

## Tartalomjegyzék

Felhasználói dokumentáció.....	3
A feladat megfogalmazása.....	3
A felhasznált java technikák.....	3
A program futtatása.....	3
A program felhasználói felülete.....	4
A menürendszer használata.....	4
A Fájl menü.....	4
A Fájl menü Új menüpontja.....	5
Fájl menü Betöltés menüpontja.....	7
Fájl menü Mentés menüpontja.....	7
Fájl menü Bezárás menüpontja.....	7
Fájl menü Kilépés menüpontja.....	7
A Hálózat menü.....	8
A Hálózat menü Futtatás menüje.....	8
A Futtatás menü HS menü Félduplex menüpontja.....	8
A Futtatás menü HS menü Duplex menüpontja.....	8
A Futtatás menü Időszelet menüpontja.....	8
A Futtatás menü LCR menüpontja.....	8
A Futtatás menü Maxterjed menüpontja.....	8
A Futtatás menü Optmaxterjed menüpontja.....	8
A Hálózat menü Összes elemzése menüpontja.....	9
A Hálózat menü Szünet menüpontja.....	9
A Hálózat menü Folytatás menüpontja.....	9
A Hálózat menü Leállítás menüpontja.....	9
A Nézet menü.....	10
A Beállítások menü.....	11
A Beállítások menü Elemzés beállításai menüpontja.....	11
A Beállítások menü Kitüntetett folyamat menüpontja.....	12
A Beállítások menü Várakozási idő menüpontja.....	12
A Súgó menü.....	13
A Súgó menü Nyelv menüje.....	13
A Súgó menü Nyelv menü Magyar menüpontja.....	13
A Súgó menü Nyelv menü Magyar menüpontja.....	13
A Súgó menü Névjegy menüpontja.....	13
Gyorsbillentyűk táblázata.....	14
Fejlesztői dokumentáció.....	15
A feladat megfogalmazása.....	15
A felhasznált technikák.....	17
Osztály diagramm.....	19
Uzenet csomag.....	20
Uzenet osztály.....	20
Folyamat csomag.....	20

Folyamat osztály .....	20
<b>Halozat csomag</b> .....	22
Halozat osztály .....	22
<b>Ablak csomag</b> .....	26
Foablak osztály .....	26
Ujhalozat osztály .....	28
Ujhalozatelek osztály .....	29
Elemzes1 osztály .....	29
Elemzes2 osztály .....	30
Rajz osztály .....	30
XMLszuro osztály .....	31
<b>Feliratok csomag</b> .....	31
SajatResourceBundle osztály .....	31
Feliratok_hu osztály .....	32
Feliratok_en osztály .....	32
<b>Kivetelek csomag</b> .....	32
Elnemletezofolyamathoz osztály .....	32
Hurokel osztály .....	32
Legalabb2folyamat osztály .....	32
Marletezofolyamat osztály .....	33
Nemosszefuggoaltalanos osztály .....	33
Nemosszefuggogyuru osztály .....	33
Pozitivegeszek osztály .....	33
<b>Szimulacio csomag</b> .....	33
Szimulacio osztály .....	33
<b>Tesztelés:</b> .....	34
A Futtatás menü HS menü Félduplex menüpontja .....	34
A Futtatás menü HS menü Duplex menüpontja .....	38
A Futtatás menü Időszelet menüpontja .....	42
A Futtatás menü LCR menüpontja .....	46
A Futtatás menü Maxterjed menüpontja .....	50
A Futtatás menü Optmaxterjed menüpontja .....	56
A Hálózat menü Összes elemzése menüpontja .....	62

# Felhasználói dokumentáció

## A feladat megfogalmazása

Vezető választó algoritmusok szimulációja. A Le Lann, Chang és Roberts tiszteletére elnevezett LCR algoritmus, a Hirschberg—Sinclair algoritmus, az Időszelet algoritmus, a MaxTerjed és az OptMaxTerjed algoritmus ezek. A hálózat folyamatokból áll. A folyamatok által küldött üzeneteket kell kiírni és összeszámolni. A folyamatok azonosítói a felhasználó által is megadhatóak, vagy a gép által véletlenszerűen is kioszthatóak. Az üzenetek karakteres és grafikus megjelenítésére is van lehetőség. Létre lehet hozni új hálózatot. A hálózatot el lehet menteni, és az elmentett hálózatot vissza is lehet tölteni. A program kényelmes használatát grafikus felhasználói felület biztosítja.

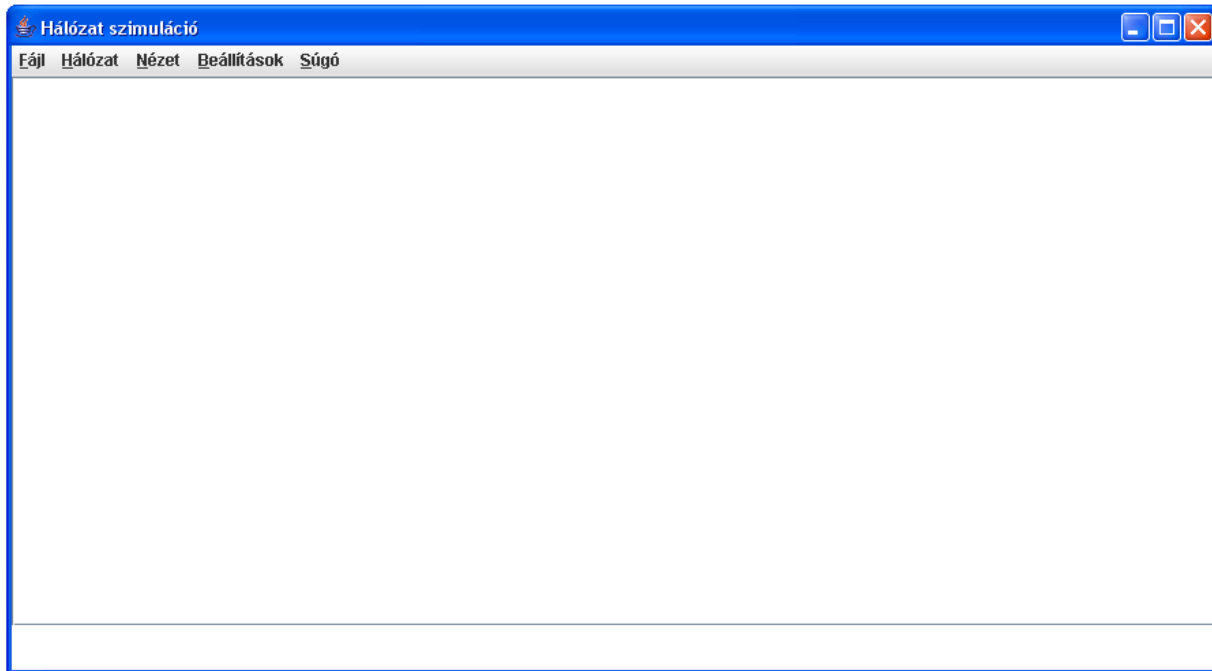
## A felhasznált java technikák

A programot java nyelven írtam. A felhasználói felület és a grafikus megjelenítés a java.awt, java.awt.geom és javax.swing csomagokkal történik. A hálózatok tárolása xml fájlban lehetséges. Az xml műveletek megvalósítása a javax.xml.parsers, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.stream, org.xml.sax. és az org.w3c.dom. csomagokkal történik. A fájl műveletek a java.io csomag használatával történik. Az adatok tárolására és egyéb feladatok a java.util csomag használatával valósítottam meg.

## A program futtatása

A futtatásához java 1.5 vagy annál újabb változat kell. A program futtatásához javasolt beállítani a Path-ban a javat tartalmazó könyvtár aktuális verziószámú jdk könyvtárának bin alkönyvtárát a benne lévő állományok eléréséhez. Ha ennek a könyvtárnak az állományai futtathatóak bármely más könyvtárból is, akkor a program indítása a „Szimulacio/bin” könyvtárban kiadott „java szimulacio/Szimulacio” paranccsal lehetséges.

## A program felhasználói felülete

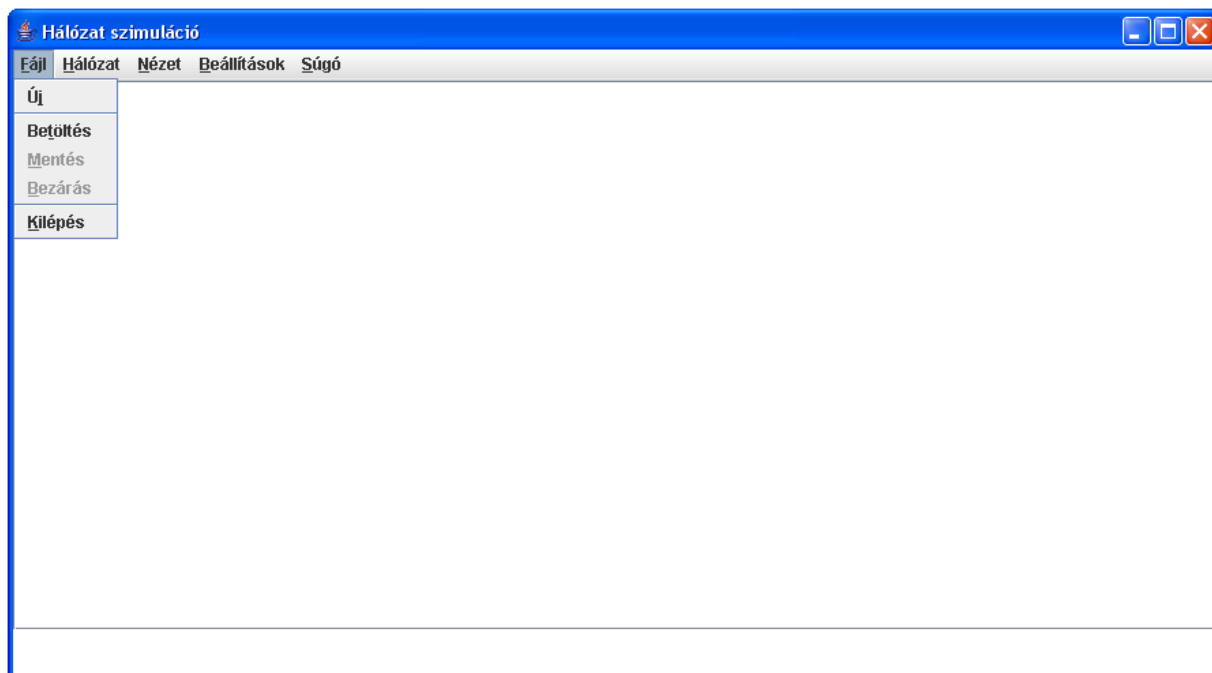


### A menürendszer használata

#### A Fájl menü

A Fájl menü gyorsbillentyűje: Alt + f

Az adatokkal kapcsolatos műveleteket lehet ebben a menüben elvégezni.




## A Fájlmű Új menüpontja

Az Új menüpont gyorsbillentyűje: Alt + j

Ezzel a menüponttal lehet új hálózatot létrehozni. Választhatunk, hogy gyűrűt, vagy általános hálózatot kívánunk létrehozni. A hálózat nevét kötelező megadni. A hálózat mérete és a folyamatok azonosítói közül legalább az egyiket meg kell adni. Ha mind a kettő meg van adva, akkor a méretnek és a folyamat azonosítók számának meg kell egyeznie. A hálózat neve szöveg karakterekből állhat. A hálózat méretének egynél nagyobb természetes számnak kell lennie. A folyamatok azonosítóinak természetes számoknak kell lenniük, egymástól veszővel elválasztva. Ha a folyamatok azonosítóit nem adjuk meg, akkor véletlenszerűen kapnak azonosítókat a folyamatok. Ha gyűrűt akarunk létrehozni, akkor a Rendben gombbal létrehozzuk az új hálózatot, ha az adatokat helyesen adtuk meg. Ha általános hálózatot akarunk létrehozni és megadtuk a folyamatok azonosítóit, akkor egy új párbeszéd ablak nyílik meg, ha helyes adatokat adtuk meg. Az új párbeszéd ablakban a folyamatok kimenő szomszédjait. A folyamatok azonosítóját kell megadni vesszővel elválasztva. Ha volt létrehozva, vagy betöltve hálózat, akkor a régi automatikusan bezáródik. Ha a megadott adatokban hiba volt, akkor hibaüzenetet kapunk. A Mégsem gombbal bezárjuk az ablakot, és nem hozunk létre új hálózatot.

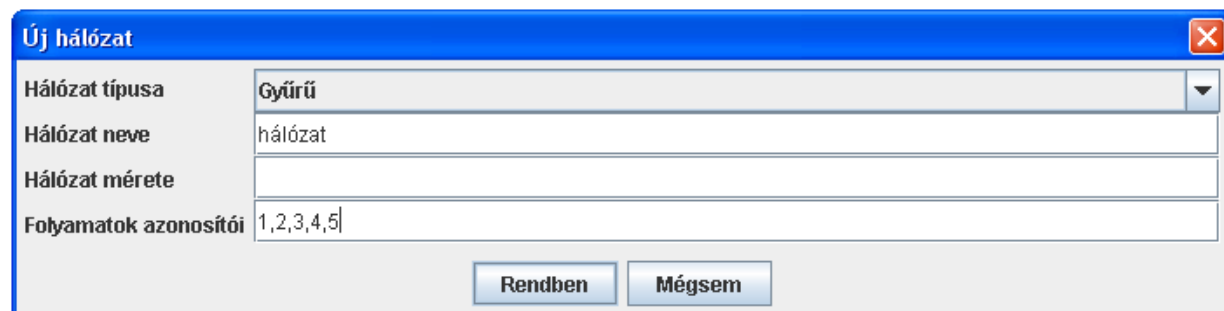
Példák új hálózat helyes megadásai:



Új hálózat

Hálózat típusa	Gyűrű
Hálózat neve	hálózat
Hálózat mérete	5
Folyamatok azonosítói	

Rendben Mégsem



Új hálózat

Hálózat típusa	Gyűrű
Hálózat neve	hálózat
Hálózat mérete	
Folyamatok azonosítói	1,2,3,4,5

Rendben Mégsem



Új hálózat

Hálózat típusa	Gyűrű
Hálózat neve	hálózat
Hálózat mérete	5
Folyamatok azonosítói	1,2,3,4,5

Rendben Mégsem

**Új hálózat** ✖

Hálózat típusa	Általános
Hálózat neve	hálózat
Hálózat mérete	5
Folyamatok azonosítói	1,2,3,4,5

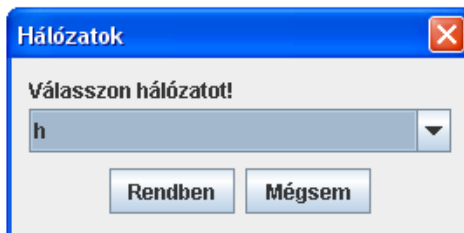
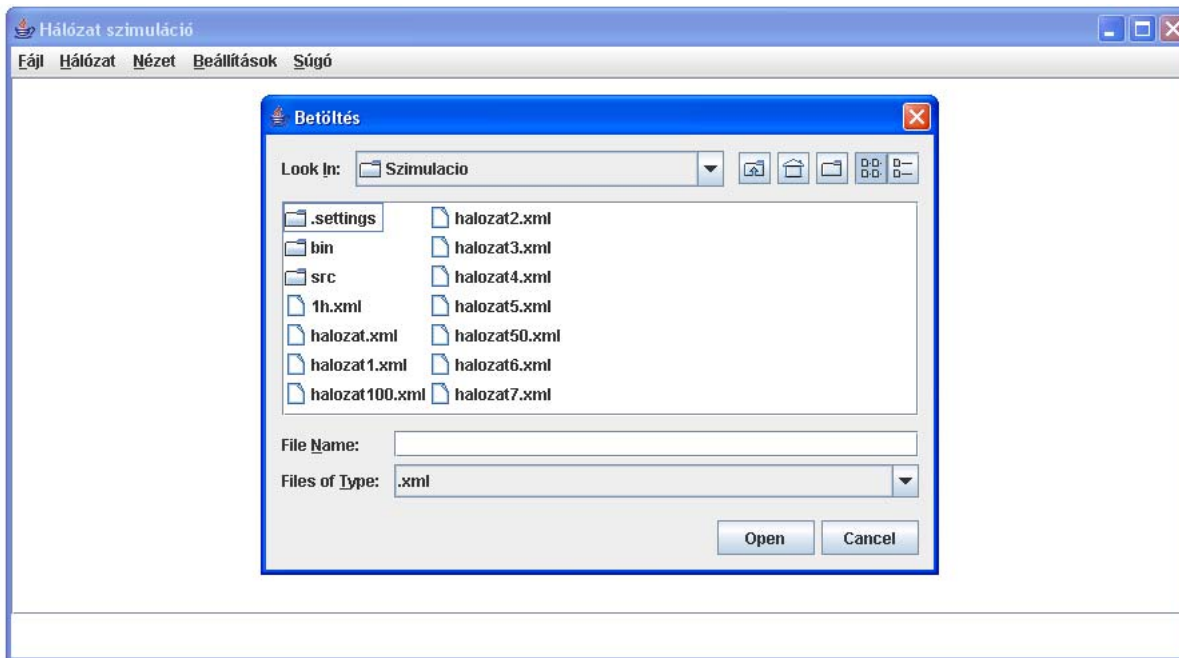
**Kimenő élek beállítása** ✖

A(z) 1 azonosítójú folyamatból kimenő élek	2,3
A(z) 2 azonosítójú folyamatból kimenő élek	3,4
A(z) 3 azonosítójú folyamatból kimenő élek	1,2,5
A(z) 4 azonosítójú folyamatból kimenő élek	2,5
A(z) 5 azonosítójú folyamatból kimenő élek	1,2,4

## Fájl menü Betöltés menüpontja

A Betöltés menüpont gyorsbillentyűje: Alt + t

XML fájlból lehet betölteni hálózatot. Először ki kell választani a fájlt. Ha érvényes XML állományt választottunk, akkor ki kell választani a név alapján az állományban tárolt hálózatok közül, hogy melyiket szeretnék használni. Ha volt létrehozva, vagy betöltve hálózat, akkor a régi automatikusan bezáródik.



## Fájl menü Mentés menüpontja

A Mentés menüpont gyorsbillentyűje: Alt + m

A menüpont csak akkor érhető el, ha már létrehoztunk egy új hálózatot, vagy már betöltöttünk egyet. XML fájlba menthetjük el a hálózatot. A Fájl nevét kell megadni. Ha nem írjuk a név után a ".xml" kiterjesztést akkor a program automatikusan megteszi ezt. Ha már létezik olyan nevű állomány, akkor abba próbálja meg elmenteni, mert egy állományban több hálózatot is tárolhatunk. Ha volt már olyan nevű hálózat az állományban, akkor a program felajánlja, hogy felülírja a régebbi hálózatot. Ha a felülírást választjuk, akkor elmenti a hálózatot, különben nem.

## Fájl menü Bezárás menüpontja

A Bezárás menüpont gyorsbillentyűje: Alt + b

A menüpont csak akkor érhető el, ha már létrehoztunk egy új hálózatot, vagy már betöltöttünk egyet. Bezárja a hálózatot. Ezek után nem tudunk műveleteket végrehajtani ezzel a hálózattal.

## Fájl menü Kilépés menüpontja

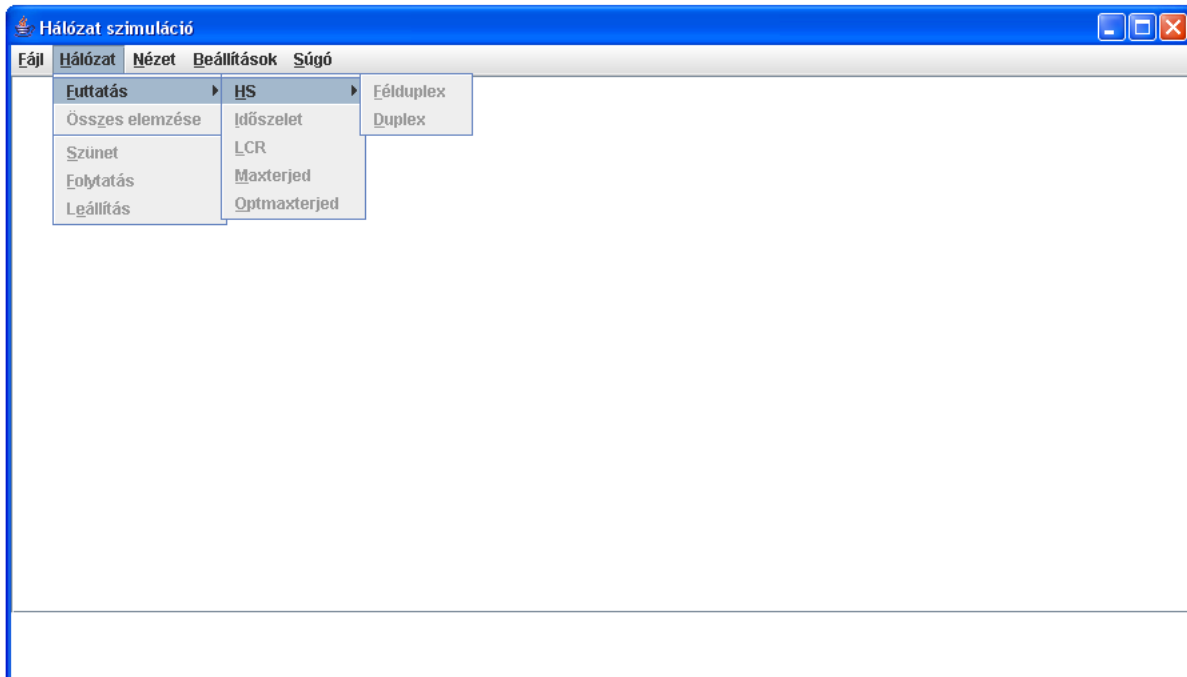
A Kilépés menüpont gyorsbillentyűje: Alt + k

Ezzel a menüponttal léphetünk ki a programból.

## A Hálózat menü

A Hálózat menü gyorsbillentyűje: Alt + h

A hálózat vezető választó szimulációval kapcsolatos műveleteket tudjuk ebben a menüpontban elvégezni.



### A Hálózat menü Futtatás menüje

A Futtatás menü gyorsbillentyűje: Alt + f

Ebben a menüben tudjuk kiválasztani, hogy melyik vezető választó algoritmust szeretnénk futtatni.

### A Futtatás menü HS menü Félduplex menüpontja

A Félduplex menüpont gyorsbillentyűje: Alt + f

A HS algoritmus félduplex változatát tudjuk elindítani.

### A Futtatás menü HS menü Duplex menüpontja

A Duplex menüpont gyorsbillentyűje: Alt + d

A HS algoritmus duplex változatát tudjuk elindítani.

### A Futtatás menü Időszelet menüpontja

Az Időszelet menüpont gyorsbillentyűje: Alt + i

Az Időszelet algoritmust tudjuk elindítani.

### A Futtatás menü LCR menüpontja

Az LCR menüpont gyorsbillentyűje: Alt + l

Az LCR algoritmust tudjuk elindítani.

### A Futtatás menü Maxterjed menüpontja

A Maxterjed menüpont gyorsbillentyűje: Alt + m

A Maxterjed algoritmust tudjuk elindítani.

### A Futtatás menü Optmaxterjed menüpontja

Az Optmaxterjed menüpont gyorsbillentyűje: Alt + o

Az Optmaxterjed algoritmust tudjuk elindítani.



### **A Hálózat menü Összes elemzése menüpontja**

Az Összes elemzése menüpont gyorsbillentyűje: Alt + z

Az összes algoritmust lefuttatja a Beállítások Elemzés beállításában megadott értékszer. Először a megadott hálózaton futnak le az algoritmusok. Utána véletlen hálózatokon futnak le. A Nézet automatikusan Elemzésre állítódik.

