Szabó Dominik Tamás – JM08B3 – 2021. ősz

Baktériumok túlélése

A programom célja, hogy baktériumok együttélését, szaporodását szimulálja N napon át. Az életük eseményeinek esélyét befolyásolhatja, hogy az adott egyed milyen tulajdonsággal bír (akár a többi egyedhez képest) vagy hogy hogyan lép kapcsolatba más egyedekkel. Ezeket az eseményeket (pl. baktérium szaporodik, meghal, harcolni kezd stb.) a program fájlba menti, ahova a felhasználó kéri. A program továbbá statisztikát állít ki a szimuláció összességéről. Kiírja a legtöbb egyeddel rendelkező baktériumfajokat, a legsikeresebb (legszaporább) 0. napi baktériumot, a kiugró tulajdonságokkal rendelkező egyedeket. Ezenkívül kimutatást készít a fajok egyedszámának napi változásáról.

Három forrásfájl tartozik a programhoz. Az egyik a 0. napi baktériumfajok tipikus tulajdonságait olvassa be, a második pedig, hogy a 0. napon hány egyed tartozik a különböző fajokhoz. A harmadik pedig a szimuláció hosszáról, illetve az élelem érkezéséről közöl információt a programnak.

tulajdonsagok.txt

genus (string), species (string) -> a baktériumfaj tudományos neve

age (unsigned int) ->a fajhoz tartozó baktériumok átlagos kora napban

max_lifespan (unsigned int) -> annyi nap, amennyinél többet nem élhet a faj adott egyede

aggression_coef (double) -> segít kiszámolni egy-egy agresszív döntés esélyét

altruism_coef (double) -> segít kiszámolni egy-egy segítőkész döntés esélyét

vision_coef (double) -> segít kiszámolni, talál-e aznapi élelmet az egyed

activeness_coef (double) -> segít kiszámolni, hogy útra keljen-e az egyed aznap

strength (unsigned int) -> nyerési esélyek kiszámolásához harc esetén

q_altruism (unsigned int) -> a baktérium formájára, színére, kinézetére stb. jellemző állandó

fajok.txt

genus (string), species (string) -> a baktériumfaj tudományos neve

darab (unsigned int) -> a baktériumfaj hány egyeddel rendelkezzen a nulladik napon

sim.txt

simlength (unsigned int) – ennyi napig fut majd a szimuláció

foodreserve (unsigned int) – megadja a 0. Napi összélelemmennyiséget

dailyfood(unsigned int) – megadja, mennyi élelem “terem”/kerül naponta a rendszerbe

Működés leírása (főprogram, int main(void){})

A program indulásakor beolvassa a megfelelő fájlokat és azok tartalmából felépíti a nulladik napi baktérium állományt. Ez a következőképpen zajlik: a fajok.txt-ben megtalálható különböző fajok megfelelő számban kerülnek be egy láncolt listába úgy, hogy a lista minden baktériuma tartalmazza a fajra jellemző tipikus (kezdeti) tulajdonságokat, ezenkívül néhány alapértelmezett tulajdonságot. (pl: minden nulladik napi baktérium élő, grand-ancestor értéke 0, stb).

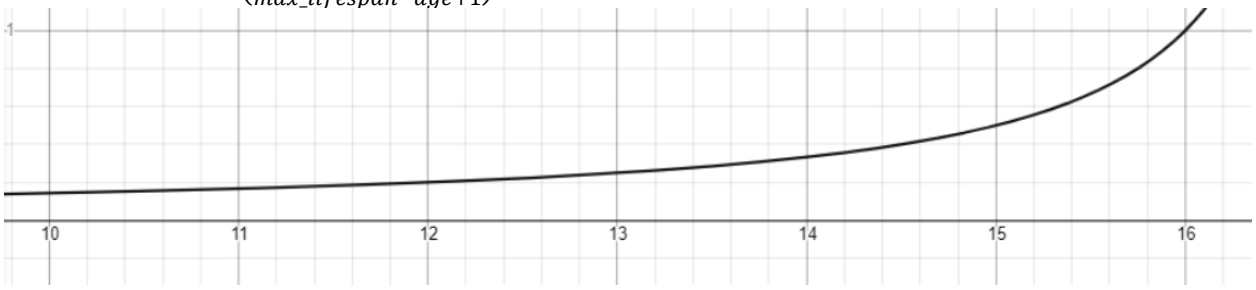
Ezenkívül standard inputra bekér információkat arról, hogy hova kérjük az események logolását (fájlnév) (a log pedig ilyen fájlnévvel a futtatható fájlhoz relatív „logfolder" mappában lesz elhelyezve), illetve megadhatunk számokat, amik a színes diagramot specifikálják.

a. Log: logfájl neve
b. Felbontás (karakterek száma): ez a színes diagram maximális hossza
c. Várt maximum egyedszám: segít kiszámolni azt az arányszámot, amivel beszorozva egy faj élő egyedeinek számát, megkapjuk, hogy hány karakterrel kell őket reprezentálni a színes diagramban.

