RURGenetika

zápočtový program

Programování II

Rudolf Rosa

cvičící: Doc. RNDr. Pavel Töpfer, CSc.

Obsah

Specifikace....1
Původní specifikace....1
Upravená specifikace....2
Program....3
třída Populace....4
Datové položky....4
Metody....6
třída VlastnostiPopulace....6
maxPocetJedincu....6
vybiravost....6
oblast Udalost, UdalostiRoku....6
UdalostiRoku....6
Udalost....7
oblast Gen, Alela, AlelovyPar, Genotyp....7
Gen....7
AlelovyPar....7
Genotyp....7
oblast Jedinec, Jedinci....8
Jedinec....8
Jedinci....10
třída Program....11
oblast Main()....11
oblast Formy....11
oblast Datové zdroje....11
oblast Metody....11
Formy....11
Hlavni....11
PridatGen....12
Jedinci....12
VlastnostiPopulace....12
SledovaneGeny....12
AboutBox....12
ZadejtePocet....12
Uživatelská část....13
Genetický základ....13
Geny....13
Dědičné znaky....13
Alely....13
Mezialelické vztahy mezi alelami stejného genu....13
Jak funguje dědičnost....14
Mendelovy zákony....14
Hardy-Weinbergův zákon....15
Návod k použití....17
1. Populace....17
2. Geny a znaky....17
3. Model....18
Aplikace....19
Použitá literatura....20
Použité softwarové prvky....20

Specifikace

Program se zabývá modelováním vývoje populace z genetického hlediska. Odvětví genetiky, ke kterému se tento program váže, se nazývá populační genetika.

Vše, co program umí, je podrobně popsáno v dalších kapitolách dokumentace. Stručně lze říci, že program modeluje vývoj složení populace se zadanou charakteristikou. Je možné sledovat zastoupení jednotlivých genů a znaků nebo celých genotypů a fenotypů (genotyp = soubor genů; fenotyp = soubor znaků, projev genotypu), kde jednotlivé znaky jsou podmíněny monogenně (jeden znak je projevem jednoho genu), geny jsou uloženy na autozomech (autozom = nepohlavní chromozom) a mají pouze dva typy alel. Geny přitom mohou ovlivňovat život jedince (např. jeho délku).

Původní specifikace

Program se ve velké míře této specifikace drží; odchylky od ní (většinou vynechané genetické jevy) jsou označeny přeškrtnutím a některé z nich ještě podrobněji popsány níže.

Simulace se bude zabývat modelováním vývoje populace z genetického hlediska. Dostane na vstupu charakteristiku a strukturu populace a požadavek na výsledek. Poté se spustí simulace v podobě vzájemného křížení jedinců v populaci. Výstupem je struktura populace, buď vypisovaná v každé generaci nebo jen jako stav v *n*-té generaci (podle požadavku na výsledek).

Hardyho-Weinbergův zákon říká, že struktura dostatečně velké populace je v genetické rovnováze (s časem se výrazně nemění), pokud je populace panmiktická – jedinci se mezi sebou kříží náhodně. To typická populace ale není, dochází k přirozenému či umělému výběru, a tedy ke změnám struktury populace. A právě pozorování těchto změn na počítačovém modelu by měl můj program umožnit.

Strukturou populace se rozumí zastoupení jedinců daných vlastností (daného genotypu, resp. fenotypu) v populaci. Samozřejmě není dobrý nápad sledovat celý genotyp, neboť se jedná o desetitisíce genů (třeba u člověka), takže by program v takovém případě po dlouhé době zobrazil víceméně nic neříkající výsledek. V modelu se tedy vždy bude sledovat genů jen několik, tak jak je to ostatně v genetice běžné. Zkoumané geny nebo skupinky genů přitom mohou mít různé projevy, různé vztahy mezi alelami téhož genu (dominance, kodominance...) ~~i mezi různými geny (volná kombinovatelnost, genová vazba...)~~, ~~mohou být umístěny na pohlavních chromozomech (X a Y)~~, mohou v některé kombinaci způsobovat neplodnost jedince... – to vše má vliv na předávání genetické informace potomkům a na její projevy.