### **A Hálózat menü Szünet menüpontja**

A Szünet menüpont gyorsbillentyűje: Alt + s

Ennek a menüpont használatával tudjuk a HS algoritmus futását szüneteltetni.

### **A Hálózat menü Folytatás menüpontja**

A Folytatás menüpont gyorsbillentyűje: Alt + f

Ennek a menüpont használatával tudjuk a Szünet menüponttal leállított HS algoritmust újraindítani.

### **A Hálózat menü Leállítás menüpontja**

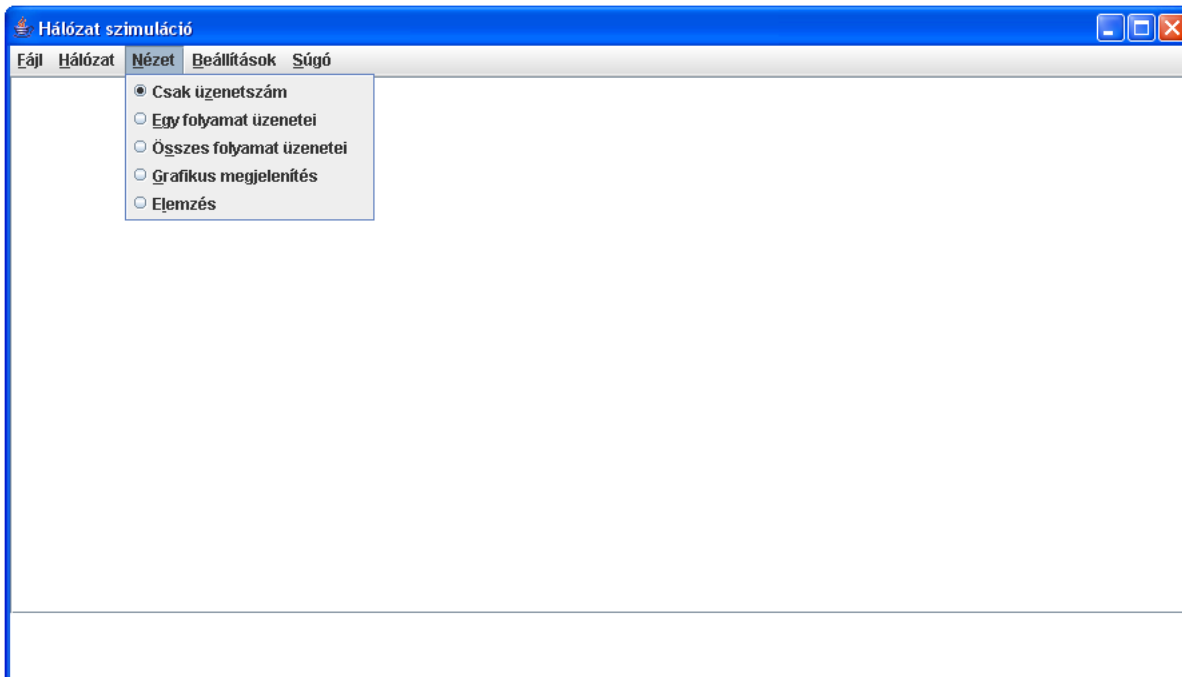
A Leállítás menüpont gyorsbillentyűje: Alt + l

Ennek a menüpont használatával tudjuk a HS algoritmust leállítani. Ez után már nem tudjuk folytatni, csak újra indítani.

## A Nézet menü

A Nézet menü gyorsbillentyűje: Alt + n

A Nézet menüben lehet beállítani, hogy a HS algoritmus a futása során milyen adatokat jelenítsen meg. Alapértelmezett a Csak üzenetszám menüpont.



**Csak üzenetszám** kiválasztása esetén csak az üzenetek számát írja ki a program.

A Csak üzenetszám menüpont gyorsbillentyűje: Alt + z

**Egy folyamat üzenetei** kiválasztása esetén a kitüntetett folyamat üzenetei és az összes üzenetszám kerül kiírásra.

Az Egy folyamat üzenetei menüpont gyorsbillentyűje: Alt + e

**Összes folyamat üzenetei** kiválasztása esetén üzenetváltásonként írja ki az összes folyamat üzeneteit. Az előző lépés üzenetei törölődnek minden lépés előtt.

Az Összes folyamat üzenetei menüpont gyorsbillentyűje: Alt + s

**Grafikus megjelenítés** kiválasztása esetén rajzban jelenik meg a hálózat. Az üzeneteit gyűrűben nyilak jelzik. Általános hálózatban a nyilak hegyeinek végpontjai össze vannak kötve, ha van üzenet azon az élen.

A Grafikus megjelenítés menüpont gyorsbillentyűje: Alt + g

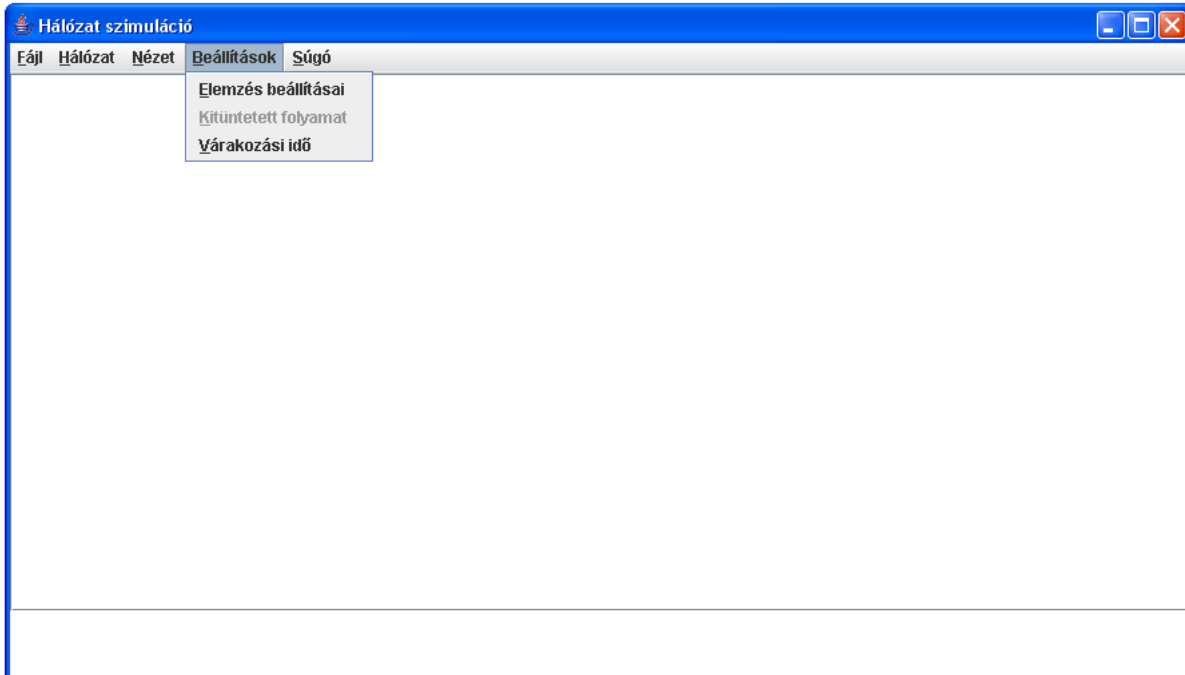
**Elemzés** kiválasztása esetén a *Beállítások Elemzés beállításai*ban megadott darabszor fog lefutni az algoritmus a *Beállítások Elemzés beállításai*ban megadott méretű hálózatok mindegyikén. Ha van olyan méret megadva, mint amilyen az aktuális hálózat mérete, akkor az aktuális hálózaton is le fog futni az algoritmus. Ennek a nézetnek az eredménye a futások minimális, átlagos és maximális lépésszáma és üzenetszáma. Valamint a bonyolultság elméleti felső becsléshez tartozó konstans értékek. Minden algoritmus időbonyolultsága:  $\theta(n)$ . Az LCR, a Maxterjed és az Optmaxterjed algoritmusok üzenetszám bonyolultsága:  $\theta(n^2)$ . A Hirschberg—Sinclair algoritmus üzenetszám bonyolultsága:  $\theta(n \cdot \log n)$ . Az Időszelet algoritmus üzenetszám bonyolultsága  $\theta(n)$ .

Az Elemzés menüpont gyorsbillentyűje: Alt + l

## A Beállítások menü

A Beállítások menü gyorsbillentyűje: Alt + b.

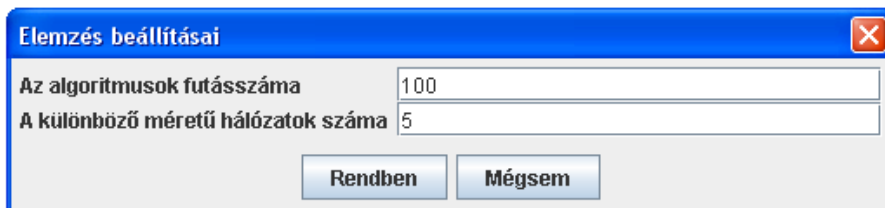
A Beállítások menüben a HS algoritmus futásával kapcsolatos beállításokat lehet elvégezni.



## A Beállítások menü Elemzés beállításai menüpontja

Az Elemzés beállításai menüpont gyorsbillentyűje: Alt + e.

Ennek a menüpont használatával lehet beállítani, hogy a Hálózat menü Összes elemzése menüpont és a Nézet menü Elemzés menüpont kiválasztása esetén az algoritmusok mennyiszor fussanak le. A Nézet menü Elemzés menüpont kiválasztása esetén mennyi különböző méretű hálózatos fusson az algoritmus. A következő párbeszéd ablakban ezeknek a hálózatoknak a méreteit tudjuk megadni.



## A Beállítások menü Kitüntetett folyamat menüpontja

A Kitüntetett folyamat menüpont gyorsbillentyűje: Alt + k.

Ennek a menüpontnak a használatával lehet beállítani, hogy Nézet Egy folyamat üzenetei esetén melyik sorszámú (nem azonosítójú) folyamat üzenetei kerüljenek kiírásra. Az alapértelmezett az 1. Új hálózat létrehozása, vagy betöltése esetén az alapértelmezett kitüntetett csúccsal jön létre. A kitüntetett csúcsot a HS algoritmus futása során is állíthatjuk, az átállítást követően az új kitüntetett csúcs üzenetei kerülnek kiírásra.

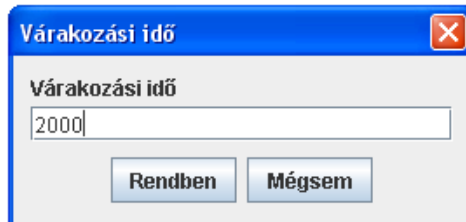


## A Beállítások menü Várakozási idő menüpontja

A Várakozási idő menüpont gyorsbillentyűje: Alt + v.

Ennek a menüpontnak a használatával lehet beállítani, hogy Egy folyamat azonosítói, Összes folyamat azonosítói és Grafikus megjelenítés esetén mennyi idő teljen el két lépés között milliszekundumban megadva. Az alapértelmezett érték 2000 milliszekundum (2 másodperc). Természetes számot vár az ablak bemenő adatként. Ha nem azt adunk meg hibaüzenetet kapunk. A várakozási időt a HS algoritmus futása során is lehet állítani, ekkor a következő lépés után már az új várakozási idő szerinti szünet lesz két lépés között.

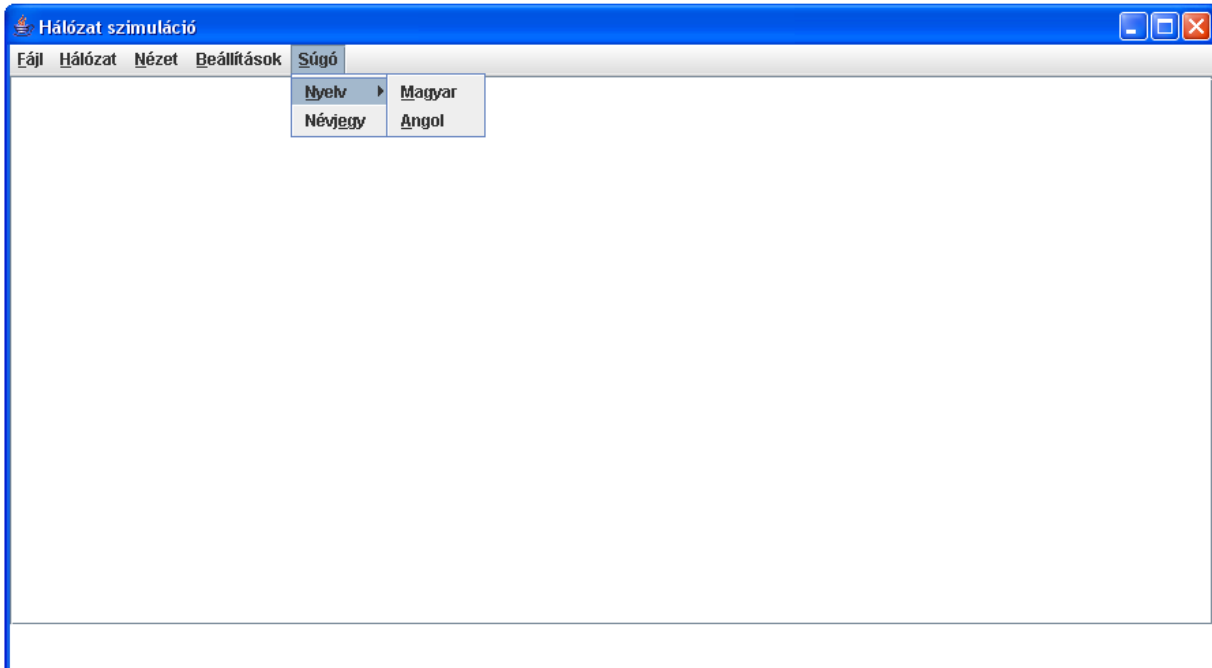
Várakozási idő helyes megadása:



## A Súgó menü

A Súgó menü gyorsbillentyűje: Alt + s.

A Súgó menüben a programmal kapcsolatos beállításokat lehet elvégezni, és a programmal kapcsolatos információkat lehet találni.



### A Súgó menü Nyelv menüje

A Nyelv menü gyorsbillentyűje: Alt + n.

### A Súgó menü Nyelv menü Magyar menüpontja

A Magyar menü gyorsbillentyűje: Alt + m.

Ezzel a menüponttal lehet a program nyelvét magyarra állítani.

### A Súgó menü Nyelv menü Magyar menüpontja

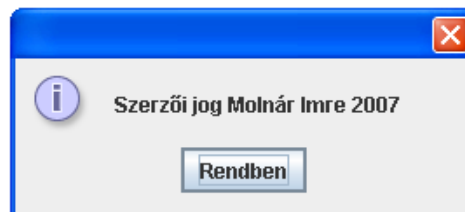
Az Angol menü gyorsbillentyűje: Alt + a.

Ezzel a menüponttal lehet a program nyelvét angolra állítani.

### A Súgó menü Névjegy menüpontja

A Névjegy menü gyorsbillentyűje: Alt + e.

Ezzel a menüponttal kaphatunk információt a program szerzőjéről és készítésének évéről.



## Gyorsbillentyűk táblázata

A menürendszer menüpontjaihoz a különböző nyelveken más gyorsbillentyűk tartoznak. A Fentiekben csak a magyar elnevezéseket és gyorsbillentyűket tartalmazza. A következő táblázat összefoglalja az elnevezéseket és a hozzájuk tartozó gyorsbillentyűt. Az azonos szinten lévő menüpontok gyorsbillentyűinek különbözniük kell. A különböző szinten lévők esetleges egyezése nem okoz gondot.

Magyar elnevezés	Magyar gyorsbillentyű	Angol elnevezés	Angol gyorsbillentyű
Fájl	f	File	f
Új	j	New	n
Betöltés	t	Load	l
Mentés	m	Save	s
Bezárás	b	Close	c
Kilépés	k	Exit	e
Hálózat	h	Network	n
Futtatás	f	Run	r
HS	h	HS	h
Félduplex	f	Halfduplex	h
Duplex	d	Duplex	d
Időszelet	i	Timeslice	t
LCR	l	LCR	l
Maxterjed	m	Maxexpand	m
Optmaxterjed	o	Optmaxexpand	o
Összes elemzése	z	Complete analysis	e
Szünet	s	Break	b
Folytatás	f	Continue	c
Leállítás	l	Stop	s
Nézet	n	View	v
Csak üzenetszám	z	Only message count	c
Egy folyamat üzenetei	e	Messages of one process	o
Összes folyamat üzenetei	s	Messages of all process	a
Grafikus megjelenítés	g	Graphical appearance	g
Elemzés	l	Analysis	a
Beállítások	b	Settings	s
Elemzés beállításai	e	Settings of analysis	a
Kitüntetett folyamat	k	Favorite process	f
Várakozási idő	v	Waiting time	w
Súgó	s	Help	h
Nyelv	n	Language	l
Magyar	m	Hungarian	h
Angol	a	English	e
Névjegy	e	About program	a

# Fejlesztői dokumentáció

## A feladat megfogalmazása

Vezetőválasztó algoritmus szimulációja. Az algoritmus egy hálózaton fut le. Kétféle hálózat adható meg: gyűrű és általános hálózat. A hálózat folyamatokból áll. A folyamatok által küldött üzeneteket kell kiírni és összeszámolni. A programban létre lehet hozni új hálózatot. Az új hálózatnak nevet, méretet vagy a folyamatok azonosítóit kell megadni. Ha hálózatnak a mérete a bemenő paraméter és a folyamatok azonosítói nem, akkor a program  $l$  és a hálózat méretének kétszerese közötti azonosítókat oszt ki a folyamatok között a program. Ha a folyamatok azonosítói és a hálózat mérete is adottak, akkor az azonosítók számának meg kell egyeznie a mérettel. Az algoritmus futása során a folyamatok üzeneteit kell megjeleníteni és összeszámolni, és az algoritmus lépéseinek számát összeszámolni. Az üzenetek szöveges és grafikus megjelenítése is lehetséges. Szöveges megjelenítés esetén a teljes üzenet kerül kiírásra. Grafikus megjelenítésnél elég azt jelezni, hogy melyik folyamat küld üzenetet. Szöveges megjelenítésnél lehet csak egy folyamat üzeneteit megjeleníteni, vagy az összes folyamat üzeneteit az algoritmus lépései szerint. A különböző megjelenítési formák között lehet választani. Az algoritmus lépései között várakozási idő van, hogy a felhasználó figyelemmel tudja követni az üzenetek útját. A várakozási idő állítása is lehetséges. Az algoritmus futását szüneteltetni is lehet, és teljesen leállítani az algoritmus lefutása előtt. Elemző nézetre is van lehetőség, akkor egy algoritmus többször fut le különböző méretű hálózatokon. Ekkor a minimális, átlagos és maximális lépésszám és üzenetszám kerül kiírásra. Valamint az ezekhez az értékekhez tartozó elméleti felső becslések konstans szorzói. Megadott méretű hálózatokon az összes lehetséges futtatható algoritmus többszöri futtatására is van lehetőség. Ekkor is a minimális, átlagos és maximális lépésszám és üzenetszám kerül kiírásra.

A hálózat adatai háttértárolón tárolhatóak és visszatölthetőek. Egy állományba több hálózat adatait is el lehet menteni. A hálózatokat a fájlokban a nevük azonosítja. Egy állományon belül nem lehet két azonos nevű hálózat. A tárolás XML formában történik. Az XML állomány szerkezete a következő:

```
<!ELEMENT lista (halozat*)>
<!ELEMENT halozat (nev, csucsok)>
<!ELEMENT nev (#PCDATA)>
<!ELEMENT csucsok (csucs)>
<!ELEMENT csucs (azonosito, szomszedok)>
<!ELEMENT azonosito (#PCDATA,)>
<!ELEMENT szomszedok (szomszed*)>
<!ELEMENT szomszed (#PCDATA)>
<!ATTLIST halozat gyuru (igaz|hamis) #REQUIRED>
```

A vezető választó algoritmusok a következők: a Le Lann, Chang és Roberts tiszteletére elnevezett LCR algoritmus, a Hirschberg—Sinclair algoritmus, az Időszelet algoritmus, a MaxTerjed és az OptMaxTerjed algoritmus.

Az LCR algoritmus vázlatosan: az algoritmus gyűrűben használható, egyirányú kapcsolatot használ, és nem igényli a gyűrű méretének ismeretét. A vezető folyamatnak lesz kimenete. Minden folyamat körbeküldi az azonosítóját a gyűrűben. Amikor valamelyik folyamathoz egy azonosító érkezik, összehasonlítja azt a sajátjával. Ha az érkezett azonosító nagyobb, mint a sajátja, a folyamat továbbítja azt, ha kisebb, akkor eldobja. Amennyiben az érkezett azonosító megegyezik a saját azonosítójával, akkor a hálózat önmagát nevezi meg vezetőnek.

A HS algoritmus vázlatosan: az algoritmus gyűrűben használható, kétirányú kapcsolatot használ, és nem igényli a gyűrű méretének ismeretét. A vezető folyamatnak lesz kimenete. Minden  $P_i$  folyamat a  $0,1,2,\dots$  szakaszokban (fázisokban) működik. Minden  $l$  szakaszban a  $P_i$  folyamat az  $u_i$  azonosítóját tartalmazó üzenetet küld mindkét szomszédja felé. Ezek az üzenetek  $2^l$  távolságot tesznek meg, azután visszatérnek a  $P_i$  kiindulási folyamathoz. Ha mindkét üzenet visszaérkezik, a  $P_i$  folyamat a következő szakasszal folytatódik. Amíg az  $u_i$  üzenet  $P_i$ -től távolodik, minden  $u_i$  útjába eső  $P_j$  folyamat összehasonlítja  $u_i$ -t a saját azonosítójával,  $u_j$ -vel. Ha  $u_i < u_j$ , akkor  $P_j$  eldobja az üzenetet, míg ha  $u_i > u_j$ , akkor  $P_j$  továbbítja  $u_i$ -t. Ha  $u_i = u_j$ , akkor ez azt jelenti, hogy a  $P_i$  folyamathoz még visszafordulás előtt a saját azonosítója érkezett, vagyis a  $P_i$  folyamat önmagát választja vezetőnek. Ha a  $P_i$  folyamat által küldött üzenet közeledik hozzá, akkor azt minden folyamat mindig továbbítja.

Az Időszelet algoritmus vázlatosan: az algoritmus gyűrűben használható, egyirányú kapcsolatot használ, és a folyamatok ismerik a hálózat  $n$  méretét. Az algoritmus a legkisebb azonosítójú folyamatot fogja vezetőnek választani. A számítások az  $1,2,3,\dots$  szakaszokban (fázisokban) hajtódnak végre, ahol minden szakasz  $n$  egymás utáni menetből áll. Minden szakasz olyan lehetséges üzenet forgalmat jelent, amihez bizonyos azonosítójú üzenetek tartoznak, és azok körbemennek a gyűrűn. A  $v$  szakaszban, ami a  $(v-1)n + 1, \dots, vn$  menetekből áll, csak a  $v$  azonosítójú üzenetek forgalma engedélyezett. Ha valamelyik időpillanatban a  $P_i$  a  $v$  azonosítóval rendelkezik és a  $(v-1)n + 1$  menet nélkül ér véget, hogy a  $P_i$  folyamat előzőekben egyetlen *nem-null* üzenetet kapott volna, akkor  $P_i$  önmagát választja vezetőnek, és a saját azonosítóját körbeküldi a gyűrűn. Amikor az ez üzenet megy körbe a gyűrűn, az összes érintett folyamat feljegyzi annak megérkezését. Ez a feljegyzés a későbbiekben megakadályozza őket abban, hogy önmagukat választhassák vezető folyamatnak, vagy bármely későbbi fázisban üzenetküldést kezdeményezzenek.