Egy nap szimulálása a következőképp zajlik (a ciklust követve):

1. Kiírjuk a napi headert, illetve statisztikákat: hanyadik napon vagyunk éppen, hány baktériummal lett ma több, élő baktériumok számának **`(uint currentlyAlive(baclistnode *list))`** változása az előző naphoz képest, illetve hogy mennyi étel maradt. Ezenkívül lerajzolunk egy sornyi színes diagramot.
   Ezzel a **`void colorfulEconioStats(…)`** függvény foglalkozik. Meg kell adni neki a baktériumlistát, a felbontást, a várt maximumot, illetve a fajlistát.
   Továbbá az excellel használható fájlba is írjuk, hány élő baktérium van.
2. Növeljük az élő baktériumok korát, majd szimuláljuk a baktériumok öregedéséből származó halálokat:
   Egyelőre ez a képlet határozza meg az 'age' korú, 'max_lifespan' élettartamú baktérium elpusztulásának valószínűségét:

$$p_{elpusztul}(age) = \left(\frac{1}{max\_lifespan - age + 1}\right)$$



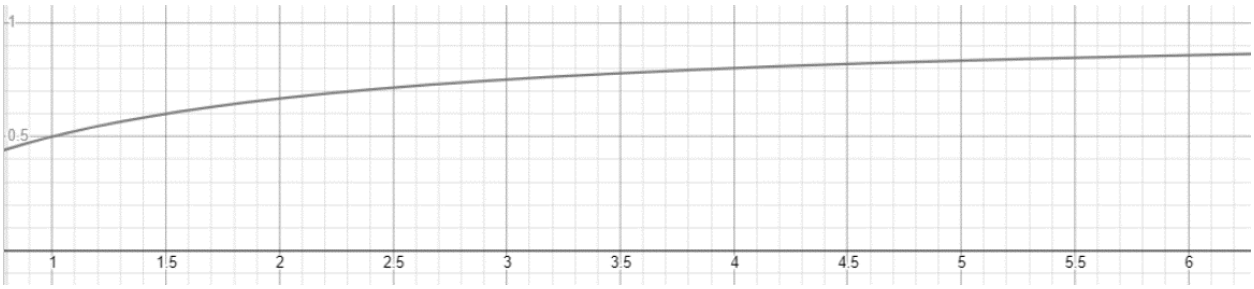
   Az ábrán a p(age) függvény grafikonja látható. 'max_lifespan' = 16, illetve p(16)=1
   Az öregedésért **`void agingDeaths(baclistnode *list)`** függvény felel.

3. Ezután pedig az első körös élelem elosztás történik. A 'sim.txt'-ből származó, illetve a baktériumok eddigi fogyasztásából kiszámolható a kioszthatóélelem. Nem feltétlenül találják meg az összes élelmet.
   Az élelem „kiosztását" a **void foodDistribution(baclistnode *list, uint * foodreserve)** függvény végzi.
   a. Egy foodreserve csökkenés azzal jár, hogy 2-vel nő a baktérium számára felhasználható ételszám.

   b. A baktériumokhoz kerülő ételt különbözőképp használhatják fel a baktériumok.
      i. 1 ételbe kerül öregedni naponta.
      ii. *const int foodToReproduce = 2* ételbe kerül szaporodni
      iii. Ha „elköltötte” ezt az ételmennyiséget, elméletileg akár végtelen számban sokszorozódhat. Ennek akadálya, hogy ehhez végtelenszer egymás után *const double fertilityChance=0.5* valószínűséggel 1-es értéket kapjon. (Lásd: sProb() függvény)

