést az adott funkció gyakorlati alkalmazásával és debug-üzenetek kiírásával tettem meg.

3.3.1. Főablak, dialógus ablakok, gombfelületek

A főablakon teszteltem a menüpontok helyes működését, a felületre dinamikusan felvitt vezérlők helyes, megfelelően elhelyezett megjelenését.

A dialógusablakokban a vezérlők helyes megjelenését, feliratait, illetve a beállított korlátok, lépésközök helyességét ellenőriztem és teszteltem megfelelően adódnak-e át a beállított értékek az alsóbb rétegeknek.

A gombfelületeken a gombok és egyéb kezelők elhelyezkedését, feliratainak és hatásuknak helyességét figyeltem.

3.3.2. Gráfmodell

A gráfmodell tagfüggvényeinek hibamentességét az implementációkor beiktatott debug-üzenetekkel ellenőriztem. Figyeltem az adattagok értékét, a mutatók beállí-

tását és a használt listák tartalmát.

A tagfüggvények működését és az ott elvégzendő ellenőrzések helyes kiértékelődését az adott metódus alkalmazásával teszteltem. Pontokat, éleket, egységeket, akadályokat helyeztem a megjelenítőre érvényes és érvénytelen területekre vagy töröltem a gráfból, és közben figyelem a kiírt debug-üzeneteket, illetve az akcióm tényleges hatását. Itt teszteltem a gombfelület gombjainak és az algoritmus lépéseinak tényleges gráfra vett hatását és a hozzájuk tartozó kezelőfüggvények helyes működését a várt változásokhoz tartozó információk kiírásával és a nézeten megjelenő változások megfigyelésével.

3.3.3. Algoritmusmodell

Az algoritmusok futásának helyességét a gráfnak küldött szignálok következményeként a nézeten történt változások alapján tudtam megítélni. A hibamentesség és helyes grafelemkezelés ellenőrzésére különböző speciális gráfokon, vagy speciális akciók után teszteltem az algoritmust. Például egységek nélküli gráfon, üres gráfon, vagy adott egységre való futtatás után az adott egységet törölve és újrafuttatva az algoritmust.

Algoritmusok eredményessége

Az alábbiak jól megfigyelhetők a különböző algoritmusok futtatása során.

A Dijkstra algoritmus, ha az élsúlyok a tényleges pontok közötti távolságot tükrözik (hitelesek), akkor tulajdonképpen egy szélességi keresést hajt végre, tehát nagyon nagy mennyiségű csúcsot megvizsgál. Ugyanakkor véletlenre állított élsúlyok esetén is legrövidebb utat talál a kezdő és célcíms között.

Az A* algoritmus hiteles élsúlyok mellett, euklideszi távolság heurisztikával, minden rács minden véletlen gráfon legrövidebb utat talál hiszen ez esetben ez alulbecslő heurisztika, ráadásul jóval kevesebb csúcsot látogat meg, mint a Dijkstra. Manhattan heurisztikával látogatja meg a legkevesebb csúcsot, így az a leggyorsabb, de hiteles élsúlyok mellett sem garantál legrövidebb utat. A két pontszámoló heurisztika csak nagyon speciális esetekben működik igazán hatékonyan és szintén nem garantál legrövidebb utat. Véletlen élsúlyok mellett nyilvánvalóan már egyik A* változat sem feltétlenül ad legrövidebb utat, de sebességük a heurisztikus értékek beszámítása miatt továbbra is gyorsabb, mint a Dijkstra algoritmusé.

Ugyanezek mondhatók el a megvalósított többszereplős A* algoritmusról is, hiszen egységenként megegyezik az A*-al. Az algoritmusok a legnagyobb (4900 pontú) generálható gráfon több különböző akadálymennyiséggel -elhelyezés mellett is az

elvártnak megfelelően futottak, de a többszereplős keresés százas nagyságrendű egységszám mellett, ekkora gráfon már aránylag lassú.

3.3.4. Gráftervező és szimulátor nézete

A nézeteken a modellnél is szükséges adatérték, mutató és listatartalom-ellenőrzésen felül a megjelenítés helyességét ellenőriztem. Tükrözi-e a gráfon, szimulátoron történt változásokat, megfelelő-e az elemek színezése, szövegkiírása, működik-e a kijelölés minden esetben és hatása megfelel-e az elvártnak.

3.3.5. Egységek mozgása

Az egységek mozgása a sok különböző lehetséges mozgásforma, az egységek együttes mozgása és a mozgás közbeni utasításkiadás lehetősége miatt rengeteg hibaforrást rejti.