A MaxTerjed algoritmus vázlatosan: az algoritmus tetszőleges erősen összefüggő hálózatban használható. A folyamatok ismerik a hálózat átmérőjét. A legnagyobb azonosítójú folyamat választja magát vezetőnek. Minden folyamat azt a maximális egyedi azonosítót tartja nyilván, amely a végrehajtás adott pillanatáig eljutott hozzá (kezdetben ez a saját egyedi azonosítója). Mindegyik menetben minden egyes folyamat továbbítja ezt a maximális egyedi azonosítót a kimenő szomszédjainak. Ha átmérő darab menet után egy folyamat a saját egyedi azonosítójával azonos értékű maximális egyedi azonosítót tárol, vezetőnek választja magát; egyébként nemvezető lesz.

Az OptMaxTerjed algoritmus vázlatosan: az algoritmus tetszőleges erősen összefüggő hálózatban használható. A folyamatok ismerik a hálózat átmérőjét. A legnagyobb azonosítójú folyamat választja magát vezetőnek. A MaxTerjed algoritmust optimalizált változata. Egy folyamat csak akkor küldi el a legnagyobb addig megismert egyedi azonosító értékét, amikor annak először birtokába jut, és nem pedig minden menetben.



## A felhasznált technikák

A program java nyelven írtam. A futtatásához java 1.5 vagy annál újabb változat kell. A következő java osztályok használatával készült a program.

A **java.awt.BorderLayout** osztály a komponensek elhelyezésére szolgál. A komponenseket a „NORTH”, „SOUTH”, „EAST”, „WEST” nevekkel lehet ellátni, ennek megfelelően a felső, az alsó, a bal, illetve a jobb oldali kerethez lesznek igazítva. A „CENTER” elnevezés esetén a komponens a maradék helyet fogja elfoglalni. Így maximum öt komponenst tartalmazhat a **Container**.

A **java.awt.Color** osztály az RGB alapú színkezelést valósítja meg. Ha rajzoláskor színt akarunk választani, akkor egy **Color** objektumot kell átadni paraméterként. Az osztály tartalmaz előre definiált statikus **Color** objektumokat. Ezek a gyakorlatban használt színeknek felelnek meg, neveik a következők: **black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow**.

A **java.awt.Graphics** osztály biztosítja a primitív rajzoló műveleteket, ugyanakkor grafikus kontextust is. Utóbbi funkciója miatt kontextus manipuláló metódusokat is tartalmaz.

A **java.awt.Graphics2D** osztály: A **Graphics** osztály leszármazottja, amelynek jobbak a geometriákat, a koordináta-transzformációkat és a színeket kezelő, valamint a szöveg megjelenítő funkciói.

A **java.awt.GridLayout** osztály a komponenseket egy négyzetrácsban helyezi el sorfolytonos feltöltéssel.

A **java.awt.event.ActionEvent** osztály az általános esemény.

A **java.awt.event.ActionListener** osztály **ActionEvent** típusú események fogadásához szükséges interfész.

A **java.awt.event.KeyEvent** osztály a komponens szintű billentyűzetesemény-osztály.

A **java.awt.geom.Ellipse2D** osztály a befoglaló téglalapjával definiált ellipszisek absztrakt osztálya. Két konkrét megvalósítása van a **Ellipse2D.Double** és az **Ellipse2D.Float** osztályok

A **java.awt.geom.Line2D** osztály a kétdimenziós szakaszok absztrakt osztály. Két konkrét megvalósítása van a **Line2D.Double** és a **Line2D.Float** osztályok.

A **java.io.DataOutputStream** ez a szűrő a konstruktorának paraméterül adott kimeneti csatornára a Java adattípusaiba tartozó értékeket tud kiírni.

A **java.io.File** elérési útvonal ábrázolására szolgál. Egy **File** objektummal műveleteket is végezhetünk a fájlrendszer azon fájlján, amelyet az elérési útvonal azonosít. Ezekhez az is szükséges, hogy a **SecurityManager** (ha van) engedélyt adjon rá, különben **SecurityException** váltódik ki.

A **java.io.FileNotFoundException** kiváltódik, ha meg akarunk nyitni egy fájlt, de a paraméterül adott elérési útvonalhoz nem tartozik a fájlrendszerben fájl.

A **java.io.FileOutputStream** a kimeneti bájtcsonna a fájlrendszer egy fájlja.

A **java.io.IOException** sikertelen vagy félbeszakadt input/output műveletek váltják ki.

A **java.io.RandomAccessFile** véletlen elérésű fájl: használható bemenetnek és kimenetnek egyaránt. Megvalósítja a **DataInput** és **DataOutput** interfészeket. Műveletei a **DataInputStrem** és **DataOutputStream** osztályokhoz hasonlóan történik.

A **java.io.StringWriter** olyan kimeneti karaktercsatorna, mely az adatokat egy stringbufferbe írja ki.

A **java.util.ArrayList** tömb jellegű **List** implementáció.

A **java.util.Locale** nemzetközi programokban a környezet azonosítására szolgál.

A **java.util.Properties** szöveges jellemzők tárolására és visszakérdezésére szolgáló osztály.

A **java.util.Random** egyenletes és normális eloszlású pszeudo-véletlenszámok előállítására szolgáló osztály.

A **java.util.ResourceBundle** környezetfüggő jellemzők tárolására szolgáló osztály.

A **javax.swing.ButtonGroup** gombok és menüpontok csoportját reprezentáló osztály.

A **javax.swing.JButton** nyomógombokat reprezentáló osztály.

A **javax.swing.JComponent** ez az osztály az ősoosztálya minden Swing komponensnek.

A **javax.swing.JDialog** dialógus ablakokat reprezentáló osztály.

A **javax.swing.JFileChooser** fájlkiválasztó dialógusdobozokat reprezentáló osztály.

A **javax.swing.JFrame** kerettel és fejléccel rendelkező ablakokat reprezentáló osztály.

A **javax.swing.JLabel** címkéket megvalósító osztály.

A **javax.swing.JMenu** menüket megvalósító osztály.

A **javax.swing.JMenuBar** menüsorokat megvalósító osztály.

A **javax.swing.JMenuItem** menüpontokat megvalósító osztály.

A **javax.swing.JOptionPane** dialógusdobozokat megvalósító osztály.

A **javax.swing.JPanel** a legegyszerűbb Swing konténer megvalósító osztály.

A **javax.swing.JRadioButtonMenuItem** csoportba szervezhető kiválasztható menüpontokat reprezentáló osztály.

A **javax.swing.JScrollPane** görgethető panelt megvalósító osztály.

A **javax.swing.JTable** táblázat megvalósítását lehetővé tevő osztály.

A **javax.swing.JTextArea** többsoros szövegeket reprezentáló osztály.

A **javax.swing.JTextField** egysoros szövegdobozokat reprezentáló osztály.

A **javax.xml.parsers.DocumentBuilder** xml dokumentum építő osztály.

A **javax.xml.parsers.DocumentBuilderFactory** xml dokumentum építő osztályt létrehozó osztály.

A **javax.xml.parsers.ParserConfigurationException** xml dokumentum létrehozása közben fellépő hibát reprezentáló osztály.

A **javax.xml.transform.Transformer** az xml megjelenítését megformázó osztály

A **javax.xml.transform.TransformerConfigurationException** a megjelenítés beállítása közben fellépő hibát reprezentáló osztály

A **javax.xml.transform.TransformerFactory** xml Transformert osztályt gyártó osztály.

A **javax.xml.transform.TransformerException** az xml formázása közben fellépő hibát reprezentáló osztály.

A **javax.xml.transform.dom.DOMSource** a DOM ábrázolási módot megvalósító osztály.

A **javax.xml.transform.stream.StreamResult** az xml karakter sorozattá alakító osztály.

A **org.w3c.dom.Document** az xml dokumentumot megvalósító osztály.

A **org.w3c.dom.Element** xml adatok létrehozására szolgáló osztály.

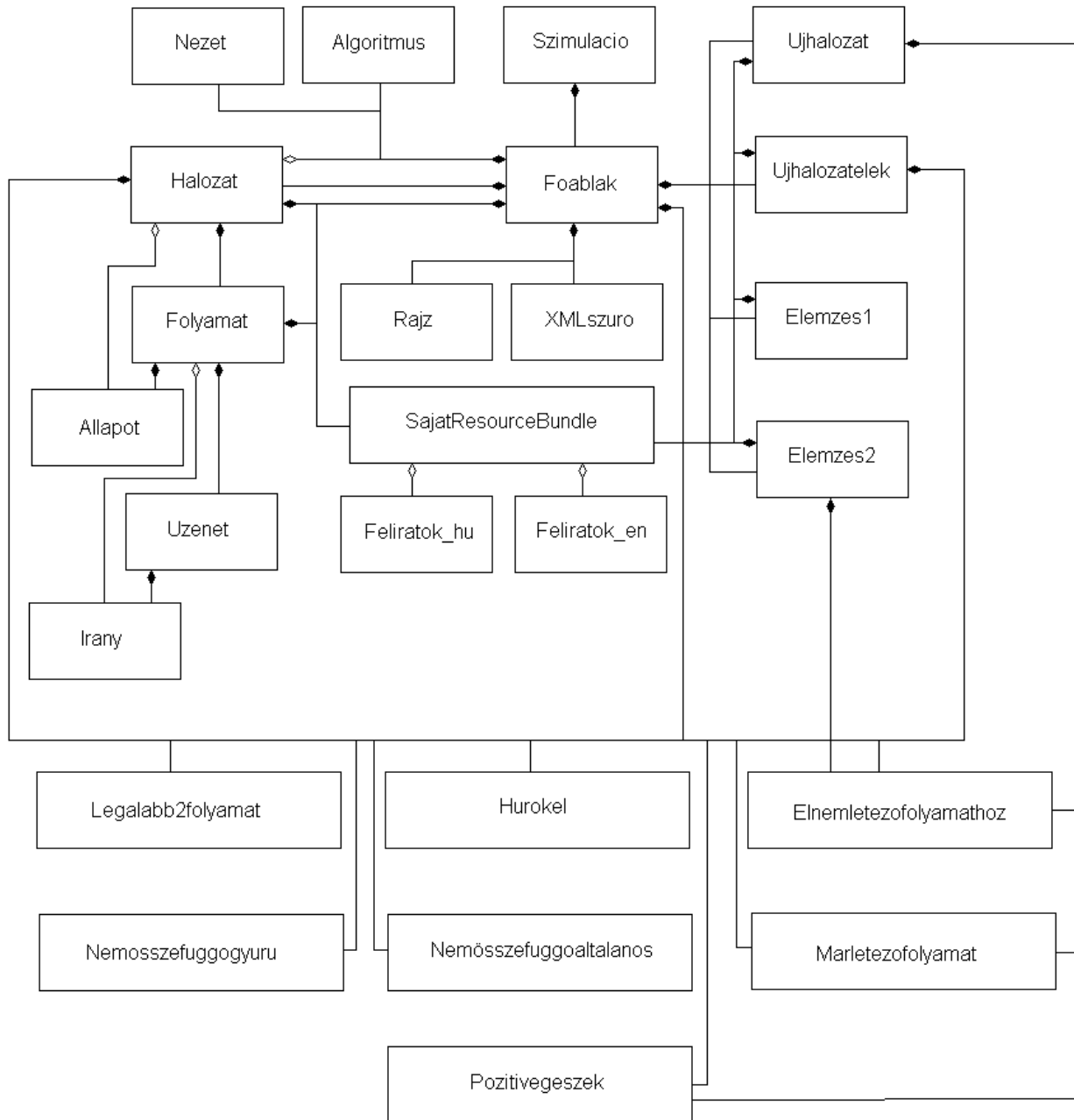
A **org.w3c.dom.NamedNodeMap** xml csúcsok attribútumainak tárolását megvalósító osztály.

A **org.w3c.dom.Node** xml csúcsot megvalósító osztály.

A **org.w3c.dom.NodeList** xml csúcsok listáját megvalósító osztály.

A **org.xml.sax.SAXException** a SAX ábrázolású feldolgozás közben fellépő hibákat reprezentáló osztály.

## Osztály diagramm



## Csomagok, osztályok

### Uzenet csomag

#### Uzenet osztály

Az *Uzenet* osztály valósítja meg a folyamatok által küldött üzeneteket.

**public enum Irany {jon, megy}** Az üzenetek irányát megvalósító osztály.

Változói:

**private int azonosito** Az üzenetben az azonosítót tároló változó.

**private int fazis** Az üzenetben a fázist tároló változó.

**private Irany irany** Az üzenetben az irányt tároló változó.

Konstruktorai:

**public Uzenet()** Létrehozz egy új *Uzenet*et. A változóknak nem ad értéket.

**public Uzenet(int azonosito)** A paraméterben megadott értéket adja az azonosítónak, és létrehozz egy új *Uzenet*et.

**public Uzenet(int azonosito, Irany irany, int fazis)** A paraméterben megadott értékeket adja az azonosítónak, az iránynak és a fázisnak, és létrehozz egy új *Uzenet*et.

Függvényei:

**public int getAzonosito()** Visszaadja az üzenetben lévő azonosítót.

**public int getFazis()** Visszaadja az üzenetben lévő fázist.

**public Irany getIrany()** Visszaadja az üzenetben lévő irányt.

### Folyamat csomag

#### Folyamat osztály

A *Folyamat* osztály valósítja meg a hálózatban lévő folyamatokat.

**public enum Allapot{ismeretlen, vezető, nemvezető}** Folyamat állapotát magvalósító osztály.

Változói:

**private Allapot allapot** Folyamat állapotát tároló változó. Kezdeti értéke ismeretlen.

**private int azonosito** Folyamat azonosítóját tároló változó.

**private int fazis** Folyama fázisát tároló változó. Kezdeti értéke nulla. Csak a HS algoritmus használja.

**private ArrayList<Uzenet> kap** Általános hálózat esetén az érkező üzenetek tárolására szolgáló lista.

**private Uzenet kap\_minusz** Negatív irányból érkező üzenet tárolására szolgáló változó.

**private Uzenet kap\_plusz** Pozitív irányból érkező üzenet tárolására szolgáló változó.

**private Uzenet kuld\_minusz** Negatív irányba küldendő üzenetet tároló változó.

**private Uzenet kuld\_plusz** Pozitív irányba küldendő üzenetet tároló változó.

**private int max\_azon** A kap változóban tárolt üzenetek közül a legnagyobb azonosító tárolására szolgál. Csak általános hálózat esetén használatos.

**private int menetszam** Folyamat menetszámát tároló változó. Kezdeti értéke 1. Csak a HS algoritmus használja.

**private SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

Konstruktorai:

**public Folyamat()** Létrehoz egy új folyamatot. A változóknak nem ad értéket.

**public Folyamat(int azonosito)** Létrehoz egy új folyamatot. A paraméterben megadott értéket adja az *azonositonak*.

Függvényei:

**public Allapot getAllapot()** Visszaadja a folyamat állapotát.

**public int getAzonosito()** Visszaadja a folyamat azonosítóját.

**public String getUzenetDuplex()** Szövegesen kiírandó üzenettel tér vissza duplex gyűrű esetében.

**public String getUzenetSimplex()** Szövegesen kiírandó üzenettel tér vissza szimplex gyűrű és általános hálózat esetében.

**public int getUzenetSzamAltalanos(Integer[] szomszed)** Visszaadja, hogy a folyamat mennyi üzenetet küldött általános hálózat esetében. A paraméterben a szomszédsági mátrix folyamatra vonatkozó sorát kell megadni.

**public int getUzenetSzamGyuru()** Visszaadja, hogy a folyamat mennyi üzenetet küldött gyűrű esetében. A visszatérési értékek: 0, 1, 2.

**public boolean isUzenetMinus()** Igazzal tér vissza, ha a folyamat küldött üzenetet a negatív irányba. Különben hamissal.

**public boolean isUzenetPlusz()** Igazzal tér vissza, ha a folyamat küldött üzenetet a pozitív irányba. Különben hamissal.

**public void KezdoUzenetHSDuplex()** Folyamat kezdő üzeneteit állítja be a HS algoritmus duplex változatának esetére.

**public void KezdoUzenetHSFelDuplex()** Folyamat kezdő üzeneteit állítja be a HS algoritmus félduplex változatának esetére.

**public void KezdoUzenetIdoszelet()** Folyamat kezdő üzeneteit állítja be az Időszelet algoritmus esetére.

**public void KezdoUzenetLCR()** Folyamat kezdő üzeneteit állítja be az LCR algoritmus esetére.

**public void KezdoUzenetMaxTerjed()** Folyamat kezdő üzeneteit állítja be *Maxterjed* és *Optmaxterjed* algoritmusok esetére.

**public void UzenetFogadasHS(Folyamat p1, Folyamat p2)** Paraméterben megadott folyamatoktól fogadja az üzenetet. HS algoritmusok esetén használható.

**public void UzenetFogadasIdoszelet(Folyamat p1)** Paraméterben adott folyamattól fogadja az üzenetet. Időszelet algoritmus esetén használható.

**public void UzenetFogadasLCR(Folyamat p1)** Paraméterben megadott folyamattól fogadja az üzenetet. LCR algoritmus esetén használható.

**public void UzenetFogadasMaxTerjed(int sorszam, int meret, Folyamat[] f, Integer[][] szomszed)** Az üzenetek fogadására szolgál *Maxterjed* és *Optmaxterjed* algoritmusok esetén. A *sorszam* paraméter a fogadó folyamat sorszáma a szomszédsági mátrixban. A *meret* a hálózatban lévő folyamatok száma. Az *f* a folyamatokat tároló tömb, a *szomszed* a szomszédsági mátrix.

**public void UzenetSzamitasHS()** Az új üzeneteket számolja ki HS algoritmus esetén.

**public void UzenetSzamitasIdoszelet(long lepszam, int halozat\_meret)** Az új üzenetet számolja ki *Időszelet* algoritmus esetén. A *lepszam*ban kell megadni, hogy az algoritmus, hányadik lépésben van. A *meret* a hálózatban lévő folyamatok száma.

**public void UzenetSzamitasLCR()** Az új üzenetet számolja ki LCR algoritmus esetén.

**public void UzenetszamitasMaxTerjed(long menetszam, int atmero)** Az új üzenetet számolja ki *Maxterjed* algoritmus esetén. A *lepszam*ban kell megadni, hogy az algoritmus, hányadik lépésben van. Az *atmeroben* a hálózat átmérőjét.

**public void UzenetszamitasOptMaxTerjed(long menetszam, int atmero)** Az új üzenetet számolja ki *Optmaxterjed* algoritmus esetén. A *lepszamban* kell megadni, hogy az algoritmus, hányadik lépésben van. Az *atmeroben* a hálózat átmérőjét.

## Halozat csomag

### Halozat osztály

A *Halozat* osztály valósítja meg a hálózatot.

Változói:

**private int atmero** Hálózat átmérőjét tároló változó.

**private Long[][] azonositok** Grafikus megjelenítésnél az azonosítókat és a megjelenítési helyének koordinátáit tároló mátrix.

**private Folyamat[] f** Folyamatokat tároló tömb.

**private Foablak fa** *Foablak*ra mutató változó. A *Foablak* függvényeinek elérése érdekében.

**private boolean gyuru** Változó értéke igaz, ha a hálózat gyűrű, és hamis, ha a hálózat általános.

**private int kituntetett** Kitüntetett folyamat sorszámát tároló változó.

**private Ellipse2D[] korok** Folyamatokat szimbolizáló körök tárolására szolgáló tömb.

**private volatile boolean leall** Az algoritmusok leállításáért felelős változó. Kezdeti értéke hamis.

**private int meret** Hálózatban lévő folyamatok számát tároló változó.

**private int meret\_me** Hálózatban lévő folyamatok számánál eggyel kisebb számot tároló változó.

**private String nev** Hálózat nevét tároló változó.

**private Line2D[][] nyilakaltalanos** Az üzeneteket jelképező a nyilak hegyének végeit összekötő vonalakat tároló tömb általános hálózat esetén.

**private Line2D[][] nyilakbelul** Az üzeneteket jelképező, gyűrűn belüli nyilak tárolására szolgáló tömb.

**private Line2D[][] nyilakkivul** Az üzeneteket jelképező, gyűrűn kívüli nyilak tárolására szolgáló tömb.

**private Line2D[] osszekotok** Folyamatok közötti kommunikációs csatornákat szimbolizáló vonalak tárolására szolgáló tömb gyűrű hálózat esetén.

**private Line2D[][] osszekotokaltalanos** Folyamatok közötti kommunikációs csatornákat szimbolizáló vonalak tárolására szolgáló tömb általános hálózat esetén.

**private SajatResourceBundle srb** A szövegek megjelenítéséért felelős osztály példányát tárolja.

**private Integer[][] szomszed** Folyamatok szomszédsági mátrixa.

**private volatile boolean var** Az algoritmusok futásának felfüggesztéséért felelős változó. Kezdeti értéke hamis.

**private int varakozas** Az algoritmusok futása során szöveges vagy grafikus megjelenítés esetén két lépés közötti idő.

Konstruktorai:

**public Halozat()** Létrehoz egy új hálózatot. A várakozási időt 2000 milliszekundumra állítja.

**public Halozat(Foablak fa, boolean gyuru, String nev, Folyamat[] f, Integer[][] szomszed, int varakozas) throws Elnemletozofolyamathoz, Hurokel, Legalabb2folyamat, Marletezofolyamat, Nemosszefuggoaltalanos, Nemosszefuggogyuru, Pozitivegeszek** Létrehoz egy új hálózatot. Az *fa* értékének a *foablakot* kell megadni. A *gyuru* értékének igazat, ha gyűrűt szeretnénk létrehozni és hamisat, ha általános hálózatot. A *nev*nek a hálózat nevét kell megadni. Az *f*-nek a folyamatokat tartalmazó tömböt kell megadni. A *szomszed*nek a szomszédossági

mátrixot kell megadni. A *varakozas*nak a várakozási időt két lépés között szöveges vagy grafikus megjelenítés esetén. Ha egy folyamatból nem létező folyamatba mutat él, akkor *Elnemletezofolyamathoz* kivételt dob. Ha szomszédossági mátrixban egy folyamatnak önmagára mutató éle van, akkor *Hurokel* kivételt dob. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha a hálózat nem összefüggő, akkor *Nemosszefuggoaltalanos* vagy *Nemosszefuggogyuru* kivételt dob attól függően, hogy gyűrű vagy általános hálózat. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**public Halozat(Foablak fa, boolean gyuru, String nev, Folyamat[] f, Integer[][] szomszed, int varakozas, int kituntetett) throws Elnemletezofolyamathoz, Hurokel, Legalabb2folyamat, Marletezofolyamat, Nemosszefuggoaltalanos, Nemosszefuggogyuru, Pozitivegeszek** Mint az előző, csak a kitüntetett folyamat sorszámát is meg kell adni.

**public Halozat(Foablak fa, boolean gyuru, String nev, int atmero, Folyamat[] f, Integer[][] szomszed, int varakozas)** Létrehoz egy új hálózatot. Az *fa* értékének a *foablakot* kell megadni. A *gyuru* értékének igazat, ha gyűrűt szeretnénk létrehozni és hamisat, ha általános hálózatot. A *nev*nek a hálózat nevét kell megadni. Az *atmeronek* a hálózat átmérőjét kell megadni. Az *f*-nek a folyamatokat tartalmazó tömböt kell megadni. A *szomszed*nek a szomszédossági mátrixot kell megadni. A *varakozas*nak a várakozási időt két lépés között szöveges vagy grafikus megjelenítés esetén.

**public Halozat(Foablak fa, boolean gyuru, String nev, int atmero, Folyamat[] f, Integer[][] szomszed, int varakozas, int kituntetett)** Mint az előző, csak a kitüntetett folyamat sorszámát is meg kell adni.

**public Halozat(Foablak fa, boolean gyuru, String nev, int meret, int varakozas) throws Legalabb2folyamat** Létrehoz egy új hálózatot. Az *fa* értékének a *foablakot* kell megadni. A *gyuru* értékének igazat, ha gyűrűt szeretnénk létrehozni és hamisat, ha általános hálózatot. A *nev*nek a hálózat nevét kell megadni. A *meret*nek a hálózatban lévő folyamatok számát. A *varakozas*nak a várakozási időt két lépés között szöveges vagy grafikus megjelenítés esetén. Egy folyamatok azonosítói és általános hálózat esetén az élek véletlenszerűen lesznek kiosztva. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob.