4. Növeljük a maradék ételek számát annyival amennyi terem. *(foodreserve+=dailyfood;)*
5. Minden baktériumhoz tartozik egy ’activeness_coef’ változó. Az ötlet mögötte az alábbi: egy egyed dönthet úgy, hogy egy napon nem „hagyja el otthonát”. Úgy kell elképzelni, mintha téli álmot aludna. Nem tud szaporodni, de azt sem kockáztatja meg, hogy őt megtámadja egy másik éhes baktérium. (Viszont öregszik, természetes okokból elpusztulhat). Tehát: aktív baktérium: minden olyan baktérium, ami nem választja az adott napon ezt a „megbúvó” stratégiát.
   A **`void process_activity(baclistnode *list)`** függvény felelős az *isActive* változó beállításáért.
6. Következő lépésként gyakorlatilag létrehozunk egy olyan gráfot az élő baktériumok között, ami megadja, melyik baktérium melyik másikkal találkozik. (csúcsok: aktív baktériumok, élek: találkozások)
   Ez véletlenszerűen lesz legenerálva, *const double graphEdgeChance = 0.1* valószínűséggel húzzuk be bármelyik élt, ha a baktériumok száma kisebb Ezek az élek pedig fésűs listában vannak tárolva. (Lásd: Lenti fésűs lista ábra)
   A találkozások által lehetővé tett interakciókat a **`void process_meetings(baclistnode *list)`** függvény végzi.
   a. Ha van olyan élő baktérium, aminek nem jutott elég élelem (pl. szaporodásra), az úgy dönthet, hogy megtámad egy olyan baktériumot, akinek van étele, hogy elvegye annak készleteit. Ezt az ’aggression_coef’ befolyásolja. Persze ennek feltétele, hogy találkozzanak az adott napon.
      A harc végkimenetele pedig a baktériumok ’strength’ értékétől függ.
      Tegyük fel, hogy az erősebb baktérium strength értéke ’a’, a gyengébbé ’b’.
      Ekkor az erősebb baktérium nyerési esélyét az alábbi képlet határozza meg:

$$x = \frac{a}{b} \qquad p_{nyerési}(x) = \frac{-1}{x+1} + 1$$



Az ábrán a $p_{nyerési}(x)$ függvény grafikonja látható. Példa: ha a/b=2, $p_{nyerési}(x)$=0.667
A harcok végkimenetele a **`void harc(baclistnode *tamado, baclistnode *vedo)`** függvény által lesz végleg meghatározva, megfelelő állapotok beállítva.

   b. Ha van olyan élő baktérium, aminek saját túlélésére elég étel jut, dönthet úgy (a döntés meghozatalával az ’altruism_coef’ változó van kapcsolatban), hogy egy hozzá hasonló (ez azt jelenti, hogy ’q_altruism' számuk hasonló[1]) baktériumnak ételt ad. Ez persze saját vesztét is okozhatja, de gondolhatjuk, hogy a nem önző stratégia is bizonyulhat sikeresnek.
   Ezt a feladatot a **`void altruism(baclistnode *giver, baclistnode *taker)`** függvény valósítja meg.
7. Akinek elég étele maradt szaporodásra (illetve nem ölte meg pl. egy másik baktérium), akkor az szaporodhat, létrehozva egy magához tulajdonságaiban hasonló, de vele nem tökéletesen megegyező utódot. Szaporodásért felelős függvény:
   `void process_reproduction(baclistnode *list, uint *availableId)`
8. Feldolgozzuk a baktériumok ételfogyasztását, illetve az éhenhaló baktériumokat.
   Ezt a **`void process_foodconsumption(baclistnode *list)`** függvény végzi.
9. A program futásidejének csökkentése érdekében eltávolítjuk a halott, nem nulladik napi(a nulladik napi baktériumokra még szükségünk lesz) baktériumokat, felszabadítjuk a hozzájuk tartozó adatszerkezeteket a memóriából.
   `void removeDeadBacs(baclistnode *list)`

[1] hasonló alatt azt értem itt, hogy a két szám távolsága mennyire nagy vagy kicsi (Lásd: **`calcAltruismChance()`** függvény)

Kiértékelés, statisztikák kiállítása:

1. A szimuláció alatt fut a logolás.
2. A **`void printMaxpopSpecies(baclistnode *list)`** kiírja melyik faj/fajokból van a legtöbb egyed. Először megkeresi mennyi a legtöbb egyedszám, majd kiírja azokat a fajokat, amelyeknek maximum számú egyedszáma volt.
3. A **`void printMostSuccesfulBac(baclistnode *list)`** kiírja, melyik baktériumnak lett a szimuláció végén legtöbb élő leszármazottja. Először megkeresi mennyi a legtöbb egyedszám, majd kiírja azokat egyik egyedet, aminek maximum számú leszármazottja volt.
4. A **`void printMinMaxBacProperties(baclistnode *list)`** kiírja, hogy milyen intervallumra szűkült/tágult a tulajdonságok halmaza.