Charakteristikou populace se rozumí soubor informací o populaci: počet jedinců v populaci, ~~způsob rozmnožování~~, kritéria přirozeného či jiného výběru, natalita, mortalita, ~~otázka incestu~~, pravděpodobnost samovolných změn genetické informace (mutace alel genů, ~~chromozomové aberace~~) apod. Ovlivňuje tedy zásadním způsobem chování modelu. Při zadání jakési ideální panmiktické populace (větší počet jedinců, náhodné křížení, vyloučení mutací,...) by tedy měl pro průběh simulace platit H-W zákon, ostatní populace by se od něj měly nějakým způsobem více či méně odchylovat.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Program bude postaven objektově v jazyce C#, ~~buď konzolově nebo~~ okenně ~~(zatím netuším, jak se okenní aplikace dělají, takže o tomto rozhodnu později; v případě konzolové aplikace by pravděpodobně minimálně vstup byl v podobě textového souboru.)~~ Situace bude pravděpodobně zjednodušena na model, ve kterém se jedna generace rozmnožuje vždy najednou v jednom časovém okamžiku, půjde tedy pravděpodobně o simulaci s pevným časovým krokem.

Součástí samozřejmě budou vhodná a realistická testovací vstupní data - pravděpodobně něco týkající se člověka, např. geneticky podmíněné choroby. Obecně není nutné se omezovat jen na lidské populace, je to ale pravděpodobně nejzajímavější; také je k dispozici více dobře dostupných dat než u jiných organismů.

Upravená specifikace

Zde podrobněji rozeberu, co jsem oproti původní specifikaci vynechal a proč.

Především jsem zjistil, že genetika je ve skutečnosti mnohem složitější, než se obvykle prezentuje. Na úrovni vědeckého výzkumu víceméně neplatí taková ta běžná pravidla, jako že existují dva typy alel (dominantní a recesivní) nebo že na základě dominance nebo kodominance dvou alel se může u jedince projevit jeden znak. Ukazuje se totiž, že alely se vyskytují v mnohých mutacích, které někdy mají vliv na expresi znaku, jindy nikoli. Také se ukazuje, že téměř všechny znaky jsou polygenní – závisí na několika genech, ale často také na několika desítkách genů, z nichž některé mají větší váhu a některé menší. V současné době vědci u většiny znaků neznají přesně jejich závislost na konkrétních alelách konkrétních genů; tyto informace se totiž získávají především statistickými měřeními, a tedy máme informace z této oblasti spíše přibližné.

Na druhou stranu, většinou se s genetikou pracuje právě té **zjednodušené** podobě, kterou používám i já zde. Je to genetika, která se vyučuje na gymnáziích i na lékařských fakultách, a je tedy dle mého názoru dost dobrá na zápočtový program studenta Matfyzu. Pokud bych měl pracovat s genetikou reálnou, pravděpodobně by funkčnost programu musela být značně specializovaná a zároveň omezená, aby byla problematika v rámci zápočtového programu vůbec zvládnutelná.

Již u původní specifikace jsem počítal s tím, že budu pracovat s genetikou v oné zjednodušené podobě; v průběhu práce na programu jsem ale zjistil, že různé skutečnosti jsou různě složité na modelování. Proto jsem se rozhodl vytvořit model ještě o něco jednodušší, a to tím, že jsem některé aspekty prostě **vynechal**. Kvůli tomu pochopitelně bohužel nelze modelovat všechny genetické jevy; volbu mezi tím, co do programu zapojit a co vynechat, jsem se ale snažil provádět tak, aby to, co modelovat lze, bylo možné modelovat co možná nejrealističtěji a v plné míře. Toto řešení jednoznačně považuji za lepší, než kdyby bylo možné modelovat všechno, ale výrazně nedokonale.