**public Halozat(Foablak fa, boolean gyuru, String nev, int meret, int varakozas, int kituntetett) throws Legalabb2folyamat** Mint az előző, csak a kitüntetett folyamat sorszámát is meg kell adni.

**public Halozat(Foablak fa, String nev, Folyamat[] f, int varakozas) throws Legalabb2folyamat, Marletezofolyamat, Pozitivegeszek** Létrehoz egy új hálózatot. Az *fa* értékének a *foablakot* kell megadni. A *nev*nek a hálózat nevét kell megadni. Az *f*-nek a folyamatokat tartalmazó tömböt kell megadni. A *varakozas*nak a várakozási időt két lépés között szöveges vagy grafikus megjelenítés esetén. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**public Halozat(Foablak fa, String nev, Folyamat[] f, int varakozas, int kituntetett) throws Legalabb2folyamat, Marletezofolyamat, Pozitivegeszek** Mint az előző, csak a kitüntetett folyamat sorszámát is meg kell adni.

Függvényei:

**public Document DocumentBetoltesXMLbol(File fajlnev) throws FileNotFoundException, IOException, ParserConfigurationException, SAXException** Paraméterként kapott nevű XML fájlt betölti és a belőle létrehozott XML dokumentummal tér vissza. Hiba esetén kivétel dobódik.

**private synchronized void elemzes(Algoritmus algoritmus)** Paraméterként kapott *algoritmus* elemző futtatását végző eljárás.

**private synchronized void elemzesosszes()** Ez az eljárás végzi az összes algoritmus elemző futtatását.

**public ArrayList<Integer> FolyamatAzonositokDocbol(Document doc, String halozatnev)** **throws Nemosszefuggoaltalanos** Paraméterben kapott XML dokumentumban megkeresi a paraméterben kapott nevű hálózatot és a hálózat folyamatainak azonosítóit tartalmazó listával tér vissza. Ha hibás az XML fájl, akkor *Nemosszefuggoaltalanos* kivételt dob.

**public Folyamat[] folyamatkeszito(ArrayList<Integer> al)** Paraméterül kapott listából tömbös tárolási formába hozza a folyamatokat.

**public int getAtmero()** Visszaadja a hálózat átmérőjét.

**public Folyamat[] getFolyamatokTomb()** Visszaadja a folyamatokat tároló tömböt.

**public boolean getGyuru()** Igazzal tér vissza, ha gyűrű a hálózat, és hamissal, ha általános hálózat.

**public int getKituntetett()** Visszaadja a kitüntetett folyamat sorszámát a tömbben.

**public int getMeret()** Visszaadja a hálózat méretét.

**public String getNev()** Visszaadja a hálózat nevét.

**public Integer[][] getSzoszed()** Visszaadja a szomszédossági mátrixot.

**public int getVarakozas()** Visszaadja a várakozási időt.

**private void grafikaszamitas()** Grafikus megjelenítéshez szükséges adatokat számolja ki ez az eljárás.

**public boolean HalozatGyurueDocbol(Document doc, String halozatnev)** Paraméterben kapott dokumentumban megkeresi a paraméterben kapott nevű hálózatot és igazzal tér vissza, ha gyűrű, különben hamissal.

**public void HalozatMentesXMLbe(Document doc, File fajlnev)** **throws FileNotFoundException, IOException, ParserConfigurationException, SAXException, TransformerException, TransformerConfigurationException** Elmenti a paraméterként kapott XML dokumentumot a paraméterként kapott nevű fájlba. Hibák esetén kivétel dobódik.

**public String[] HalozatNevekDocbol(Document doc)** Paraméterben kapott XML dokumentumban található hálózatok neveit tartalmazó tömbbel tér vissza.

**public boolean HalozatNevKeresDocban(Document doc)** Paraméterül kapott XML dokumentumban megkeresi az adott hálózat nevét. Igazzal tér vissza, ha megtalálja és hamissal, ha nem találja meg.

**public Document HalozatTorolDocbol(Document doc)** Paraméterül kapott XML dokumentumból törli az adott nevű hálózatot és ezzel a dokumentummal tér vissza.

**public boolean helyesadatokCsucsok(Folyamat[] f)** **throws Legalabb2folyamat, Marletezofolyamat, Pozitivegeszek** Paraméterben kapott folyamatok tömbről eldönti, hogy helyes-e, Ha igen, akkor igazzal tér vissza különben hamissal. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**private synchronized Long[] HSDuplexAlgoritmus()** *HS* duplex algoritmust megvalósító függvény. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.

**private synchronized void HSDuplexAlgoritmus(Nezet nezet)** *HS* duplex algoritmust megvalósító függvény. A paraméterben megadott szerint írja ki az üzeneteket.

**private synchronized Long[] HSFedDuplexAlgoritmus()** *HS féduplex* algoritmust megvalósító függvény. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.



**private synchronized void HSFeDuplexAlgoritmus(Nezet nezet)** *HS félduplex* algoritmust megvalósító függvény. A paraméterben megadott szerint írja ki az üzeneteket.

**private synchronized Long[] IdoszeletAlgoritmus()** Az *Időszelet* algoritmust megvalósító függvény. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.

**private synchronized void IdoszeletAlgoritmus(Nezet nezet)** Az *Időszelet* algoritmust megvalósító függvény. A paraméterben megadott szerint írja ki az üzeneteket.

**private float konstanslepasszam(Float lepes, int meret, Algoritmus algoritmus)** Paraméterben kapott algoritmus, paraméterben kapott lépésszámához és méretéhez tartozó elméleti felső korlát konstans szorzójával tér vissza.

**private float konstansuzenetszam(Float uzenet, int meret, Algoritmus algoritmus)** Paraméterben kapott algoritmus paraméterben kapott üzenetszámához és méretéhez tartozó elméleti felső korlát konstans szorzójával tér vissza.

**private synchronized Long[] LCRAgoritmus()** Az *LCR* algoritmust megvalósító függvény. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.

**private synchronized void LCRAgoritmus(Nezet nezet)** Az *LCR* algoritmust megvalósító függvény. A paraméterben megadott szerint írja ki az üzeneteket.

**public synchronized void leallitas()** Igazara állítja a leáll változót, és leállítja ezzel az algoritmus futását.

**private synchronized Long[] MaxTerjedAlgoritmus()** *Maxterjed* algoritmust megvalósító eljárás. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.

**private synchronized void MaxTerjedAlgoritmus(Nezet nezet)** *Maxterjed* algoritmust megvalósító eljárás. A paraméterben megadott szerint írja ki az üzeneteket.

**public synchronized void megallitas()** Vár változó értékét igazra állítja. Ezzel szünetelteti az algoritmus futását.

**private synchronized Long[] OptMaxTerjedAlgoritmus()** Az *Optmaxterjed* algoritmust megvalósító függvény. A visszatérési értéke a lépésszámot és az üzenetszámot tartalmazó tömb.

**private synchronized void OptMaxTerjedAlgoritmus(Nezet nezet)** Az *Optmaxterjed* algoritmust megvalósító függvény. A paraméterben megadott szerint írja ki az üzeneteket.

**public int osszefuggoAltalanos(boolean hurokelvizsgalat, Integer[][] szomszedok) throws Hurokel, Nemosszefuggoaltalanos** Visszatérési értéke igaz, ha a paraméterben kapott szomszédsági mátrix által reprezentált általános hálózat erősen összefüggő. Ha nem, akkor *Nemosszefuggoaltalanos* kivételt dob. Ha szomszédsági mátrixban egy folyamatnak önmagára mutató éle van és a *hurokelvizsgalat* igaz értéket kapott, akkor *Hurokel* kivételt dob.

**public boolean osszefuggoGyuru(Integer[][] szomszedok) throws Nemosszefuggogyuru** Visszatérési értéke igaz, ha a paraméterben kapott szomszédsági mátrix helyesen van megadva gyűrű esetére. Ha nincsen helyesen megadva, akkor *Nemosszefuggogyuru* kivételt dob.

**public synchronized void run()** A Thread osztály kötelező függvénye.

**public void setKituntetett(int kituntetett)** Kitüntetett folyamat a paraméterben megadott sorszámú folyamat lesz.

**public void setVarakozas(int varakozas)** Beállítja a várakozási időt a paraméterben megadottra.

**public Integer[][] szomszedkeszito(ArrayList<Integer> al, ArrayList<ArrayList<Integer>> alal) throws Elnemletezofolyamathoz** Az első paraméterben a folyamatok listáját kell megadni, a másodikban a szomszédsági mátrix adatait listák listájaként. A szomszédsági mátrixsal tér vissza. Ha egy folyamatból nem létező folyamathoz vezető él, akkor *Elnemletezofolyamathoz* kivételt dob.

**public ArrayList<ArrayList<Integer>> szomszedokDocbol(Document doc, String halozatnev)** Paraméterben kapott XML dokumentumban megkeresi a paraméterben kapott nevű hálózatot és a hálózat folyamatainak szomszédossági adatainak listák listájával tér vissza.

**public synchronized void ujrainditas()** Vár változó értékét hamisra állítja és felkelti az algoritmus.

**private synchronized void uzenetekKiirasaDuplex(Nezet nezet, long uzenetszam, int lepszam)** Nézetnek megfelelően a *foablak* megfelelő elemére kiírja az üzenetet, az üzenetszámot és a lépésszámot duplex gyűrű esetén.

**private synchronized void uzenetekKiirasaSimplex(Nezet nezet, long uzenetszam, int lepszam)** Nézetnek megfelelően a *foablak* megfelelő elemére kiírja az üzenetet, az üzenetszámot és a lépésszámot szimplex gyűrű és általános hálózat esetén.

**private void veletlenFolyamatokAzonositok(boolean gyuru, int meret)** A folyamatoknak véletlenszerűen osztja ki az azonosítókat és létrehozza a szomszédossági mátrixot. A *gyuru* értékének igazat kell adni, ha gyűrűt szeretnénk létrehozni, és hamisat, ha általános hálózatot. A *meret*nek a hálózatban lévő folyamatok számát.

## Ablak csomag

### Foablak osztály

A *Foablak* osztály valósítja meg a program kezelő felületét.

**public enum Algoritmus {lcr, hsduplex, hsfelduplex, idoszelet, maxterjed, optmaxterjed, osszelemzes}** Algoritmusok elnevezéseit megvalósító osztály.

**public enum Nezet {uzenetszam, egyfolyamat, osszesfolyamat, grafikus, elemzes}** Nézetek elnevezéseit megvalósító osztály.

Változói:

**private Algoritmus algoritmus** Kiválasztott algoritmus nevének tárolására szolgáló változó.

**private Integer futasszam** Elemző nézetnél az algoritmusok futásszámát tároló változó. Kezdeti értéke egy.

**private Halozat halozat** Hálózat tárolására szolgáló változó.

**private ArrayList<Integer> meretek** Elemző nézethez a hálózatok méretét tároló lista.

**private Nezet nezet** Kiválasztott nézet nevének tárolására szolgáló változó kezdeti értéke *uzenetszam*.

**private Rajz rajz** Grafikus megvalósítást szolgáló változó.

**static final long serialVersionUID** A JFrame osztály statikus változója.

**private SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

**private JMenu fajlmenu, halozatmenu, futtatas, hs, nezetmenu, beallitasok, sugomenu, nyelvmenu** Az azonos nevű menüket megvalósító változók.

**private JMenuBar menubar** Menübárt megvalósító változók.

**private JMenuItem uj, betoltes, mentes, bezar, kilepes, felduplex, duplex, idoszelet, lcr, maxterjed, optmaxterjed, osszelemzes, megallitas, folytatás, leallitas, elemzesbeallitasok, kitunetettfolyamat, varakozasiido, magyar, angol, nevjegy** Az azonos nevű menüpontokat megvalósító változók.

**private JPanel panel** Panelt megvalósító változó.

**private JRadioButtonMenuItem uzenetszam, egyfolyamat, osszesfolyamat, grafikus, elemzes** Választható menüpontokat megvalósító változók.

**private JScrollPane scrollpane1, scrollpane2, scrollpane3** Görgethető paneleket megvalósító változók.

**private JTable tablazat** Táblázatot megvalósító változó.

**private JTextArea uzenetszamok, uzenetek** Többsoros szöveges mezőket megvalósító változók.

Konstruktor:

**public Foablak()** Létrehoz egy új *Foablakot*.

Függvényei:

**public void actionPerformed(ActionEvent ae)** Az *ActionListener* osztály kötelező függvénye. Az események kezelésére szolgál.

**private void elemzesMegjelenites()** Az elemző megjelenítéshez szükséges elemeket helyezi a panelre.

**public void foablakKirajzolas()** Kirajzolja a *Foablakot*.

**public Algoritmus getAlgoritmus()** Visszaadja a kiválasztott algoritmus nevét.

**public Integer getFutasszam()** Visszaadja a *futasszamot*.

**public Halozat getHalozat()** Visszaadja a *halozat* változó értékét.

**public ArrayList<Integer> getMeretek()** Visszaadja a *meretek* listát.

**public Nezet getNezet()** Visszaadja a kiválasztott nézet nevét.

**public Rajz getRajz()** Visszaadja a *rajz* változót.

**public JTable getTablazat()** Visszaadja a *tablazat* változó értékét.

**private void grafikusMegjelenites()** Grafikus megjelenítéshez szükséges elemeket helyezi a panelre.

**public boolean helyesadatokCsucsok(ArrayList<Integer> al) throws Legalabb2folyamat, Marletezofolyamat, Pozitivegeszek** Meghívja a *Halozat* osztály *helyesadatokCsucsok* függvényét és annak a visszatérési értékével tér vissza. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**public void menueleres1()** Menüpontok elérhetősége a program elindulásakor.

**public void menueleres2a()** Menüpontok elérhetősége, ha létrehoztunk vagy betöltöttünk egy gyűrűt.

**public void menueleres2b()** Menüpontok elérhetősége, ha létrehoztunk vagy betöltöttünk egy általános hálózatot.

**public void menueleres3()** Menüpontok elérhetősége, ha elindítottunk egy algoritmus, vagy az összes elemzést.

**public void menueleres4a()** Menüpontok elérhetősége, ha leállítottuk vagy lefutott egy algoritmus vagy az összes elemzése gyűrű esetén.

**public void menueleres4b()** Menüpontok elérhetősége, ha leállítottuk vagy lefutott egy algoritmus vagy az összes elemzése általános hálózat esetén.

**public void menueleres5()** Menüpontok elérhetősége, a megállítjuk az algoritmus vagy az összes elemzése futását.

**public void menueleres6()** Menüpontok elérhetősége, ha folytatjuk az algoritmus vagy az összes elemzése futását.

**public void setFutasszam(Integer futasszam)** A *futasszamot* a paraméterben kapott értékre állítja.

**private void setGyorsBillentyukAngol()** Beállítja angol nyelvű menürendszer esetére a gyorsbillentyűket.

**private void setGyorsBillentyukMagyar()** Beállítja magyar nyelvű menürendszer esetére a gyorsbillentyűket.

**public void setHalozat(boolean gyuru, String nev, ArrayList<Integer> al, ArrayList<ArrayList<Integer>> alal) throws Elnemletezofolyamathoz, Hurokel, Legalabb2folyamat, Marletezofolyamat, Nemosszefuggoaltalanos, Nemosszefuggogyuru, Pozitivegeszek** A *halozat* változónak új értéket ad a paraméterben szereplő változók szerint. Ha egy folyamatból nem létező folyamatba mutat él, akkor *Elnemletezofolyamathoz* kivételt dob. Ha szomszédsági mátrixban egy folyamatnak önmagára mutató éle van, akkor *Hurokel* kivételt dob. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha a hálózat nem összefüggő, akkor *Nemosszefuggoaltalanos* vagy *Nemosszefuggogyuru* kivételt dob attól függően, hogy gyűrű vagy általános hálózat. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**public void setHalozat(boolean gyuru, String nev, Integer meret) throws Legalabb2folyamat** A *halozat* változónak új értéket ad a paraméterben szereplő változók szerint.

**public void setHalozat(String nev, ArrayList<Integer> al) throws Legalabb2folyamat, Marletezofolyamat, Pozitivegeszek** A *halozat* változónak új értéket ad a paraméterben szereplő változók szerint. Ha a méret 2-nél kisebb, *Legalabb2folyamat* kivételt dob. Ha két azonos azonosítójú folyamat szerepel a tömbben, akkor *Marletezofolyamat* kivételt dob. Ha van nem pozitív egész számú azonosítójú folyamat, akkor *Pozitivegeszek* kivételt dob.

**private void setMenuBar()** Létrehozza a menürendszert.

**private void setMenuBarNevek()** Nyelv változtatása esetén beállítja a menük elnevezéseit.

**public void setMeretek(ArrayList<Integer> meretek)** Paraméterben szereplő listát adja értékül a *futasszamnak*.

**private void setPanel()** Létrehozza a panelt és a többi komponenst és elhelyezi azokat.

**public void setRajzTorol()** A *rajz* objektum változóinak törli a tartalmát.

**public void setTablázatTorol()** A *tablázat* objektum tartalmát törli.

**public void setUzenetHozzaad(String uzenet)** Hozzáadja a paraméterben szereplő szöveget az *uzenet* szöveges mezőhöz.

**public void setUzenetSzam(String uzenetszam)** Az *uzenetszam* szöveges mező szövegét a paraméterben kapott szövegre állítja.

**public void setUzenetTorol()** Törli a szöveget az *uzenet* szöveges mezőről.

**private void szovegesMegjelenites()** Szöveges megjelenítéshez szükséges elemeket helyezi a panelre.

## Ujhalozat osztály

Új hálózat adatainak bevitelére szolgáló ablak.

Változói:

**Foablak fa** A *foablakra* hivatkozó változó.

**static final long serialVersionUID** A JFrame osztály statikus változója.

**SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

**JButton megsem, rendben** Nyomógombokat megvalósító változók.

**JComboBox halozattipusszoveg** Választó mezőt megvalósító változó.

**JLabel folyamatazon, halozatmerete, halozatneve, halozattipus** Szöveges kiírásokat megvalósító változók.

**JPanel cimpanel, cimszovegpanel, gombpanel, szovegpanel** Paneleket megvalósító változók.

**JTextField folyamatazonsoveg, halozatmeretesoveg, halozatnevesoveg** Szöveges beviteli mezőket megvalósító változók.

Konstruktor:

**public Ujhalozat(Foablak fa)** Létrehoz egy új *Ujhalozat* objektumot. A paraméterben a *Foablakot* kell megadni.

Függvényei:

**public void actionPerformed(ActionEvent e)** Az *ActionListener* osztály kötelező eljárása. Az események kezelésére szolgál.

**public void ujhalozatKirajzolas()** Kirajzolja az *Ujhalozatot*.

## **Ujhalozatelek osztály**

Általános hálózat esetében a folyamatokból kimenő élek bevitelére szolgáló ablak.

**Foablak fa** A *Foablakra* hivatkozó változó.

**ArrayList<Integer> folyamatazon** Létrehozandó *Halozat* folyamatainak az azonosítóit tároló lista.

Változói:

**String nev** Létrehozandó új *Halozat* neve.

**static final long serialVersionUID** A *JFrame* osztály statikus változója.

**SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

**JButton megsem, rendben** Nyomógombokat megvalósító változók.

**JLabel[] cimiek** Szöveges kiírásokat megvalósító változók tömbje.

**JPanel cimpanel, cimszovegpanel, gombpanel, szovegpanel** Paneleket megvalósító változók.

**JTextField[] elek** Szöveges beviteli mezőket megvalósító változók tömbje.

Konstruktor:

**public Ujhalozatelek(Foablak fa, String nev, ArrayList<Integer> folyamatazon)** Létrehoz egy új *Ujhalozatelek* objektumot. A paraméterben a *Foablakot*, a létrehozandó *Halozat* nevét és folyamatai azonosítóinak listáját kell megadni.

Függvényei:

**public void actionPerformed(ActionEvent e)** Az *ActionListener* osztály kötelező eljárása. Az események kezelésére szolgál.

**public void ujhalozatelekKirajzolas()** Kirajzolja az *Ujhalozateleket*.

## **Elemzes1 osztály**

Az elemzés nézet és összes elemzés menüpontokhoz szolgáló adatok bevitelére szolgáló első ablak.

Változói:

**Foablak fa** A *Foablakra* hivatkozó változó.

**static final long serialVersionUID** A *JFrame* osztály statikus változója.

**SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

**JButton megsem, rendben** Nyomógombokat megvalósító változók.

**JLabel futasszam, kulmeretszam** Szöveges kiírásokat megvalósító változók.

**JPanel cimpanel, cimszovegpanel, gombpanel, szovegpanel** Paneleket megvalósító változók.

**JTextField futasszamszoveg, kulmeretszamszoveg** Szöveges beviteli mezőket megvalósító változók.

Konstruktor:

**Public Elemzes1(Foablak fa)** Létrehoz egy új *Elemzes1* objektumot. A paraméterben a *Foablakot* kell megadni.

Függvényei:

**public void actionPerformed(ActionEvent e)** Az *ActionListener* osztály kötelező eljárása. Az események kezelésére szolgál.

**public void elemzes1Kirajzolas()** Kirajzolja az *Elemzes1* ablakot.

## Elemzes2 osztály

Az elemzés nézet és összes elemzés menüpontokhoz szolgáló adatok bevitelére szolgáló második ablak.

Változói:

**Foablak fa** A *Foablak*ra hivatkozó változó.

**Integer kulmeretek** Különböző méretű hálózatok száma.

**static final long serialVersionUID** A *JFrame* osztály statikus változója.

**SajatResourceBundle srb** Szövegek megjelenítéséért felelős osztály példányát tárolja.

**JButton megsem, rendben** Nyomógombokat megvalósító változók.

**JLabel[] cimiek** Szöveges kiírásokat megvalósító változók tömbje.

**JPanel cimpanel, cimszovegpanel, gombpanel, szovegpanel** Paneleket megvalósító változók.

**JTextField[] folyamatszam** Szöveges beviteli mezőket megvalósító változók tömbje.

Konstruktor:

**public Elemzes2(Foablak fa, Integer kulmeretek)** Létrehoz egy új *Elemzes2* objektumot. A paraméterben a *Foablak*ot és a különböző méretű *Halozatok* számát kell megadni.

Függvényei:

**public void actionPerformed(ActionEvent e)** Az *ActionListener* osztály kötelező eljárása. Az események kezelésére szolgál.

**public void elemzes2Kirajzolas()** Kirajzolja az *Elemzes2* ablakot.

## Rajz osztály

A grafikus megjelenítéshez szükséges osztály.

Változói:

**static final long serialVersionUID** A *JFrame* osztály statikus változója.

**private volatile int meretx** A komponens x irányú méretének állítására szolgáló változó.

**private volatile int merety** A komponens y irányú méretének állítására szolgáló változó.

**ArrayList<Long[]> azonositok** Folyamatok azonosítóit és azok elhelyezését tároló lista.

**ArrayList<Ellipse2D> korok** Folyamatokat szimbolizáló köröket tároló lista.

**ArrayList<ArrayList<Line2D>> nyilakaltalanos** Üzeneteket szimbolizáló nyilakat tároló lista általános hálózat esetében.

**ArrayList<Line2D[]> nyilakbelul** Üzeneteket szimbolizáló nyilakat tároló lista gyűrű esetében.

**ArrayList<Line2D[]> nyilakkivul** Üzeneteket szimbolizáló nyilakat tároló lista gyűrű esetében.

**ArrayList<Line2D> osszekotok** Gyűrű éleit szimbolizáló szakaszokat tároló lista.

**ArrayList<ArrayList<ArrayList<Line2D>>> osszekotokaltalanos** Általános hálózat éleit szimbolizáló szakaszokat tároló lista.