Adatszerkezetek

typedef unsigned int uint;

A struktúra egy baktérium minden lehetséges információját tartalmazza.

```
typedef struct bacteria
{
    uint id;                csak egy szám, ami minden egyes baktériumra különböző.
    uint ancestor;          annak a baktériumnak az id-je, akinek szaporodása által született
    uint grand_ancestor;    legrégebbi ős id-je

    //allapotjelzok
    int isAlive;            életben van-e a baktérium
    int isActive;           aktív-e a baktériam az adott napon
    uint age;               a baktérium kora
    uint inventory;         mennyi étel van készleten neki

    char genus[25];         nemzetség
    char species[25];       faj

    uint max_lifespan;      az az életkor napban, amikorra biztos meghal a bakterium

    //(0-1) intervallumra lesz az osszes coef normalizalva, ez elvaras.
    double aggression_coef; segít kiszámolni, milyen eséllyel támad meg másik baktériumot
    double altruism_coef;   segít kiszámolni, milyen eséllyel segit egy tarsanak
    double vision_coef;     segít kiszámolni, milyen könnyen talál ételt az adott baktérium
    double activeness_coef;  segít kiszámolni, milyen esellyel indul útnak adott napon
                              baktérium

    uint strength;          segít kiszámolni baktériumok harcának az eredményét

    uint q_altruism;          hasonló számúak nagyobb eséllyel segítenek egymásnak.
} bacteria;
```

A baktériumstruktúrákat tartalmazó elölstrázsás fésűs lista eleme.

```
typedef struct _baclistnode{
bacteria b; //bakterium adatait tartalmazo struktura
struct _edgelistnode *meetings; //talalkozasok listara mutato pointer
struct _baclistnode *next; //kovetkezo elemre mutato pointer
} baclistnode;
```

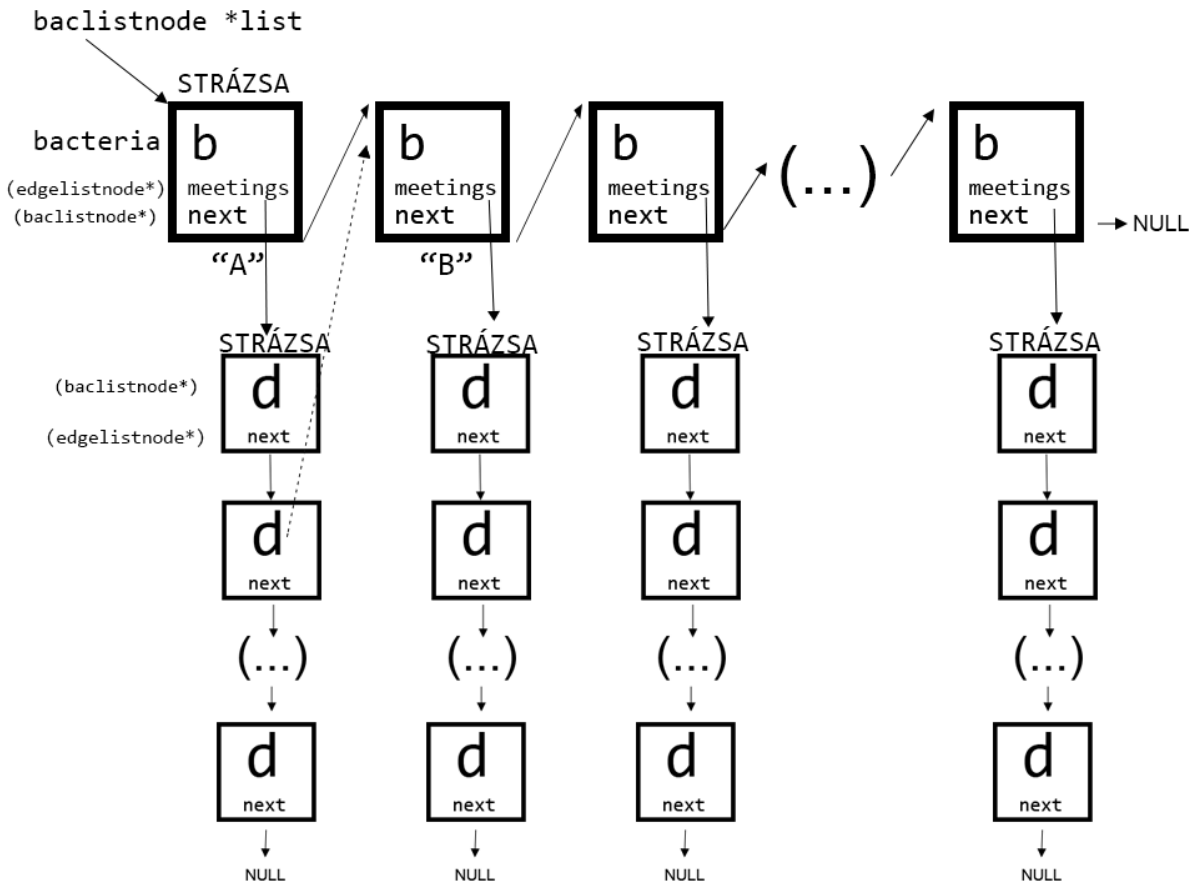
A baktériumok találkozásai ilyen listaelemekként vannak tárolva:

```
typedef struct _edgelistnode{
baclistnode *d; //annak a baktériumnak a pointere, amelyikkel találkozot az a
baktérium, amelyiknek a találkozáslistájában található ez a listaelem
struct _edgelistnode *next; //következő találkozás pointere
} edgelistnode;
```