Sok problémát okoztak be nem állított mutatók, ezeket hozzáadott mutatóérték-ellenőrzésekkel küszöböltem ki.

Bizonyos esetekben az egységek beakadtak és folyamatosan újraszámolták az útjukat, ami akadozó futást eredményezett. Ezeket legtöbb esetben az adott egység leállításával kerülöm el, de még így is előfordulnak esetek, amikor egy hosszabb újraszámolás megakasztja a futást, például, ha az egész terület üres, csak az egység célja válik foglalttá, akkor a használt A* keresés is kifejt minden pontot a térképen. Ezen segítene a program többszálúsítása, de az nem került megvalósításra.

A speciális mozgásformát végző egységekkel is adódtak problémák attól függően, hogy céljuk elérhető volt vagy nem, vagy menet közben vált elérhetetlenné, vagy éppen adott mozgásforma közben adtam másik utasítást az egységnak. A haladás közbeni új célpont kijelölése előidézett lefoglaltan maradt csúcsokat. Ezeket az adott speciális eset figyelembe vételevel, és a csúcsfelengedés szigorúbb biztosításával hárítottam el. A speciális mozgások következtében előforduló folyamatos újraszámolást az adott mozgásforma befejezésével oldottam meg bizonyos esetekben.

Az ütközések elkerülésének tesztelése nagy mennyiségű (50-70) egység együttes mozgatásával történt, különböző mozgásformák beiktatásával, lehetőleg minél több ütközési lehetőséget előidézve. Egymást keresztező éleken, egy csúcsba futó, egymás melletti éleken való ütközés és egységek sebessékgülönbségéből adódó ütközések továbbra is előfordulhatnak, de az egyszerre azonos csúcsra érkezésből és az egy élen szemben áthaladásból adódó ütközésekkel nagyon jó arányban elkerülik az egységek.

4. fejezet

Összefoglalás és végszó

Összegzésként elmondható, hogy az előzetes terveket jól tükröző, az elképzelt feladatot teljesítő program készült a szakdolgozat keretei között.

Alkalmas a keresőalgoritmusok futásának bemutatására, illetve kis változtatásokkal megvalósíthatók hozzá újabb algoritmusok. A felhasznált gráf kellő mértékben szerkeszthető, bár pár ezerné több pontú gráf nem készíthető, de ez nem is célja az alkalmazásnak. A lefuttatott keresések megfigyelésével könnyebben érhetővé válik a keresőalgoritmusok működése azok számára is, akik azelőtt küszködtek azok megértésével.

Az egység- és csapatmozgás-algoritmusok egyfajta megvalósítása a gráfon kereső algoritmus felhasználásával készült el, és egy testre szabható felületen bemutatható. A csapatmozgáshoz az egyik legkorábbi többszereplős útkeresési megoldás, az újratervező A* keresés egy változatát implementáltam, aminek köszönhetően szembesülhettem a játékbeli útkeresés megvalósításának nehézségeivel, és hasznos tapasztalatokkal gazdagodtam a témaörrel kapcsolatban. Az irodalomjegyzékben találhatók többszereplős útkereséssel foglalkozó cikkek [6, 7, 8, 9], amiket ajánlok az érdeklődők figyelmébe.

A dolgozat elkészítését megelőzte két játékokban használt mesterséges intelligenciával foglalkozó könyv témaiba illő fejezeteinek tanulmányozása [4, 5]. Az alkalmazás fejlesztése során az eddig ismeretlen módszerek, ötletek és lehetőségek megtalálásában segítettek a stackoverflow [3] és a Qt project [2] fórumainak hozzászólói, és a Qt project dokumentációja [1]

Irodalomjegyzék

- [1] V. Authors. Qt project documentation, <http://qt-project.org/doc/>, 2014-05-11, 2014. [45](#)
- [2] V. Authors. Qt project forum, <http://qt-project.org/forums>, 2014-05-11, 2014. [45](#)
- [3] V. Authors. Stackoverflow, <http://stackoverflow.com/>, 2014-05-11, 2014. [45](#)
- [4] D. M. Bourg and G. Seemann. *AI for game developers*, volume 400. O'Reilly, 2004. [45](#)
- [5] I. Millington. *Artificial Intelligence for Games*. CRC Press, 2006. [45](#)
- [6] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Meta-agent conflict-based search for optimal multi-agent path finding. In *SOCS*, 2012. [45](#)
- [7] G. Sharon, R. Stern, M. Goldenberg, and A. Felner. The increasing cost tree search for optimal multi-agent pathfinding. 2011. [45](#)
- [8] D. Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005. [45](#)
- [9] T. S. Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI*, 2010. [45](#)