Konstruktor:

**public Rajz()** Létrehoz egy új *Rajz* objektumot.

Függvényei:

**public void addAzonositok(Long[] azonosito)** Paraméterben kapott értéket hozzáadja az *azonositok* listához.

**public void addKorok(Ellipse2D kor)** Paraméterben kapott értéket hozzáadja a *korok* listához.

**public void addNyilakAltalanos(ArrayList<Line2D> nyilaltalanos)** Paraméterben kapott értéket hozzáadja a *nyilakaltalanos* listához.

**public void addNyilakbelul(Line2D[] nyil)** Paraméterben kapott értéket hozzáadja a *nyilakbelul* listához.

**public void addNyilakkivul(Line2D[] nyil)** Paraméterben kapott értéket hozzáadja a *nyilakkivul* listához.

**public void addOsszekotok(Line2D osszekoto)** Paraméterben kapott értéket hozzáadja az *osszekotok* listához.

**public void addOsszekotokaltalanos(ArrayList<ArrayList<Line2D>> osszekotoaltalanos)** A paraméterben kapott értéket hozzáadja az *osszekotokaltalanos* listához.

**public int getMeretx()** Visszaadja a *meretx* változó értékét.

**public int getMerety()** Visszaadja a *merety* változó értékét.

**public void paint(Graphics g)** A *paint* függvény felüldefiniálása.

**public void removeAllAzonositok()** Törli minden elemét az *azonositok* listának.

**public void removeAllKorok()** Törli minden elemét a *korok* listának.

**public void removeAllNyilakAltalanos()** Törli minden elemét a *nyilakaltalanos* listának.

**public void removeAllNyilakbelul()** Törli minden elemét a *nyilakbelul* listának.

**public void removeAllNyilakkivul()** Törli minden elemét a *nyilakkivul* listának.

**public void removeAllOsszekotok()** Törli minden elemét az *osszekotok* listának.

**public void removeAllOsszekotokaltalanos()** Törli minden elemét az *osszekotokaltalanos* listának.

**public void setMeretx(int x)** Beállítja a *meretx* értékét *xre*.

**public void setMerety(int y)** Beállítja a *merety* értékét *yra*.

**public void setSize(Dimension arg0)** A *setSize(Dimension arg0)* függvény felüldefiniálása.

**public void setSize(int arg0, int arg1)** A *setSize(int arg0, int arg1)* függvény felüldefiniálása.

## XMLszuro osztály

Fájl mentése és betöltése esetén a fájlválasztó ablakban megjelenő fájlok szűrésére szolgáló osztály.

Függvényei:

**public boolean accept(File f)** Igazzal tér vissza, ha egy könyvtári elem megjeleníthető a fájlválasztó ablakban. A könyvtárak és az xml kiterjesztésű fájlok ezek.

**public String getDescription()** A fájlválasztó ablakban visszatérési értéke lesz a szűrő neve.

## Feliratok csomag

### SajatResourceBundle osztály

A környezeti nyelvi beállítások tárolására szolgáló osztály.

Változója:

**static ResourceBundle rb** Környezeti beállításokat tároló változó.

Konstruktor:

**public SajatResourceBundle()** Létrehoz egy új *SajatResourceBundle* objektumot.

Függvényei:

**public void setRB(String o1, String o2)** Beállítja az *rb* változó értékét a paraméterben kapottak szerint.

**public ResourceBundle getRB()** visszaadja az *rb* változó értékét.

## **Feliratok\_hu osztály**

A magyar nyelvű feliratok tárolására szolgáló osztály.

Konstansa:

**static final Object[][] contents** Szöveg azonosítóját és szöveget tartalmazó tömbök tömbje.

Függvénye:

**public Object[][] getContents()** Visszaadja a *contents* változó értékét.

## **Feliratok\_en osztály**

A angol nyelvű feliratok tárolására szolgáló osztály.

Konstansa:

**static final Object[][] contents** Szöveg azonosítóját és szöveget tartalmazó tömbök tömbje.

Függvénye:

**public Object[][] getContents()** Visszaadja a *contents* változó értékét.

## **Kivetelek csomag**

### **Elnemletezofolyamathoz osztály**

Kivételt megvalósító osztály. Ha egy folyamatból általános hálózatban érvénytelen folyamathoz vezet él, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Elnemletezofolyamathoz()** Létrehoz egy új *Elnemletezofolyamathoz* objektumot.

### **Hurokel osztály**

Kivételt megvalósító osztály. Ha egy folyamatból általános hálózatban önmagába mutat él, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Hurokel()** Létrehoz egy új *Hurokel* objektumot.

### **Legalabb2folyamat osztály**

Kivételt megvalósító osztály. Ha egy hálózatban kettőnél kevesebb folyamat van, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Legalabb2folyamat()** Létrehoz egy új *Hurokel* objektumot.



### **Marletezofolyamat osztály**

Kivételt megvalósító osztály. Ha egy hálózatban két azonos azonosítójú folyamat van, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Marletezofolyamat()** Létrehoz egy új *Marletezofolyamat* objektumot.

### **Nemosszefuggoaltalanos osztály**

Kivételt megvalósító osztály. Ha egy általános hálózat nem erősen összefüggő, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Nemosszefuggoaltalanos()** Létrehoz egy új *Nemosszefuggoaltalanos* objektumot.

### **Nemosszefuggogyuru osztály**

Kivételt megvalósító osztály. Ha egy gyűrűnek nem megfelelően van megadva a szomszédossági mátrixszo, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Nemosszefuggogyuru()** Létrehoz egy új *Nemosszefuggogyuru* objektumot.

### **Pozitivegeszek osztály**

Kivételt megvalósító osztály. Ha egy folyamatnak nem pozitív egész az azonosítónak, akkor váltódik ki.

Konstansa:

**static final long serialVersionUID** A Exception osztály statikus változója.

Konstruktor:

**public Pozitivegeszek()** Létrehoz egy új *Pozitivegeszek* objektumot.

## **Szimulacio csomag**

### **Szimulacio osztály**

A program *main* függvényét megvalósító osztály. Ezt az osztályt kell meghívni a program indításához.

Függvénye:

**public static void main(String[] args)** A program *main* függvénye.

## Tesztelés:

A teszteléshez használt hálózatok elmentve megtalálhatóak a gyuruteszt.xml és az altalanosteszt.xml fájlokban.

### A Futtatás menü HS menü Félduplex menüpontja

Csak üzenetszám nézet esetén:

A gyűrű1 hálózat esetében:

```
HS Félduplex algoritmus
Összes üzenetek száma: 49
A lépések száma: 41
```

A gyűrű2 hálózat esetében:

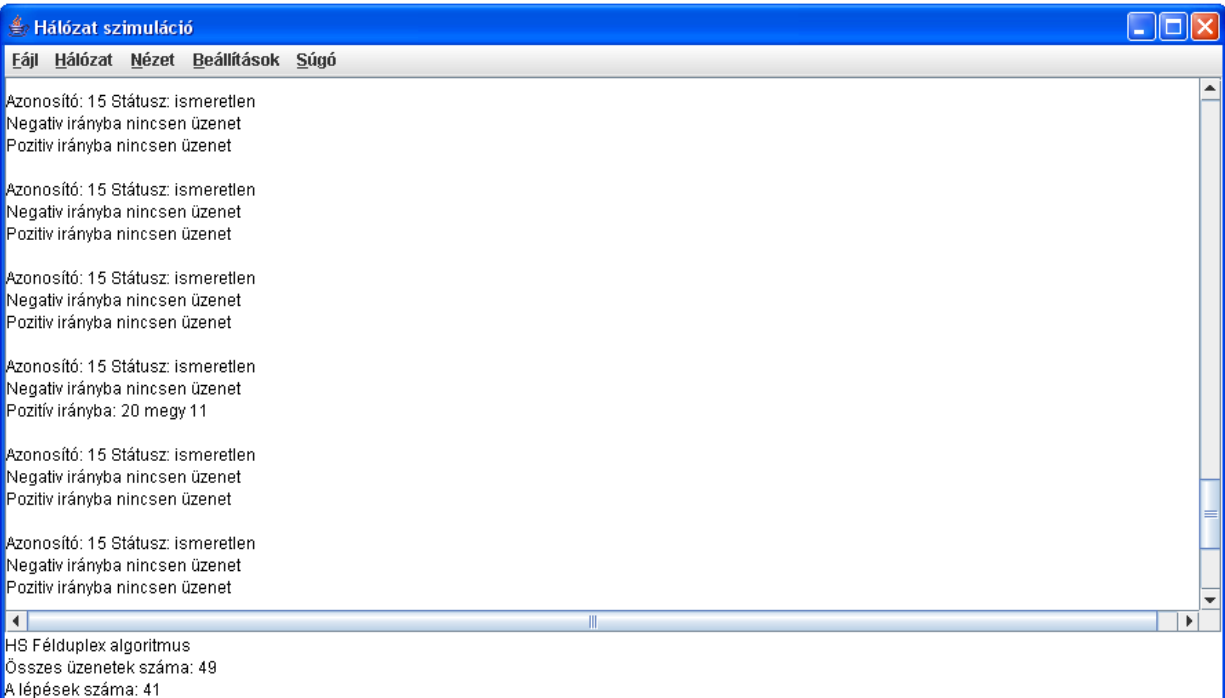
```
HS Félduplex algoritmus
Összes üzenetek száma: 65
A lépések száma: 41
```

A gyűrű3 hálózat esetében:

```
HS Félduplex algoritmus
Összes üzenetek száma: 73
A lépések száma: 41
```

Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:

A gyűrű1 hálózat esetében:



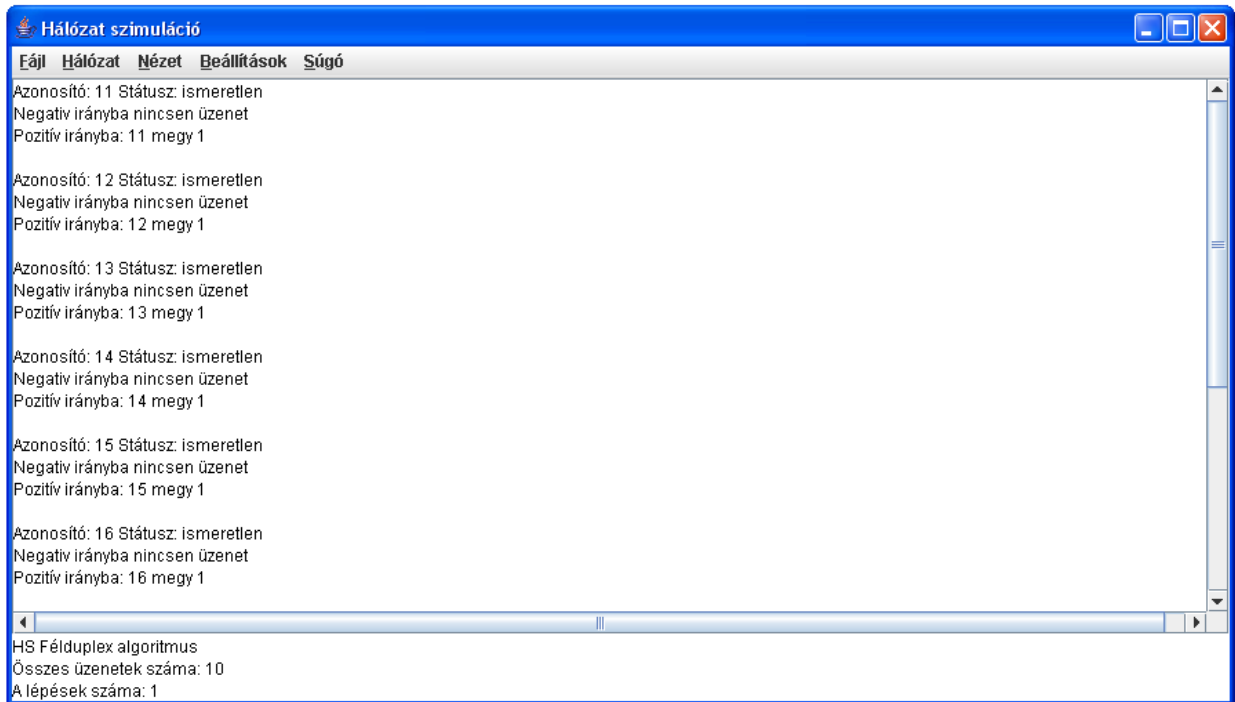
A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű1 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:

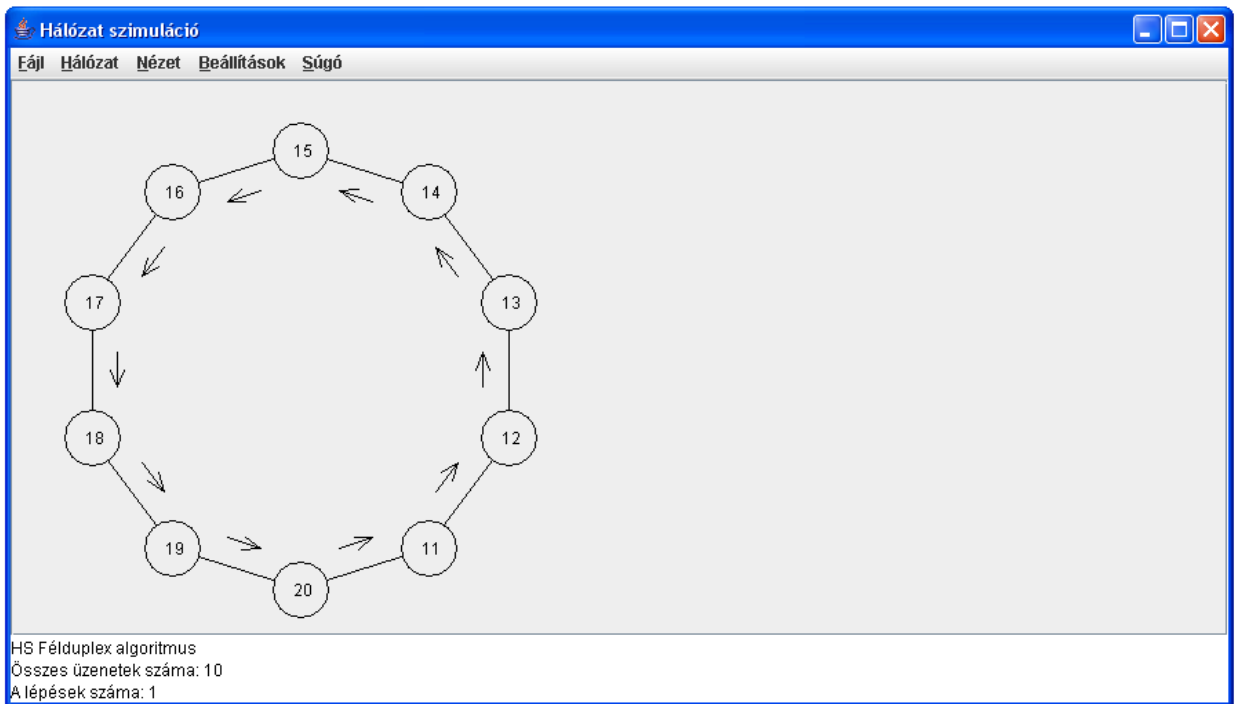


Az algoritmus hibamentesen lefutott.  
A gyűrű2 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű1 hálózat esetében:

Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.

A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50, 100 és 500.

A gyűrű1 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Félduplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	23	27.0	32
Konstans szorzó	4.0	4.0	4.0	1.9811121	2.3256533	2.75633
10 folyamat	41	41.0	41	49	69.0	80
Konstans szorzó	4.1	4.1	4.1	1.475047	2.077107	2.40824
50 folyamat	177	177.0	177	447	503.0	575
Konstans szorzó	3.54	3.54	3.54	1.5840234	1.7824693	2.0376139
100 folyamat	355	355.0	355	1002	1184.0	1474
Konstans szorzó	3.55	3.55	3.55	1.5081602	1.7820976	2.218591
500 folyamat	1523	1523.0	1523	6931	7680.0	8555
Konstans szorzó	3.046	3.046	3.046	1.5461001	1.7131797	1.9083662

A gyűrű2 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Félduplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	23	27.0	32
Konstans szorzó	4.0	4.0	4.0	1.9811121	2.3256533	2.75633
10 folyamat	41	41.0	41	57	72.0	83
Konstans szorzó	4.1	4.1	4.1	1.715871	2.1674159	2.498549
50 folyamat	177	177.0	177	437	506.0	607
Konstans szorzó	3.54	3.54	3.54	1.5485866	1.7931002	2.1510115
100 folyamat	355	355.0	355	1000	1161.0	1328
Konstans szorzó	3.55	3.55	3.55	1.50515	1.7474791	1.9988391
500 folyamat	1523	1523.0	1523	6957	7670.0	8599
Konstans szorzó	3.046	3.046	3.046	1.5518999	1.7109491	1.9181813

A gyűrű3 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Félduplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	23	27.0	32
Konstans szorzó	4.0	4.0	4.0	1.9811121	2.3256533	2.75633
10 folyamat	41	41.0	41	54	70.0	83
Konstans szorzó	4.1	4.1	4.1	1.625562	2.10721	2.498549
50 folyamat	177	177.0	177	432	504.0	631
Konstans szorzó	3.54	3.54	3.54	1.5308682	1.7860129	2.23606
100 folyamat	355	355.0	355	1044	1185.0	1392
Konstans szorzó	3.55	3.55	3.55	1.5713766	1.7836027	2.0951688
500 folyamat	1523	1523.0	1523	6716	7580.0	8623
Konstans szorzó	3.046	3.046	3.046	1.49814	1.6908727	1.923535

Az elemzésből látszik, hogy a gyűrű1 a HS Félduplex algoritmus számára egy jó bemenet. A véletlenszerűen előállított hálózatok között sem volt olyan, amelyik kevesebb üzenetet küldött volna. A gyűrű2 és gyűrű3 üzenetszáma az átlagos üzenetszámhoz van közel. Látható, hogy a konstans szorzók értéke csökken a hálózat méretének növekedésével a lépésszámnál és az üzenetszámnál is.

## A Futtatás menü HS menü Duplex menüpontja

Csak üzenetszám nézet esetén:

A gyűrű1 hálózat esetében:

```
HS Duplex algoritmus
Összes üzenetek száma: 114
A lépések száma: 41
```

A gyűrű2 hálózat esetében:

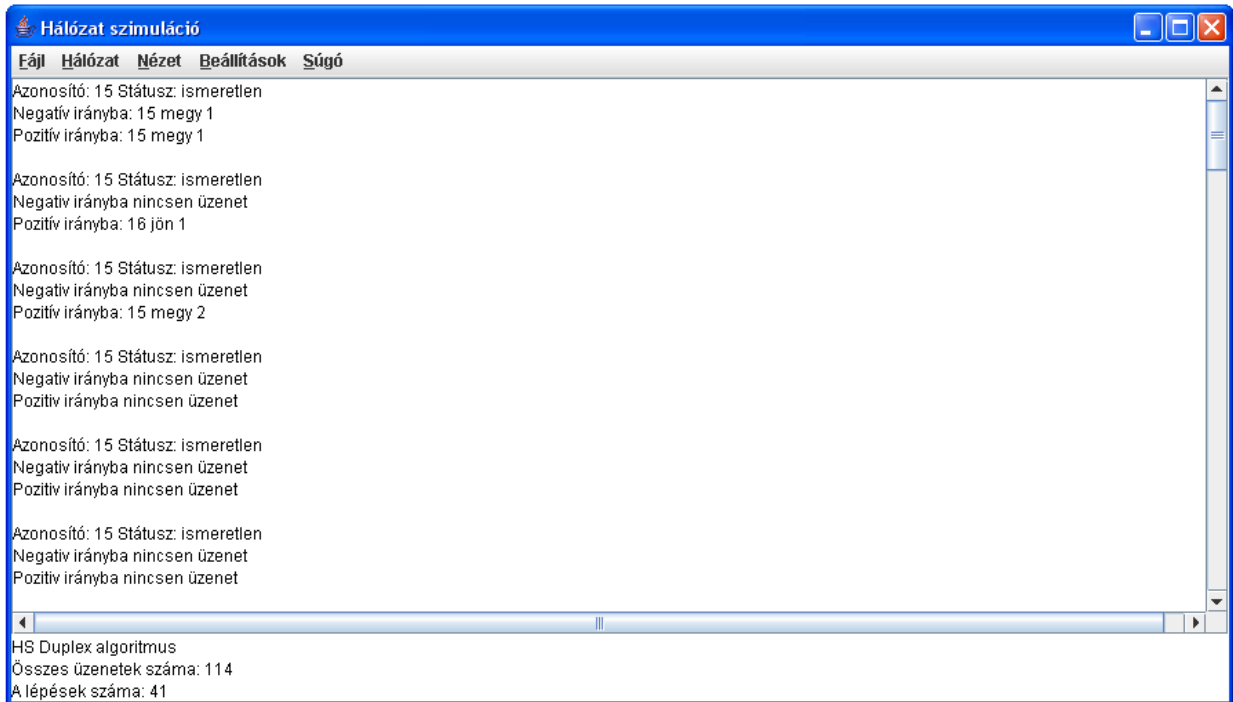
```
HS Duplex algoritmus
Összes üzenetek száma: 114
A lépések száma: 41
```

A gyűrű3 hálózat esetében:

```
HS Duplex algoritmus
Összes üzenetek száma: 158
A lépések száma: 41
```

Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:

A gyűrű1 hálózat esetében:



Az algoritmus hibamentesen lefutott.