A fajlista listaelemei:

```
typedef struct _fajlistnode{
    char genus[25]; //meg mindig nem tudom
    char species[25];
    uint db; //hány darab él az adott fajból
    int color; //ez az econios meno szines cucchoz kell:)
    struct _fajlistnode *next; //következő faj pointere
} fajlistnode;
```

Fésűs lista rajz:



A fenti ábra szemléltetni próbálja a fésűs lista adatszerkezetet. A pontozott nyíl jelentése a következő: az „A” baktérium találkozott „B”-vel.

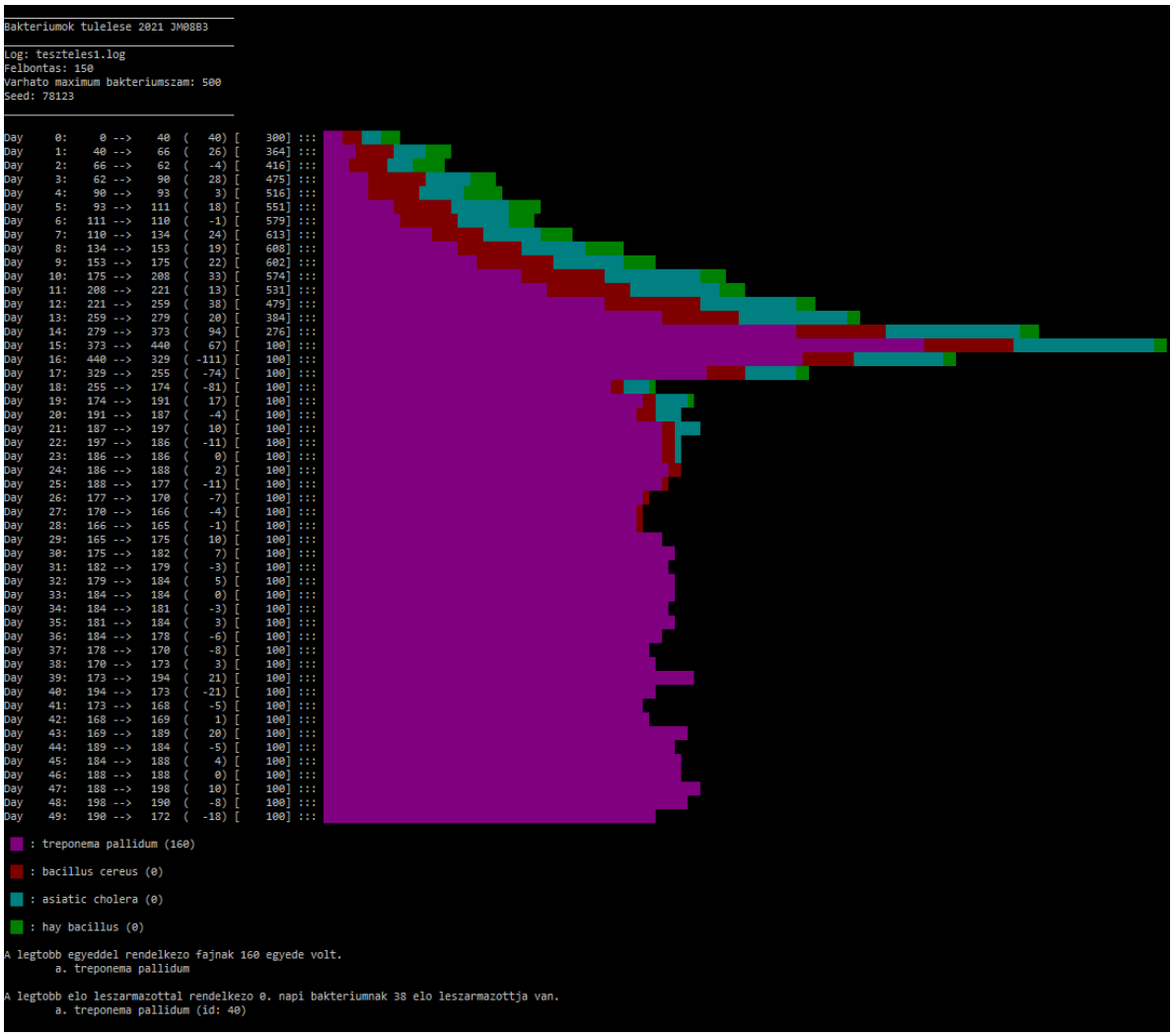
Útmutató a forrásfájlok létrehozásához, értelmezéséhez

1. sim.txt : 3 számot tartalmaz:
    a. **uint simlength;** - a szimuláció hossza napokban
    b. **uint foodreserve;** - kezdeti ételmennyiség
    c. **uint dailyfood**; - naponta ennyi étel teremjen
2. fajok.txt : fajokat tartalmaz
    a. **char samplegenus[]; char samplespecies[25]; uint sampledb;**
    sorrendben soronként, ez egy adott fajra vonatkozik: annak nemzetsége, faja és darabszáma a nulladik napon.
3. Tulajdonsagok.txt
    a. Egy sor egy adott faj tulajdonságaira vonatkozik. Több sor követi egymást, ezek a különböző fajokra vonatkoznak.
    b. Egy sor az alábbi adatokat tartalmazza:
        i. **genus** – „faj nemzetsége”
        ii. **species** – faj
        iii. **age** – tipikus kor
        iv. **max_lifespan** – max életkor
        v. **aggression_coef** – harciasság változó
        vi. **altruism_coef** - altruizmus változó
        vii. **vision_coef** – ételkereső ügyesség
        viii. **activeness_coef** - aktívság
        ix. **strength** - erő
        x. **q_altruism** – megkülönböztetésre használt szám

Tesztelés

1. Az első tesztelési módszer az lesz, hogy megpróbálom megbecsülni, milyen állapothoz fogunk jutni a szimuláció végére. Ehhez a „teszteles” mappában található bemeneti fájlokat fogom használni. Ezt úgy alakítottam ki, hogy:
    a. Körülbelül ugyanannyi kezdeti egyedszámmal induljunk fajonként
    b. Körülbelül 300 adag étellel indulunk, naponta 100 teremhet.
    c. Legyen legalább egy erős / versenyképes faj
        i. Erősebb lesz mint a többiek
        ii. Könnyen talál majd élelmet
        iii. Aktívabb lesz, mint a legtöbb faj
        iv. Tovább élni képes, mint a többiek

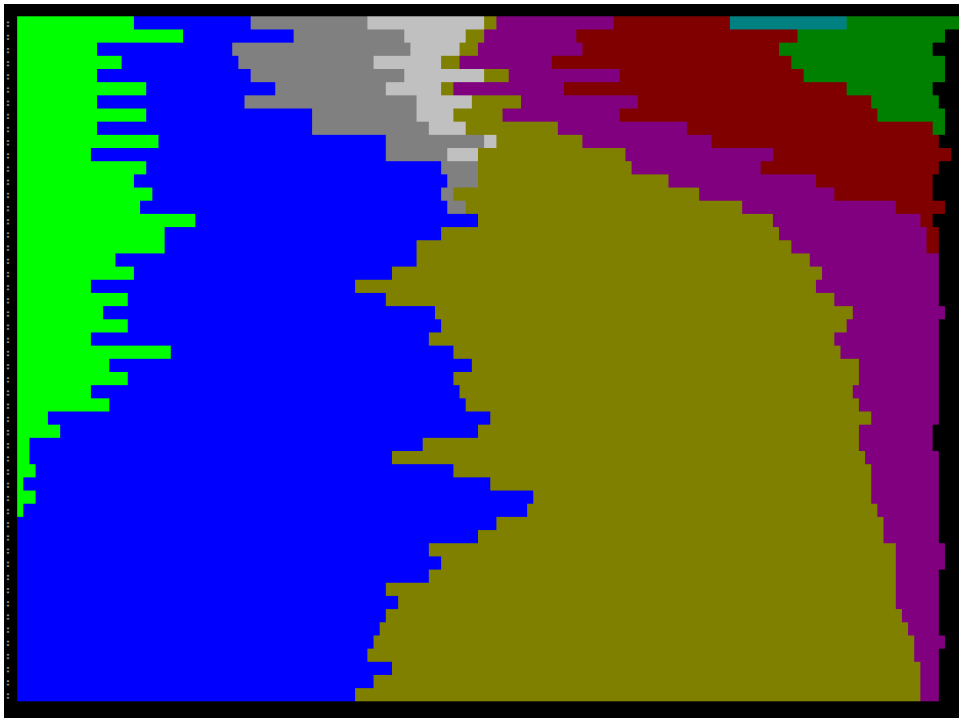
    Így én arra tippelek, hogy az általunk készített erős, „treponema pallidum” baktérium hamar, 20 napon belül dominálni fogja a mezőnyt. Továbbá, a napi 100 élelem miatt az élő egyedek száma majd 100-200 egyed közé normalizálódik. (Egy adag étel 2-vel növeli az inventory-t és 1 ételbe kerül 1 napig élni, 100*2/1=200, ezenkívül pedig más okból is el lehet halálozni.) A szimulációt 50 napig futtatom. Lássuk! (Seed=78123)