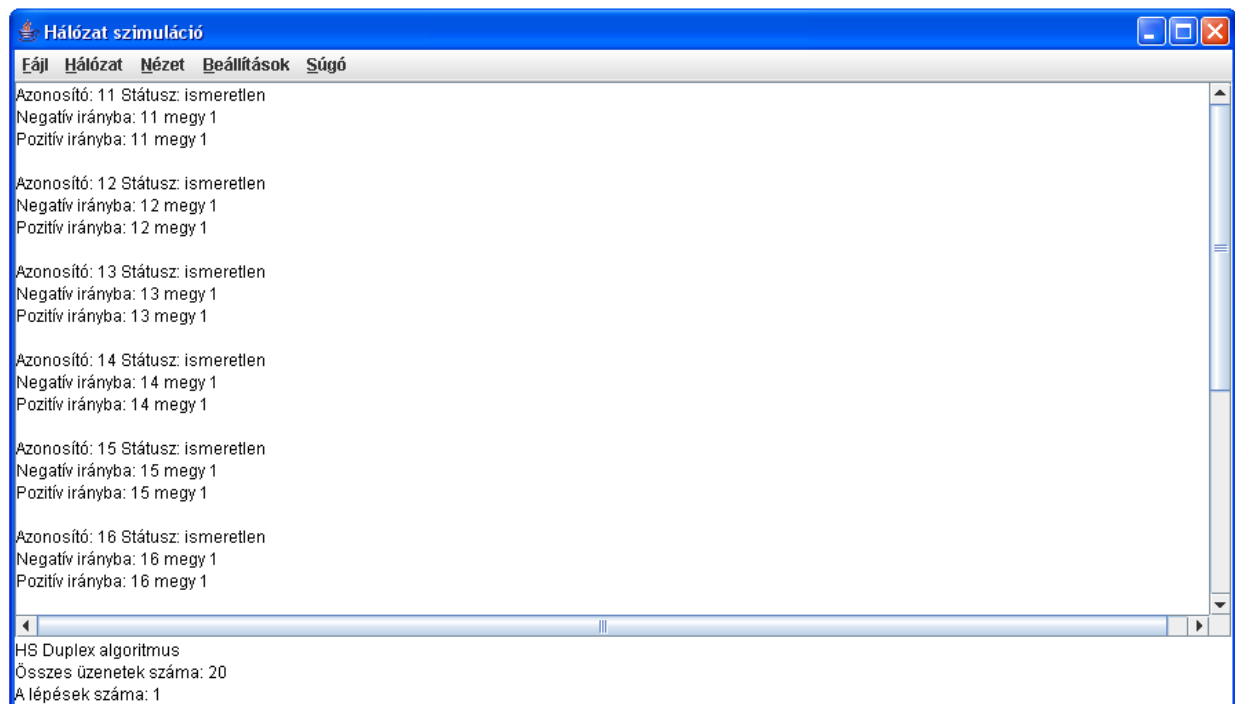
A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű1 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:

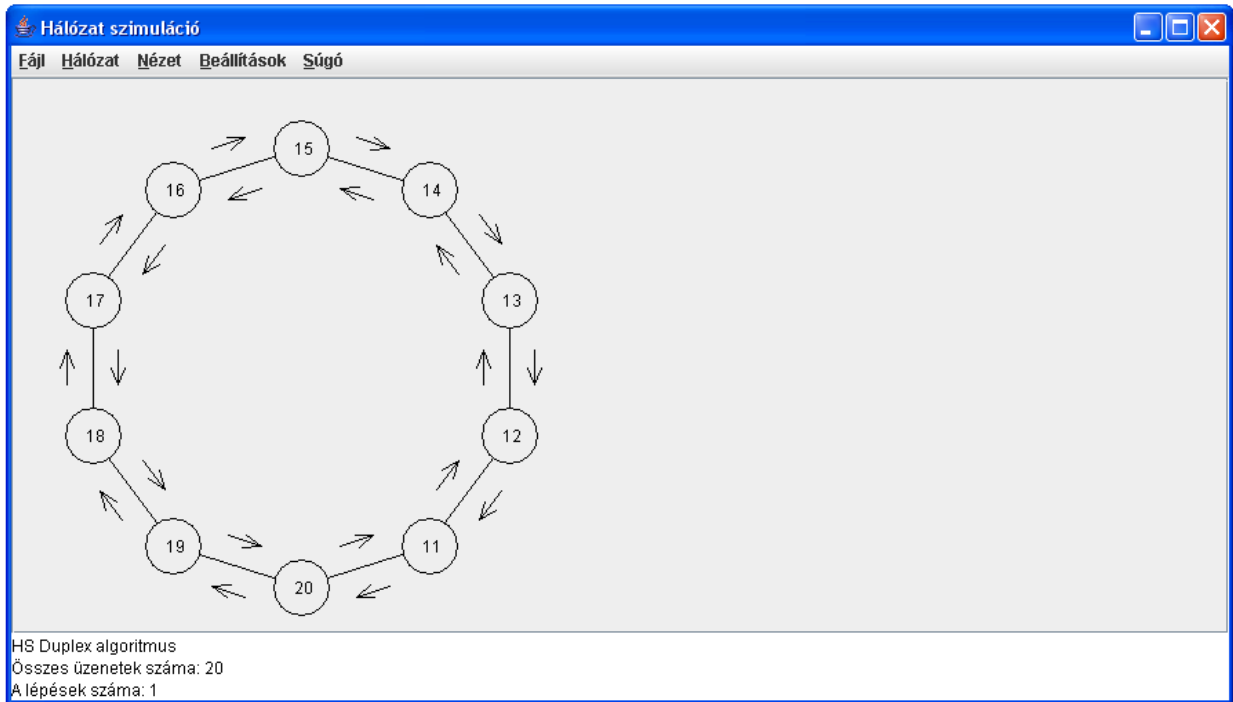


Az algoritmus hibamentesen lefutott.  
A gyűrű2 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű1 hálózat esetében:

Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.

A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.



Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50, 100 és 500.

A gyűrű1 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Duplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	52	55.0	58
Konstans szorzó	4.0	4.0	4.0	4.4790363	4.737442	4.995848
10 folyamat	41	41.0	41	114	140.0	158
Konstans szorzó	4.1	4.1	4.1	3.431742	4.21442	4.7562737
50 folyamat	177	177.0	177	936	1010.0	1080
Konstans szorzó	3.54	3.54	3.54	3.3168812	3.5791132	3.8271706
100 folyamat	355	355.0	355	2095	2358.0	2581
Konstans szorzó	3.55	3.55	3.55	3.1532893	3.5491436	3.884792
500 folyamat	1523	1523.0	1523	13946	15277.0	16213
Konstans szorzó	3.046	3.046	3.046	3.110938	3.4078445	3.6166384

A gyűrű2 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Duplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	52	55.0	58
Konstans szorzó	4.0	4.0	4.0	4.4790363	4.737442	4.995848
10 folyamat	41	41.0	41	114	140.0	158
Konstans szorzó	4.1	4.1	4.1	3.431742	4.21442	4.7562737
50 folyamat	177	177.0	177	891	1010.0	1146
Konstans szorzó	3.54	3.54	3.54	3.1574156	3.5791132	4.0610533
100 folyamat	355	355.0	355	2128	2358.0	2594
Konstans szorzó	3.55	3.55	3.55	3.202959	3.5491436	3.904359
500 folyamat	1523	1523.0	1523	13872	15267.0	16174
Konstans szorzó	3.046	3.046	3.046	3.094431	3.405614	3.6079388

A gyűrű3 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások	Súgó		
HS Duplex	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	20	20.0	20	52	55.0	58
Konstans szorzó	4.0	4.0	4.0	4.4790363	4.737442	4.995848
10 folyamat	41	41.0	41	114	141.0	158
Konstans szorzó	4.1	4.1	4.1	3.431742	4.244523	4.7562737
50 folyamat	177	177.0	177	868	1014.0	1144
Konstans szorzó	3.54	3.54	3.54	3.075911	3.593288	4.0539656
100 folyamat	355	355.0	355	2165	2359.0	2555
Konstans szorzó	3.55	3.55	3.55	3.2586496	3.5506487	3.8456583
500 folyamat	1523	1523.0	1523	14032	15244.0	16340
Konstans szorzó	3.046	3.046	3.046	3.1301222	3.4004834	3.6449683

Az elemzésből látszik, hogy a gyűrű1 és a gyűrű2 a HS Duplex algoritmus számára egy jó bemenet. A véletlenszerűen előállított hálózatok között sem volt olyan, amelyik kevesebb üzenetet küldött volna. A gyűrű3 pedig egy rossz bemenet. A véletlenszerűen előállított hálózatok között sem volt olyan, amelyik több üzenetet küldött volna. Látható, hogy a konstans szorzók értéke csökken a hálózat méretének növekedésével a lépésszámnál és az üzenetszámnál is. De az üzenetszám konstans szorzója majdnem kétszerese a HS Félduplex algoritmushoz képest.

## A Futtatás menü Időszelet menüpontja

Csak üzenetszám nézet esetén:

A gyűrű3 hálózat esetében:

```
Időszelet algoritmus
Összes üzenetek száma: 10
A lépések száma: 111
```

A gyűrű4 hálózat esetében:

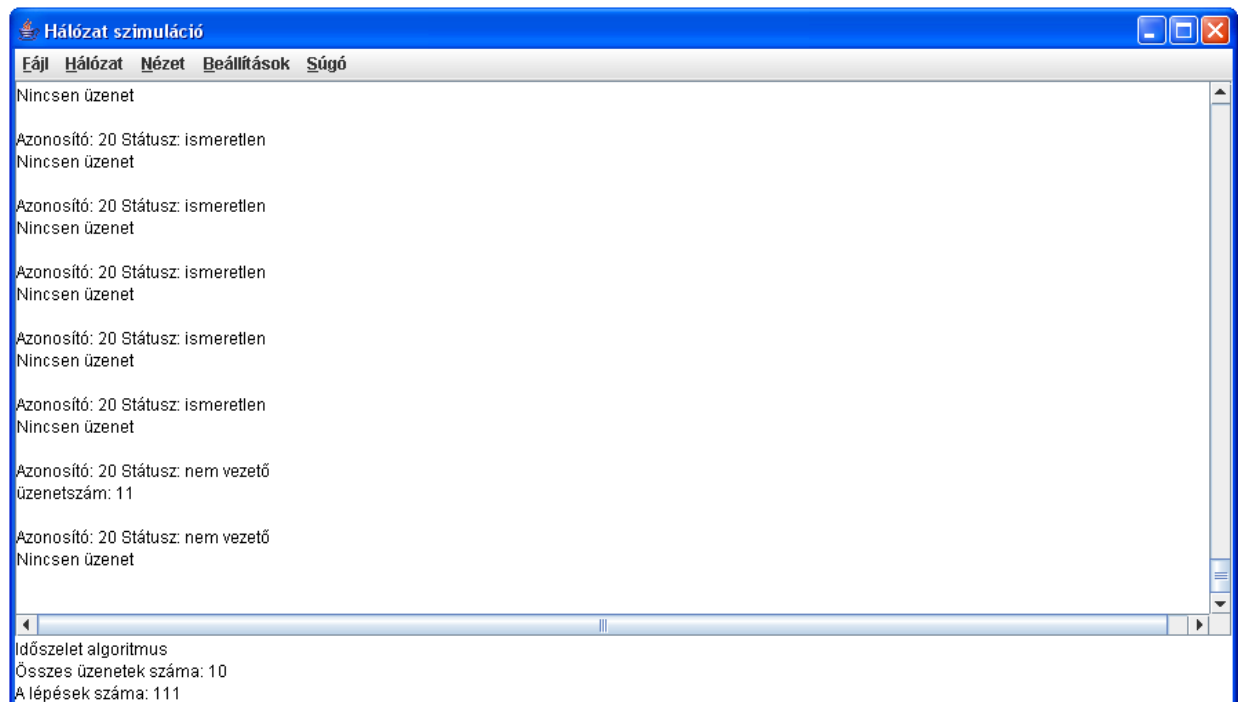
```
Időszelet algoritmus
Összes üzenetek száma: 10
A lépések száma: 11
```

A gyűrű5 hálózat esetében:

```
Időszelet algoritmus
Összes üzenetek száma: 10
A lépések száma: 911
```

Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:

A gyűrű3 hálózat esetében:



Az algoritmus hibamentesen lefutott.

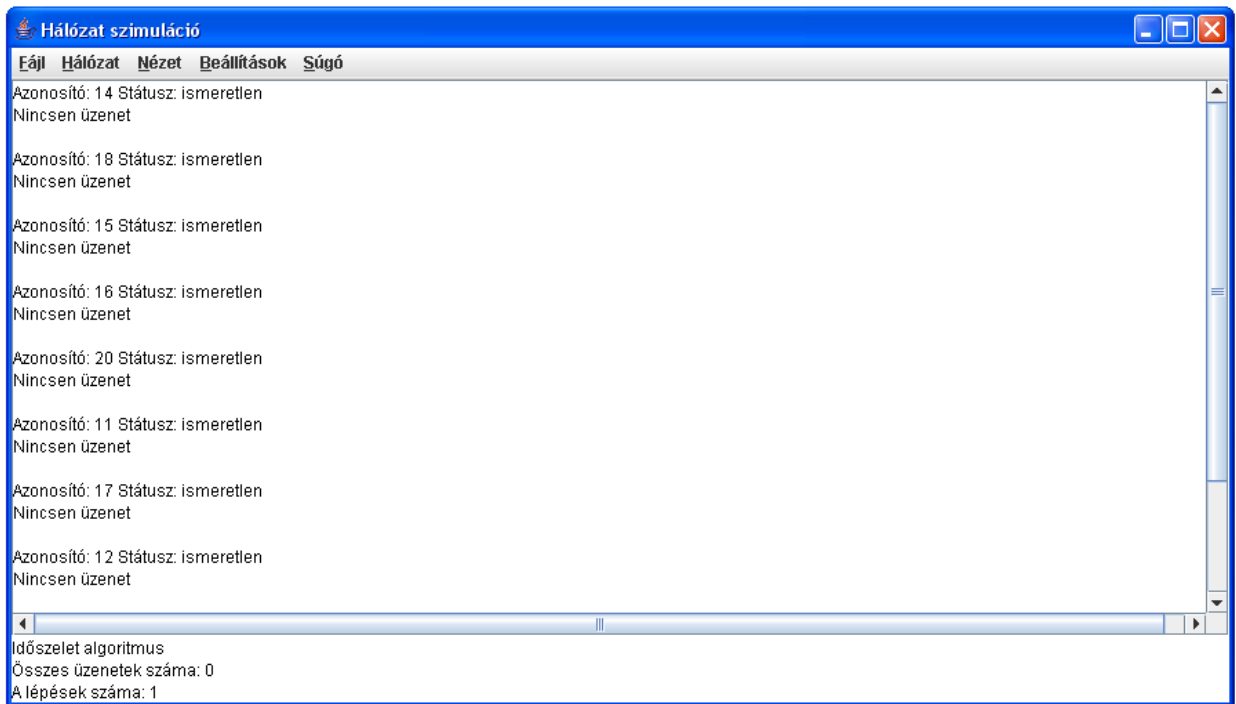
A gyűrű4 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű5 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű3 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:

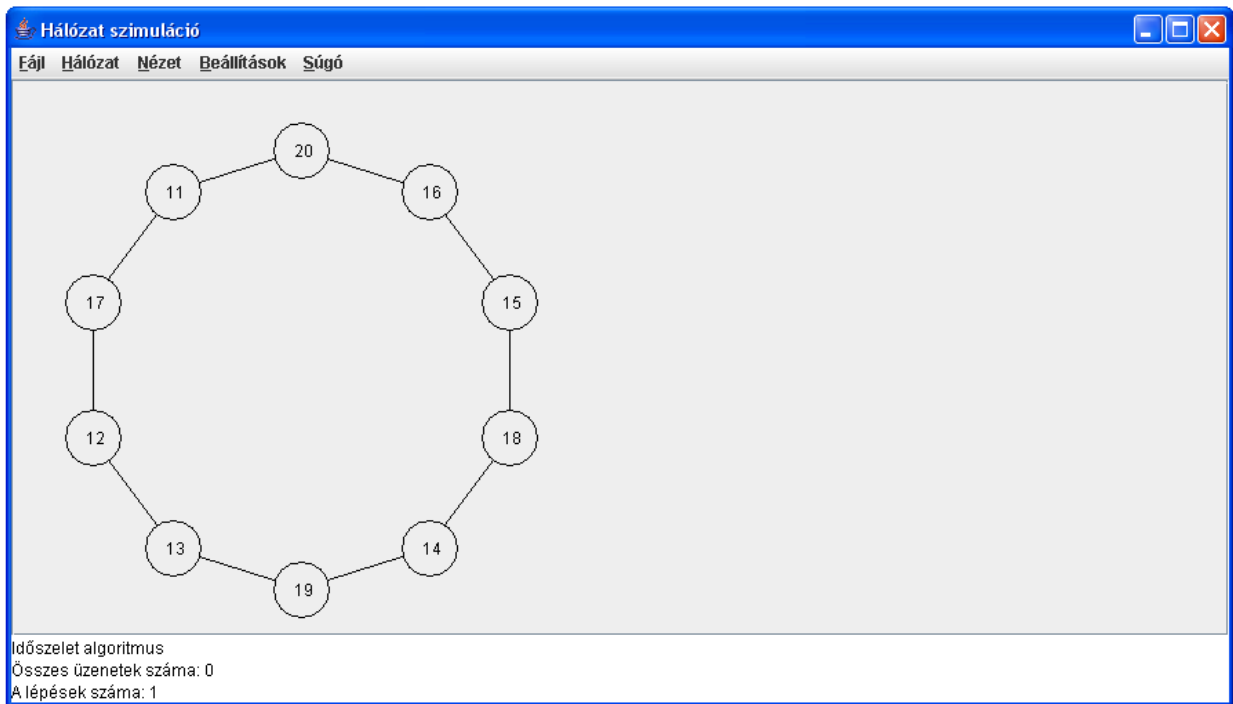


Az algoritmus hibamentesen lefutott.  
A gyűrű4 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
A gyűrű5 hálózat esetében:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű3 hálózat esetében:

Egy kép a futás közben küldött üzenetéről:



Az algoritmus hibamentesen lefutott.

A gyűrű4 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű5 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50, 100 és 500.

A gyűrű3 hálózat esetében:

Hálózat szimuláció							
Fájl	Hálózat	Nézet	Beállítások				Súgó
	Időszelet	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
	5 folyamat	6	16.0	26	5	5.0	5
	Konstans szorzó	1.2	3.2	5.2	1.0	1.0	1.0
	10 folyamat	11	52.0	111	10	10.0	10
	Konstans szorzó	1.1	5.2	11.1	1.0	1.0	1.0
	50 folyamat	51	1243.0	2501	50	50.0	50
	Konstans szorzó	1.02	24.86	50.02	1.0	1.0	1.0
	100 folyamat	101	5048.0	10001	100	100.0	100
	Konstans szorzó	1.01	50.48	100.01	1.0	1.0	1.0
	500 folyamat	4001	126545.0	244001	500	500.0	500
	Konstans szorzó	8.002	253.09	488.002	1.0	1.0	1.0

A gyűrű4 hálózat esetében:

Hálózat szimuláció							
Fájl	Hálózat	Nézet	Beállítások				Súgó
	Időszelet	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
	5 folyamat	6	16.0	26	5	5.0	5
	Konstans szorzó	1.2	3.2	5.2	1.0	1.0	1.0
	10 folyamat	11	55.0	101	10	10.0	10
	Konstans szorzó	1.1	5.5	10.1	1.0	1.0	1.0
	50 folyamat	51	1201.0	2501	50	50.0	50
	Konstans szorzó	1.02	24.02	50.02	1.0	1.0	1.0
	100 folyamat	201	5050.0	9901	100	100.0	100
	Konstans szorzó	2.01	50.5	99.01	1.0	1.0	1.0
	500 folyamat	501	119781.0	249501	500	500.0	500
	Konstans szorzó	1.002	239.562	499.002	1.0	1.0	1.0

A gyűrű5 hálózat esetében:

Hálózat szimuláció							
Fájl	Hálózat	Nézet	Beállítások				Súgó
	Időszelet	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
	5 folyamat	6	16.0	26	5	5.0	5
	Konstans szorzó	1.2	3.2	5.2	1.0	1.0	1.0
	10 folyamat	11	66.0	911	10	10.0	10
	Konstans szorzó	1.1	6.6	91.1	1.0	1.0	1.0
	50 folyamat	51	1323.0	2501	50	50.0	50
	Konstans szorzó	1.02	26.46	50.02	1.0	1.0	1.0
	100 folyamat	101	5308.0	10001	100	100.0	100
	Konstans szorzó	1.01	53.08	100.01	1.0	1.0	1.0
	500 folyamat	4501	124431.0	245501	500	500.0	500
	Konstans szorzó	9.002	248.862	491.002	1.0	1.0	1.0

Az elemzésből látszik, hogy gyűrű4 egy jó bemenet az Időszelet algoritmus számára. A véletlenszerűen előállított hálózatok között sem volt olyan, amelyik kevesebb üzenetet küldött volna. A gyűrű3 és a gyűrű5 pedig rossz bemenetek. A véletlenszerűen előállított hálózatok közül egyik sem tett meg több lépést (szándékosan így adtam meg őket). A lépésszám konstans szorzói a hálózat méretének és a lehetséges azonosítók értékének növekedésével folyamatosan növekszik. Ezért a gyakorlatban nem használják. Elméleti jelentősége van, hogy az üzenetszám konstans szorzója egy.

## A Futtatás menü LCR menüpontja

Csak üzenetszám nézet esetén:

A gyűrű1 hálózat esetében:

```
LCR algoritmus
Összes üzenetek száma: 19
A lépések száma: 11
```

A gyűrű2 hálózat esetében:

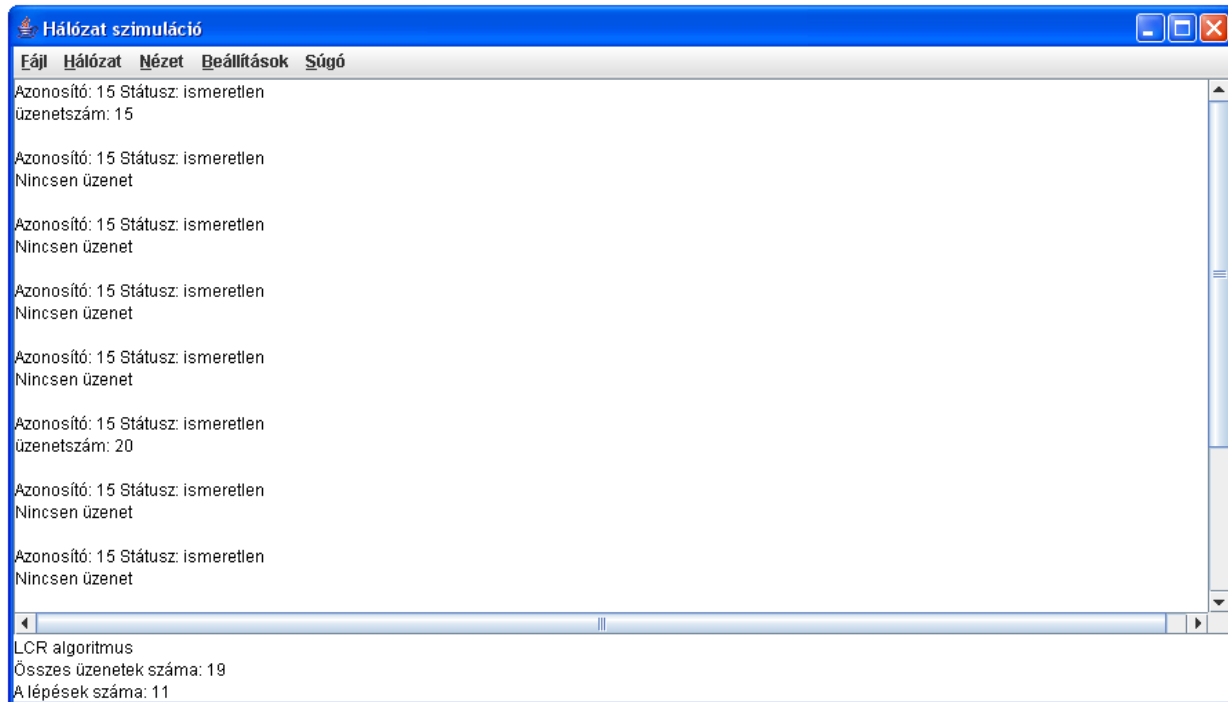
```
LCR algoritmus
Összes üzenetek száma: 55
A lépések száma: 11
```

A gyűrű3 hálózat esetében:

```
LCR algoritmus
Összes üzenetek száma: 27
A lépések száma: 11
```

Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:

A gyűrű1 hálózat esetében:



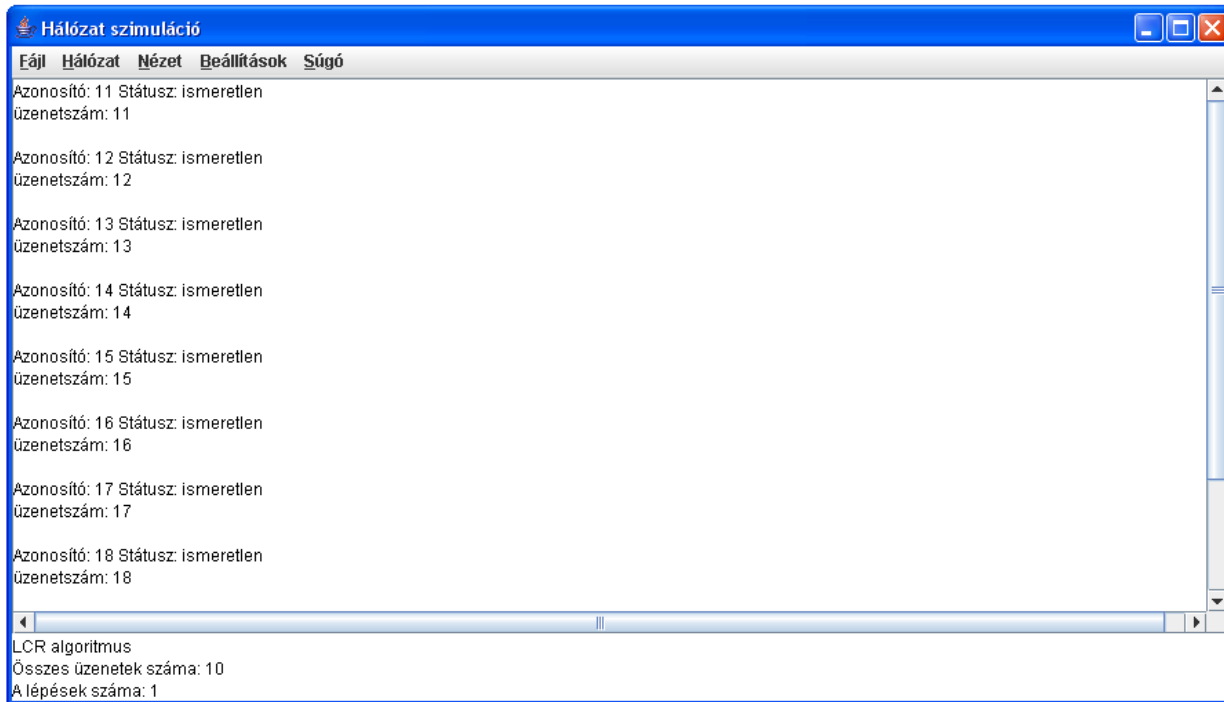
A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű1 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:

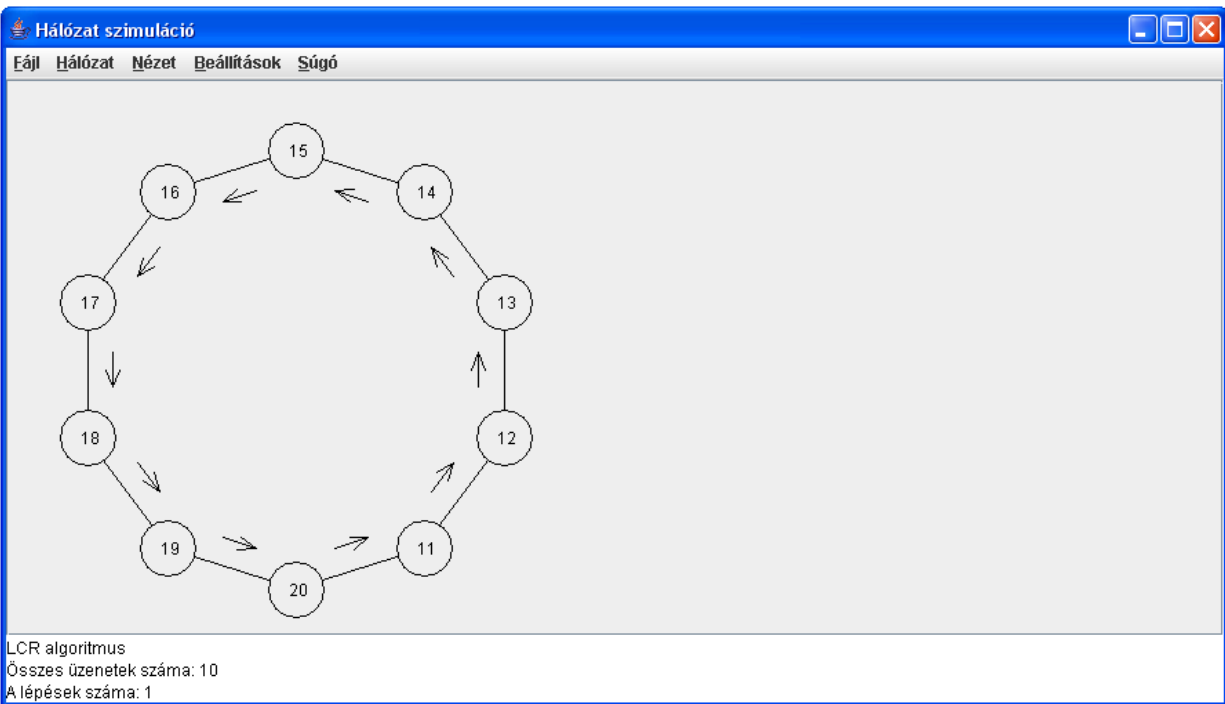


Az algoritmus hibamentesen lefutott.  
A gyűrű2 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű1 hálózat esetében:

Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.

A gyűrű2 hálózat esetében:

Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.



Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50, 100 és 500.

A gyűrű1 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
LCR	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	6	6.0	6	9	11.0	15
Konstans szorzó	1.2	1.2	1.2	0.36	0.44	0.6
10 folyamat	11	11.0	11	19	28.0	46
Konstans szorzó	1.1	1.1	1.1	0.19	0.28	0.46
50 folyamat	51	51.0	51	173	229.0	318
Konstans szorzó	1.02	1.02	1.02	0.0692	0.0916	0.1272
100 folyamat	101	101.0	101	390	512.0	683
Konstans szorzó	1.01	1.01	1.01	0.039	0.0512	0.0683
500 folyamat	501	501.0	501	2684	3380.0	4134
Konstans szorzó	1.002	1.002	1.002	0.010736	0.01352	0.016536

A gyűrű2 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
LCR	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	6	6.0	6	9	11.0	15
Konstans szorzó	1.2	1.2	1.2	0.36	0.44	0.6
10 folyamat	11	11.0	11	20	29.0	55
Konstans szorzó	1.1	1.1	1.1	0.2	0.29	0.55
50 folyamat	51	51.0	51	180	225.0	333
Konstans szorzó	1.02	1.02	1.02	0.072	0.09	0.1332
100 folyamat	101	101.0	101	415	520.0	732
Konstans szorzó	1.01	1.01	1.01	0.0415	0.052	0.0732
500 folyamat	501	501.0	501	2898	3391.0	4451
Konstans szorzó	1.002	1.002	1.002	0.011592	0.013564	0.017804

A gyűrű3 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
LCR	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	6	6.0	6	9	11.0	15
Konstans szorzó	1.2	1.2	1.2	0.36	0.44	0.6
10 folyamat	11	11.0	11	22	29.0	41
Konstans szorzó	1.1	1.1	1.1	0.22	0.29	0.41
50 folyamat	51	51.0	51	177	222.0	310
Konstans szorzó	1.02	1.02	1.02	0.0708	0.0888	0.124
100 folyamat	101	101.0	101	416	518.0	688
Konstans szorzó	1.01	1.01	1.01	0.0416	0.0518	0.0688
500 folyamat	501	501.0	501	2836	3393.0	4424
Konstans szorzó	1.002	1.002	1.002	0.011344	0.013572	0.017696

Az elemzésből látszik, hogy a gyűrű1 jó, a gyűrű3 átlagos, míg a gyűrű2 rossz bemenet az LCR algoritmus számára. A gyűrű1 üzenetszáma a többi elemzésben véletlenszerűen előállított hálózatok üzenetszámánál kisebb volt, még csak azonos sem. A gyűrű2 üzenetszáma a többi elemzésben véletlenszerűen előállított hálózatok üzenetszámánál nagyobb volt, még csak azonos sem. Az üzenetszám konstans szorzó a hálózat méretének növekedésével csökken.

## A Futtatás menü Maxterjed menüpontja

Csak üzenetszám nézet esetén:

A gyűrű1 hálózat esetében:

Maxterjed algoritmus  
Összes üzenetek száma: 100  
A lépések száma: 6

A gyűrű3 hálózat esetében:

Maxterjed algoritmus  
Összes üzenetek száma: 100  
A lépések száma: 6

Az általános1 hálózat esetén:

Maxterjed algoritmus  
Összes üzenetek száma: 162  
A lépések száma: 10

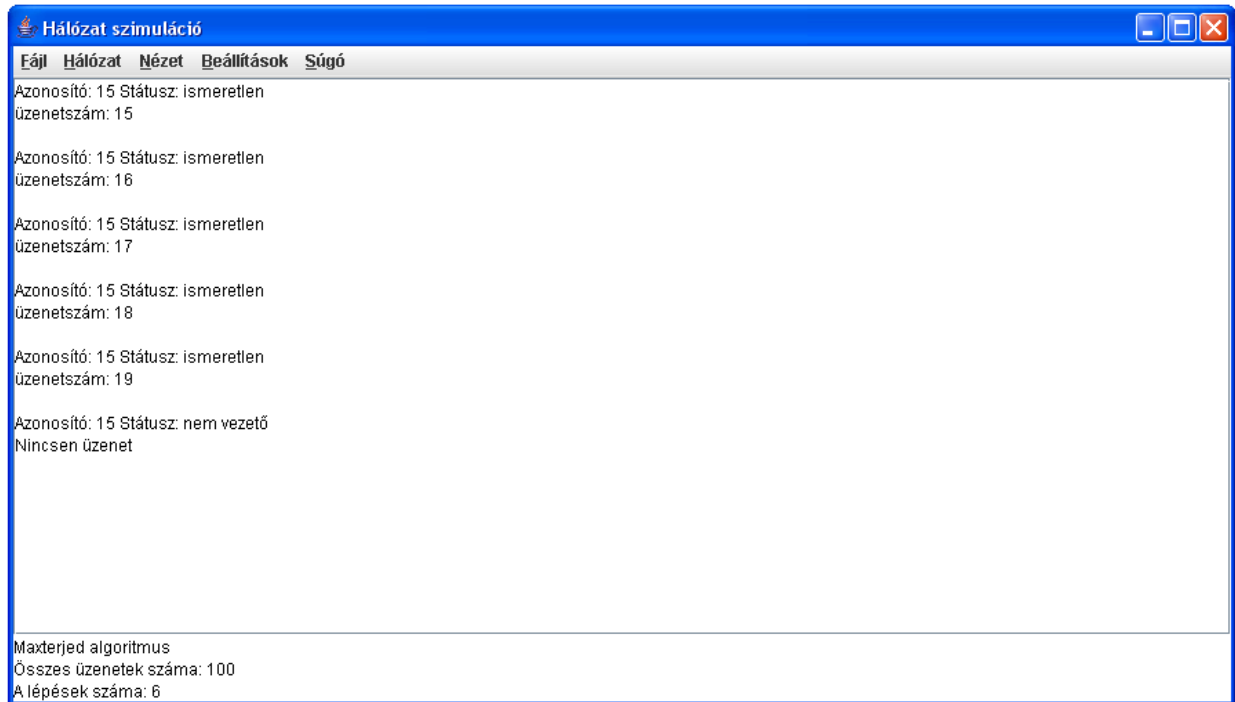
Az általános2 hálózat esetén:

Maxterjed algoritmus  
Összes üzenetek száma: 180  
A lépések száma: 7

Az általános2 hálózat esetén:

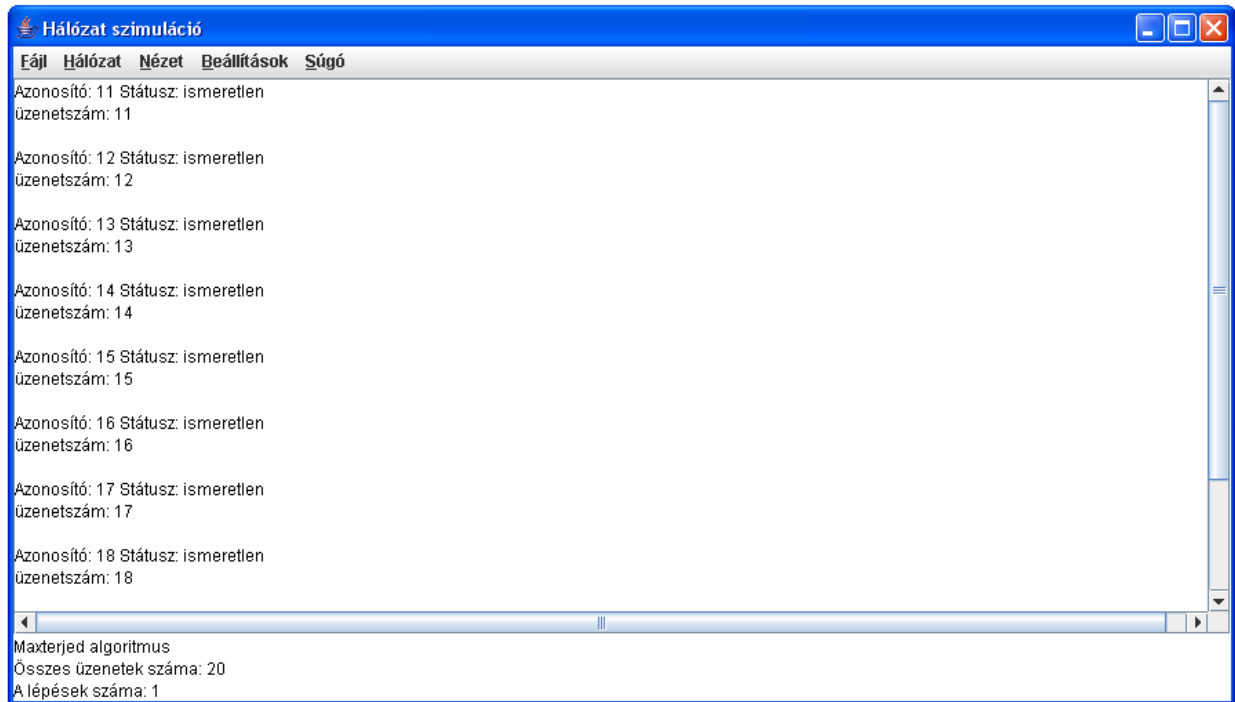
Maxterjed algoritmus  
Összes üzenetek száma: 90  
A lépések száma: 2

Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:  
A gyűrű1 hálózat esetében:



A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
Az általános1 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános2 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános3 hálózat esetén:  
Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű1 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.  
A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
Az általános1 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános2 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános3 hálózat esetén:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű1 hálózat esetében:

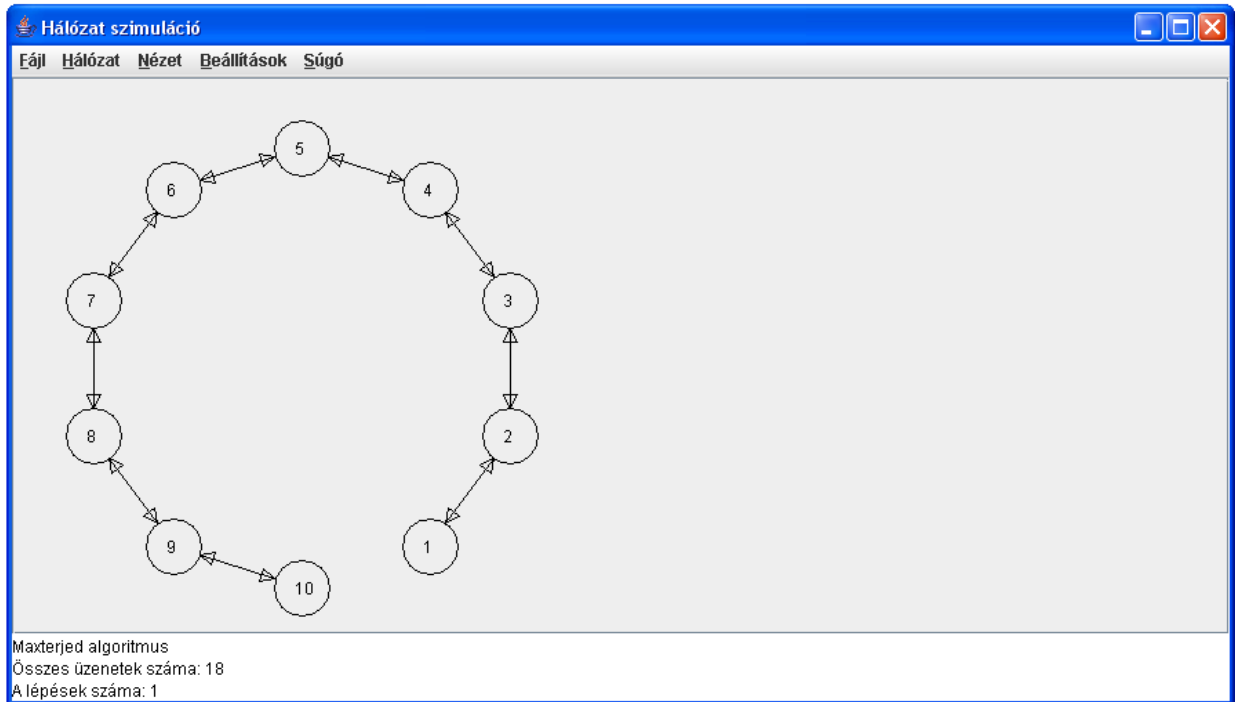
Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

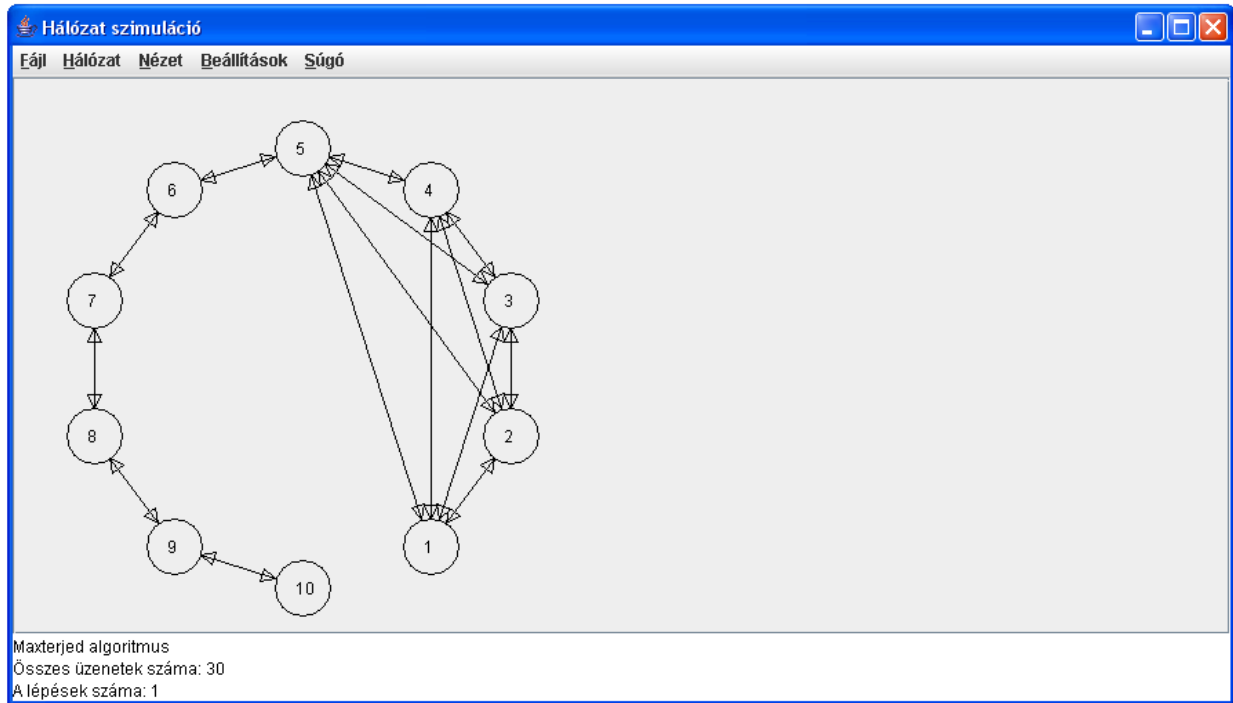
Az általános1 hálózat esetén:

Egy kép a futás közben küldött üzenetekről:

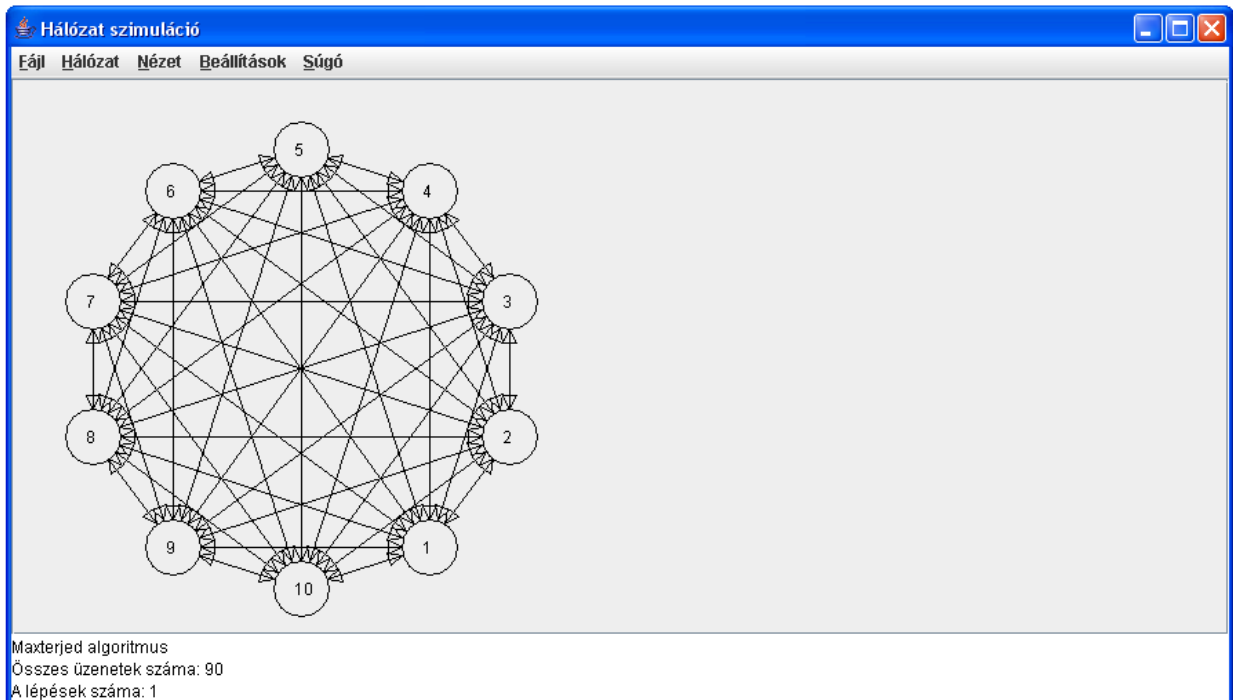


Az algoritmus hibamentesen lefutott.

Az általános2 hálózat esetén:  
Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.  
Az általános3 hálózat esetén:  
Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.

Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50, és 100.

A gyűrűl hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Maxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	4	4.0	4	30	30.0	30
Konstans szorzó	0.8	0.8	0.8	1.2	1.2	1.2
10 folyamat	6	6.0	6	100	100.0	100
Konstans szorzó	0.6	0.6	0.6	1.0	1.0	1.0
50 folyamat	26	26.0	26	2500	2500.0	2500
Konstans szorzó	0.52	0.52	0.52	1.0	1.0	1.0
100 folyamat	51	51.0	51	10000	10000.0	10000
Konstans szorzó	0.51	0.51	0.51	1.0	1.0	1.0

Az általános1 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Maxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	3.0	5	24	31.0	45
Konstans szorzó	0.6	0.6	1.0	0.96	1.24	1.8
10 folyamat	3	4.0	10	86	135.0	180
Konstans szorzó	0.3	0.4	1.0	0.86	1.35	1.8
50 folyamat	3	4.0	6	2331	2984.0	4275
Konstans szorzó	0.06	0.08	0.12	0.9324	1.1936	1.71
100 folyamat	3	4.0	5	9936	11469.0	15628
Konstans szorzó	0.03	0.04	0.05	0.9936	1.1469	1.5628

Az általános2 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Maxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	4.0	5	22	32.0	48
Konstans szorzó	0.6	0.8	1.0	0.88	1.28	1.92
10 folyamat	3	4.0	8	96	137.0	240
Konstans szorzó	0.3	0.4	0.8	0.96	1.37	2.4
50 folyamat	4	4.0	5	2232	2948.0	4008
Konstans szorzó	0.08	0.08	0.1	0.8928	1.1792	1.6032
100 folyamat	4	4.0	5	10032	11351.0	15200
Konstans szorzó	0.04	0.04	0.05	1.0032	1.1351	1.52

Az általános3 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Maxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	3.0	5	22	31.0	48
Konstans szorzó	0.6	0.6	1.0	0.88	1.24	1.92
10 folyamat	2	4.0	8	90	129.0	224
Konstans szorzó	0.2	0.4	0.8	0.9	1.29	2.24
50 folyamat	3	4.0	5	2424	2948.0	4053
Konstans szorzó	0.06	0.08	0.1	0.9696	1.1792	1.6212
100 folyamat	3	4.0	5	10032	11779.0	15668
Konstans szorzó	0.03	0.04	0.05	1.0032	1.1779	1.5668

A Maxterjed algoritmus értékei nagyban függenek a hálózat alakjától. Gyűrű esetében mindig ugyanaz az érték lett a végeredmény. A legjobb értékeket a teljes gráf alakú hálózat produkálta lépésszám és üzenetszám tekintetében is. Ez az általános3 hálózat. Az általános1 és általános2 üzenetszáma az átlagnál rosszabb. Lépésszám tekintetében az általános1 a csúcstartó, de az általános2 is rosszabb az átlagnál. A konstans szorzók a hálózat méretének növekedésével csökkennek, igaz lépésszám esetében nem mindig.