2. Ugyanezt a beállítást 1000 napig szimulálva pedig látható, hogy valamiféle evolúció is történik. A „treponema pallidum” kezdeti ereje 250. Azonban 1000 nap múlva már csak jóval erősebb egyedek maradtak:

```
Agresszio:     0.504000 - 0.652000
Segitokeszseg: 0.156000 - 0.404000
Kereses:       0.908000 - 0.992000
Aktivsag:      0.932000 - 0.992000
Ero:                968 -    16545
```

3. Kép egy másik szimuláció eredményéről : (7812312321 mappa, ezzel a seed-del) [2]



[2] Itt a várt maximum 0-ra volt téve, így a fajok egymáshoz képesti egyedszám-arányváltozása látható.

Függvények

Random bool

int sProb(double p) //p valoszínűségű bool

Különböző valószínűségeket számoló függvények

1. double calcWinChance(double x) //kiszámolja a nyerési esélyt harc esetén
2. double calcAgingDeathChance(uint age, uint lifespan) //kiszámolja az öreg baktériumok elpusztásának esélyét
3. double calcMeetChance(uint c) //Két baktérium találkozásának esélye
4. double calcActiveChance(double actcoef) //Kiszámolja, milyen eséllyel lesz aktív egy baktérium az adott napon.
5. double calcAttackChance(double agrcoef) //kiszámolja, milyen eséllyel támad
6. double calcFoodFindChance(double vision) //kiszámolja, milyen eséllyel talál ételt
7. double calcAltruismChance(baclistnode *giver, baclistnode *taker) //kiszámolja, mekkora eséllyel lesz segítőkész egy adott baktérium egy másikkal szemben

Listakezelő függvények

1. baclistnode* ujBacList() //új elölstrázsás üres baktériumlistát hoz létre
2. edgelistnode* ujEdgeList() //új elölstrázsás üres találkozáslistát hoz létre
3. fajlistnode* ujFajList() //új elölstrázsás üres fajlistát hoz létre
4. void freeEdgeList(edgelistnode *listToFree) //Találkozásliták felszabadítására használt függvény
5. void freeBaclist(baclistnode *listToFree) //A baktérium lista felszabadítására használt függvény
6. void freeFajList(fajlistnode *listToFree) //Fajlisták felszabadítására használt függvény
7. void pushBac(baclistnode *list, bacteria b, uint id) // Új baktériumot szúr be egy lista elejére. Az id-t is megkapja, ez a következő elérhető baktérium id.
8. void pushBac(baclistnode *list, bacteria b, uint id) // Új baktériumot szúr be egy lista elejére. Az id-t is megkapja, ez a következő elérhető baktérium id.
9. void pushFaj(fajlistnode *list, char *genus, char *species, int *coloroffset) //Új fajt szúr be egy fajlista elejére. Megadjuk a függvénynek a megfelelő adatokat, + coloroffset pointert
10. void generate_meetings(baclistnode *baclist, uint c) //Ez a függvény legenerálja a találkozáslistát minden baktériumhoz egy listában
11. void removeDeadBacs(baclistnode *list) //a halott, nem nulladik-napi baktériumokat láncolja ki a listából és felszabadítja azokat, ez a program optimalizálását szolgálja

Beolvasó függvények

1. void scanBac(bacteria *toWrite) // beolvas egy baktériumot standard inputról
2. bacteria fscanBac(FILE* fstream, int *X) //beolvas egy baktériumot fileból, paraméterlistán visszaadja, sikeres volt-e, ha nem akkor 1

Kiíró, statisztikát számoló függvények

1. void printBac(bacteria toPrint) //Kiír egy baktériumot releváns adataival.
2. void printBacList(baclistnode *l) //Kiírja a baktériumlistát, debugolásra volt használva.
3. void printMeetings(baclistnode *l) //Kiírja a találkozáslistákat, debugolásra volt használva.
4. void printFajlist(fajlistnode *l) //Kiírja a fajlistát, debugolásra volt használva.
5. void printColorTable(fajlistnode *fajlist) //Kiírja a jelmagyarázatot
6. void countSpecies(baclistnode *baclist, fajlistnode *fajlist) //megszámolja egy baktériumlistában, hogy a fajokból hány darab élő egyed van
7. void colorfulEconioStats(baclistnode *baclist, int res, uint expected_max, fajlistnode *fajlist) //Ez a függvény írja ki a színes diagramot.
8. uint currentlyAlive(baclistnode *list) //megszámolja, hogy egy baktériumlilstában hány db élő egyed van
9. void printMaxpopSpecies(fajlistnode *fajlist) //Kiírja, hogy mennyi volt a legtöbb élő egyeddel rendelkező faj.
10. void printMostSuccesfulBac(baclistnode *list) //kiírjuk a legsikeresebb nulladik napi baktériumot/baktériumokat
11. void printMinMaxBacProperties(baclistnode *list) //A változók intervallumjait számolja ki

A szimuláció központi függvényei

1. void mutate_coef(double *coef) //egy adott 'coef' szerű, 0-1 közötti változó mutációjának szimulálása
2. void harc(baclistnode *tamado, baclistnode *vedo) //Harc szimulálása két baktérium között
3. void altruism(baclistnode *giver, baclistnode *taker) //Végrehajtuk a segítséget - valaki ételt ad a másiknak
4. void agingDeaths(baclistnode *list) //Öregedést feldolgozó függvény
5. void foodDistribution(baclistnode *list, uint * foodreserve, uint dailyfood) //Ételosztásért felelős függvény
6. void process_activity(baclistnode *list) //Ez a függvény dolgozza fel, hogy aktív lesz-e egy baktérium az adott napon
7. void process_meetings(baclistnode *list) //Ez a függvény dolgozza fel a baktériumok közti interakciókat
8. void process_reproduction(baclistnode *list, uint *availableId) //Szaporodást feldolgozó függvény
9. void process_foodconsumption(baclistnode *list) //evést feldolgozó függvény

Econio:

A színes dolgok egyszerű kiíratására használt library. https://github.com/czirkoszoltan/c-econio

Changelog:

- Nyelvtani hiba javítva a specifikációban (fajokból->fajokhoz)
- A specifikációban átfogalmaztam ezt: (régebben: „a program kiírhatja, fájlba mentheti, ha a felhasználó úgy kéri”) → (most: „a program fájlba menti, ahova a felhasználó kéri.”)
- Működésnél eltávolítottam a „pontosan milyen eseményekre vagyunk kíváncsiak” részt, ennek az lett volna a funkciója, hogy pl. csak a szaporodásokat írjuk ki a logba, de ennek nincs sok értelme.
- Átneveztem a `void printInterestingBacs(baclistnode *list)` függvényt, ezentúl `void printMinMaxBacProperties(baclistnode *list)`, máshogy is működik mint eredetileg terveztem (átlagbaktérium számolás, összehasonlítás). Ennek az oka, hogy jobb képet ad a szimulációról, ha csak azokat az intervallumokat határozzuk meg, ami közé a túlélő baktériumok tulajdonságai estek – így akár észre is vehetünk tendenciákat!
- A maximum egyedszám/leszármazottszám kereső függvényeknek a működésében is tettem egy kis változtatást. Nemcsak egy darab, maximális adattal rendelkező esetet írok ki, hanem az összeset, ami maximális eset.
- Sokkal olvashatóbbá tettem a függvények részt, de cserébe nincs kód a pdf-ben. (ez viszont lehet jó is, szerintem). Ellenben igyekeztem a kódot értelmesen kommentelni.
- Új fajlista adatszerkezet