## A Futtatás menü Optmaxterjed menüpontja

Csak üzenetszám nézet esetén:

A gyűrű1 hálózat esetében:

Optmaxterjed algoritmus  
Összes üzenetek száma: 68  
A lépések száma: 6

A gyűrű3 hálózat esetében:

Optmaxterjed algoritmus  
Összes üzenetek száma: 48  
A lépések száma: 6

Az általános1 hálózat esetén:

Optmaxterjed algoritmus  
Összes üzenetek száma: 98  
A lépések száma: 10

Az általános2 hálózat esetén:

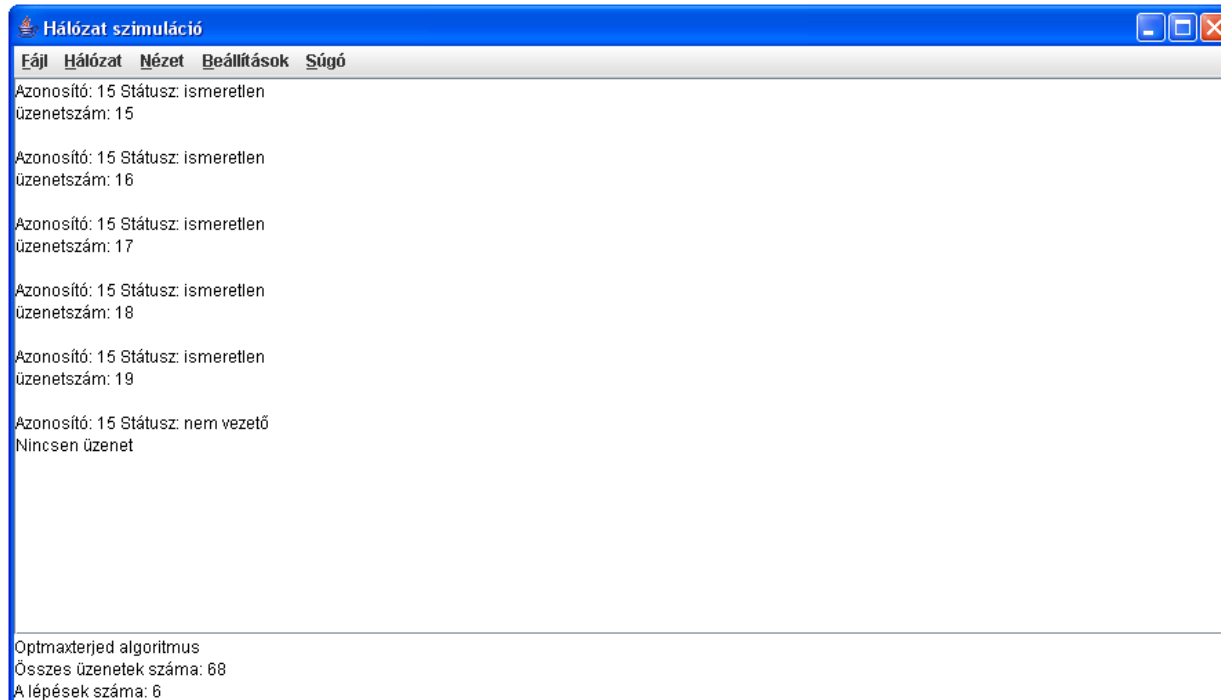
Optmaxterjed algoritmus  
Összes üzenetek száma: 155  
A lépések száma: 7

Az általános3 hálózat esetén:

Optmaxterjed algoritmus  
Összes üzenetek száma: 90  
A lépések száma: 2

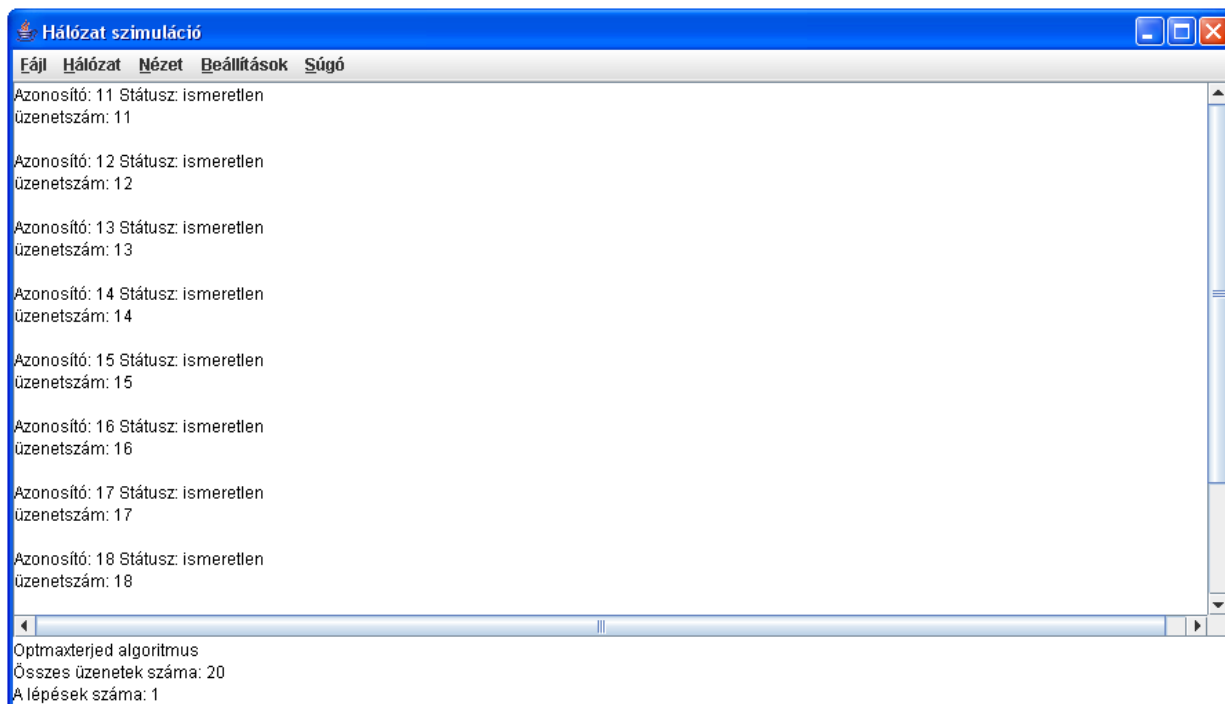


Egy folyamat üzenetei nézet esetén az ötödik folyamat üzenetei:  
A gyűrű1 hálózat esetében:



A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
Az általános1 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános2 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános3 hálózat esetén:  
Az algoritmus hibamentesen lefutott.

Összes folyamat üzenetei nézet esetén:  
A gyűrű1 hálózat esetében:  
Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.  
A gyűrű3 hálózat esetében:  
Az algoritmus hibamentesen lefutott.  
Az általános1 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános2 hálózat esetén:  
Az algoritmus hibamentesen lefutott.  
Az általános3 hálózat esetén:  
Az algoritmus hibamentesen lefutott.

Grafikus nézet esetén:

A gyűrű1 hálózat esetében:

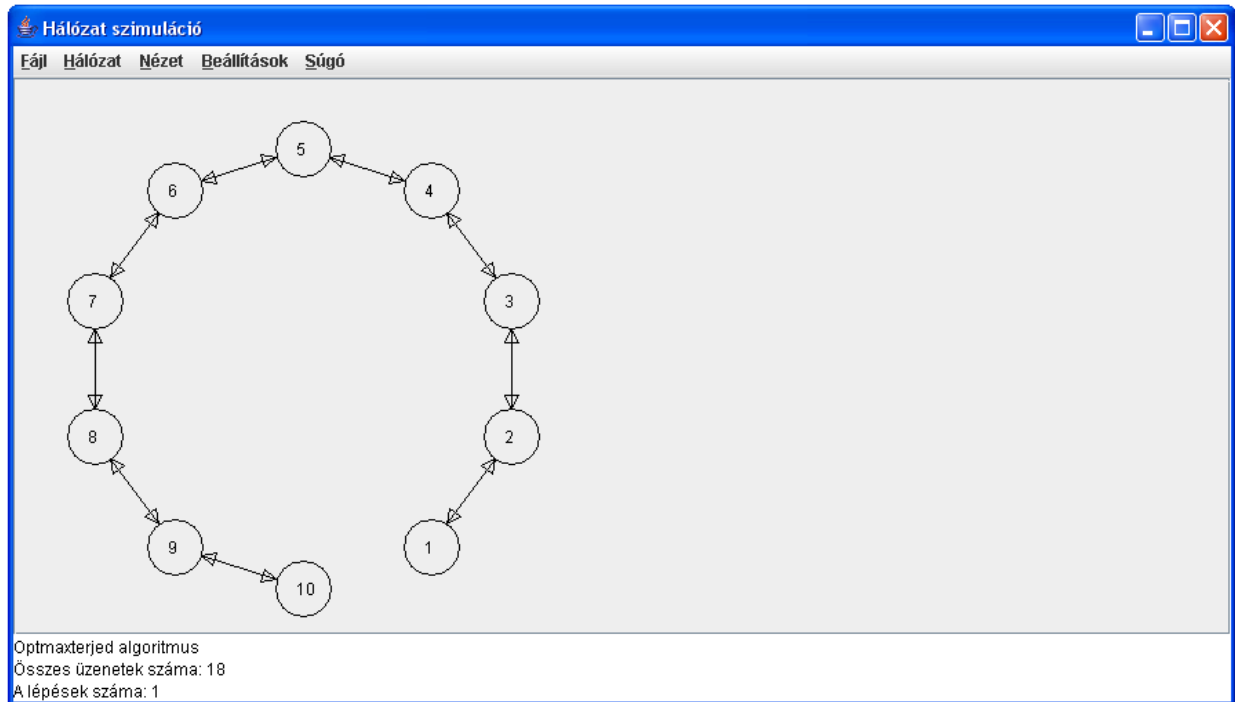
Az algoritmus hibamentesen lefutott.

A gyűrű3 hálózat esetében:

Az algoritmus hibamentesen lefutott.

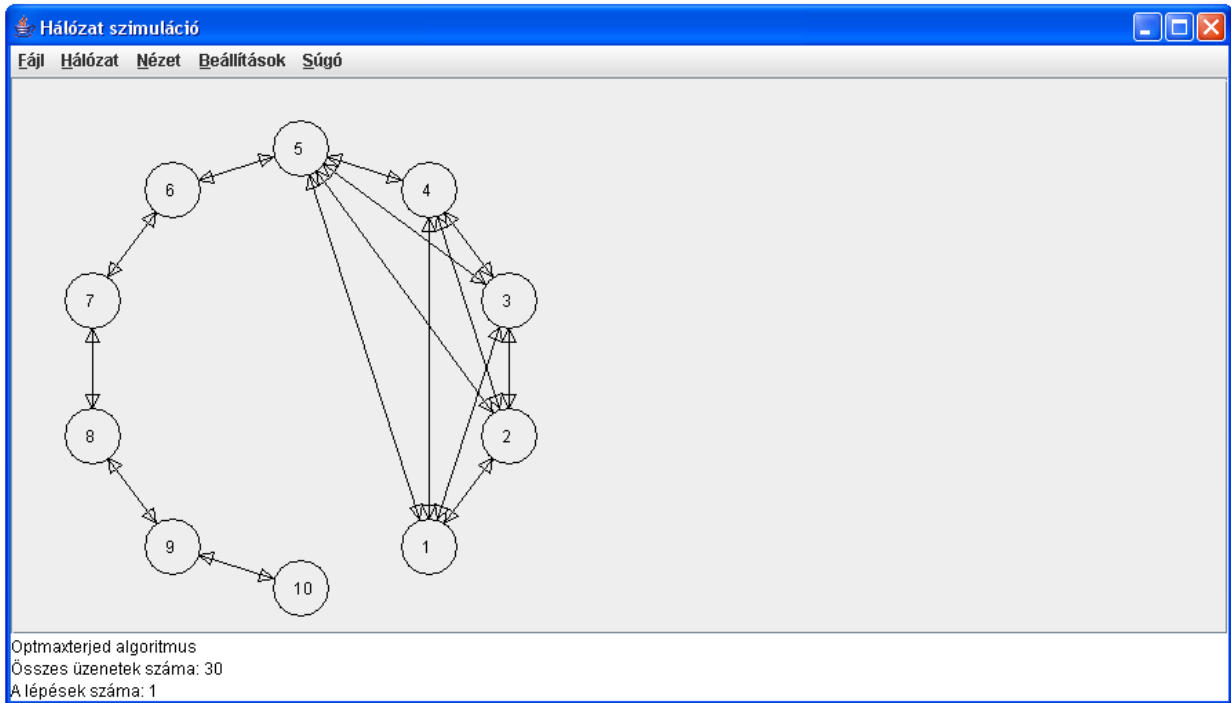
Az általános1 hálózat esetén:

Egy kép a futás közben küldött üzenetekről:

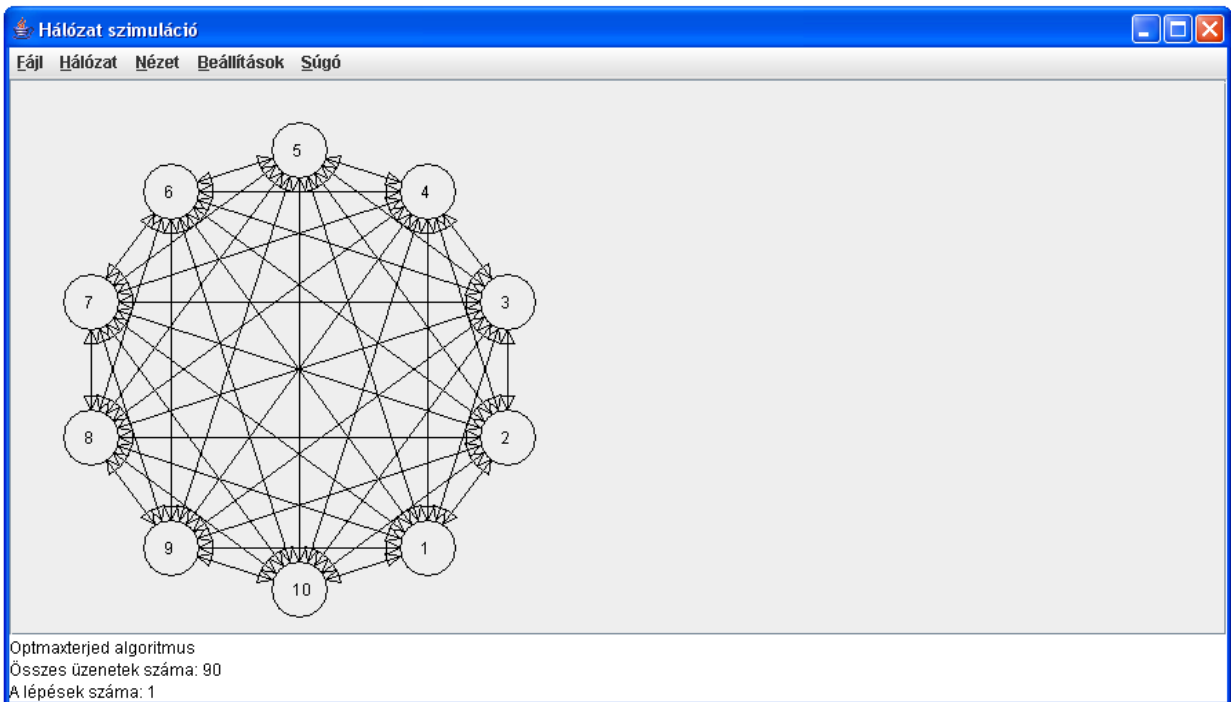


Az algoritmus hibamentesen lefutott.

Az általános2 hálózat esetén:  
 Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.  
 Az általános3 hálózat esetén:  
 Egy kép a futás közben küldött üzenetekről:



Az algoritmus hibamentesen lefutott.

Elemzés nézet. A következő beállításokkal futott: százszor futott minden méretű hálózaton, a hálózatok méretei 5, 10, 50 és 100.

A gyűrűl hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Optmaxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	4	4.0	4	20	20.0	22
Konstans szorzó	0.8	0.8	0.8	0.8	0.8	0.88
10 folyamat	6	6.0	6	44	51.0	68
Konstans szorzó	0.6	0.6	0.6	0.44	0.51	0.68
50 folyamat	26	26.0	26	368	414.0	522
Konstans szorzó	0.52	0.52	0.52	0.1472	0.1656	0.2088
100 folyamat	51	51.0	51	880	982.0	1246
Konstans szorzó	0.51	0.51	0.51	0.088	0.0982	0.1246

Az általános1 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Optmaxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	3.0	5	13	21.0	30
Konstans szorzó	0.6	0.6	1.0	0.52	0.84	1.2
10 folyamat	3	4.0	10	53	89.0	130
Konstans szorzó	0.3	0.4	1.0	0.53	0.89	1.3
50 folyamat	3	4.0	5	1678	2342.0	3457
Konstans szorzó	0.06	0.08	0.1	0.6712	0.9368	1.3828
100 folyamat	3	4.0	5	8159	9729.0	12180
Konstans szorzó	0.03	0.04	0.05	0.8159	0.9729	1.218

Az általános2 hálózat esetében:

Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Optmaxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	3.0	5	12	22.0	32
Konstans szorzó	0.6	0.6	1.0	0.48	0.88	1.28
10 folyamat	3	4.0	8	60	91.0	155
Konstans szorzó	0.3	0.4	0.8	0.6	0.91	1.55
50 folyamat	3	4.0	5	1750	2367.0	3529
Konstans szorzó	0.06	0.08	0.1	0.7	0.9468	1.4116
100 folyamat	3	4.0	5	8244	9705.0	11771
Konstans szorzó	0.03	0.04	0.05	0.8244	0.9705	1.1771

Az általános3 hálózat esetében:

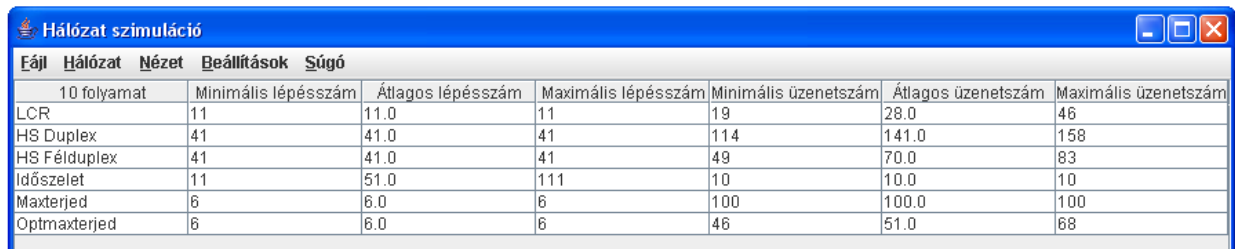
Hálózat szimuláció						
Fájl	Hálózat	Nézet	Beállítások		Súgó	
Optmaxterjed	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
5 folyamat	3	3.0	5	13	21.0	32
Konstans szorzó	0.6	0.6	1.0	0.52	0.84	1.28
10 folyamat	2	4.0	7	60	89.0	134
Konstans szorzó	0.2	0.4	0.7	0.6	0.89	1.34
50 folyamat	3	4.0	5	1770	2339.0	2957
Konstans szorzó	0.06	0.08	0.1	0.708	0.9356	1.1828
100 folyamat	3	4.0	5	8082	9444.0	14215
Konstans szorzó	0.03	0.04	0.05	0.8082	0.9444	1.4215

Az Optmaxterjed algoritmus értékei nagyban függenek a hálózat alakjától. Gyűrű esetében az Optmaxterjed már nem mindig ugyanazt az értéket produkálja, mint a Maxterjed tette. A legjobb értékeket a teljes gráf alakú hálózat produkálta lépésszám és üzenetszám tekintetében is. Ez az általános3 hálózat. Az általános1 és általános2 üzenetszáma az átlagnál rosszabb, de a Maxterjedhez képest jelentősen javult. Lépésszám tekintetében az általános1 a csúcstartó, de az általános2 is rosszabb az átlagnál. A lépésszám konstans szorzói a hálózat méretének növekedésével csökkennek. Az üzenetszám pedig inkább növekszik.

## A Hálózat menü Összes elemzése menüpontja

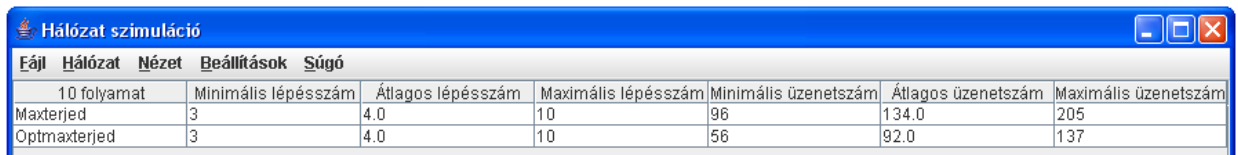
Az összes elemzés során minden algoritmus százszor futott.

A gyűrűl hálózat esetében:



10 folyamat	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
LCR	11	11.0	11	19	28.0	46
HS Duplex	41	41.0	41	114	141.0	158
HS Félduplex	41	41.0	41	49	70.0	83
Időszelet	11	51.0	111	10	10.0	10
Maxterjed	6	6.0	6	100	100.0	100
Optmaxterjed	6	6.0	6	46	51.0	68

Az általános1 hálózat esetében:



10 folyamat	Minimális lépésszám	Átlagos lépésszám	Maximális lépésszám	Minimális üzenetszám	Átlagos üzenetszám	Maximális üzenetszám
Maxterjed	3	4.0	10	96	134.0	205
Optmaxterjed	3	4.0	10	56	92.0	137

Gyűrű esetében a hat algoritmus közül az általános vezető választó algoritmusok lépésszáma a legkisebb. Kis méretű hálózat esetén az üzenetszám tekintetében az LCR algoritmus a legjobb. Általános hálózatban alkalmazható algoritmusok közül az Optmaxterjed egy esetben sem rosszabb, mint a Maxterjed.

## Irodalomjegyzék:

- [1] Nancy Ann Lynch: Osztott algoritmusok  
Kiskapu Kiadó, 2002.  
ISBN: 963-930-103-5
- [2] Nyékyné Gaizler Judit: Java 2 referencia programozóknak: 1.3  
ELTE TTK Hallgatói Alapítvány 2001.  
ISBN: 963-463-488-5
- [3] Nyékyné Gaizler Judit: J2EE útikalauz Java programozóknak  
ELTE TTK Hallgatói Alapítvány 2002.  
ISBN: 963-463-578-4
- [4] <http://java.sun.com/j2se/1.5.0/docs/api/>
- [5] <http://java.sun.com/docs/books/tutorial/uiswing/>