Asi nejvýraznějším zjednodušením je, že program pracuje striktně **monogenně**. Genotyp jedince je zde reprezentován polem zcela **nezávislých genů**, zatímco ve skutečnosti je genotyp tvořen několika páry chromozomů, na kterých jsou pak teprve uloženy jednotlivé geny. Z toho v důsledku vyplývá několik vynechávek:

- **Polygenní znaky** neboli znaky závislé na více než jednom genu. V praxi se ukazuje, že většina znaků je polygenních. Jak jsem ale již předeslal, konkrétní kompletní závislosti jednotlivých znaků na alelách genů neznáme a pro jejich stanovení využíváme spíše statistická měření a odhady. Na internetu jsou dostupné průběžně aktualizované výsledky, které publikují týmy zkoumající (převážně lidský) genom. Zkoumal jsem je v průběhu několika dní, a usoudil

jsem, že jejich využití v programu by bylo extrémně složité. Proto se (jak je obvyklé ve zjednodušené genetice) omezuji na monogenní znaky. Případný polygenní znak si uživatel samozřejmě může zadat po jednotlivých genech a výsledky si pak dozpracovat sám.

- **Pohlaví**. Zde je totiž třeba pracovat s chromozomy (X a Y). Na první pohled to může vypadat jako dost podstatný nedostatek, ve skutečnosti ale v genetice pohlaví jedinců nemá tak velký význam, jak by člověk předpokládal. Významně se uplatní pouze u dědičnosti tzv. pohlavně vázaných znaků – tj. znaků, které jsou podmíněny geny ležícími na chromozomech X a Y. To pak způsobuje rozdílnou frekvenci výskytu znaků u mužů a u žen (takovým znakem je třeba hemofilie, která je u mužů přibližně 2× častější než u žen). Toto tedy bohužel modelovat nelze. Také má v reálném světě samozřejmě význam pro výběr partnera, nicméně to se ve větším měřítku ztratí.

- **Genová vazba.** Geny, které leží na stejném chromozomu, jsou spolu tzv. ve vazbě – potomkům rodič obvykle předává celý chromozom, takže u vázaných genů není možná volná kombinovatelnost alel. Tento jev je ale vyvažován jevem zvaným crossing-over, kdy se chromozomy navzájem překříží a vazba se tedy neuplatní. Tento jev má tedy význam jen při sledování několika málo generací, při déle trvajícím pozorování proběhne crossing-over tolikrát, že se význam genové vazby ztrácí.

- **Chromozomové aberace**. Jde o chyby při přepisu genetické informace na úrovni chromozomů (chybějící či přebývající chromozom, změna pořadí genů v chromozomu, znásobení počtu chromozomů), způsobují tzv. syndromy (Downův, Turnerův, apod.). Ty obvykle způsobují neplodnost jedinců, takže pro populační genetiku nemají takřka žádný význam.

Další vynechávkou je **multialelismus**. V reálu jednotlivé geny mají často více než dva typy alel s rozmanitými mezialelickými vztahy (časem totiž mutacemi vznikají nové a nové alely téhož genu, z nichž některé se ukáží jako prospěšné a šíří se v populaci). Můj program se (opět jak je obvyklé ve zjednodušené genetice) omezuje na bialelismus: každý gen může být reprezentován alelami dvou typů, dominantní a recesivní. Ty ovšem mezi sebou mohou mít různý vztah (na výběr jsou čtyři nejběžnější možnosti). Kvůli tomuto omezení nelze bohužel modelovat např. dědičnost krevních skupin systému AB0 – zde existovaly původně alely A (dominantní) a 0 (recesivní), časem ale vznikla mutací alely A alela B (také dominantní).

Dále jsem měl v původní specifikaci uvedeno, že půjde o simulaci s pevným časovým krokem. To je sice pravda – simulace probíhá po jednotlivých letech – nicméně jinak se jedná o normální diskrétní simulaci (podrobněji v popisu programu).

Program

Aplikace je postavena objektově v jazyce C#, a to jako aplikace okenního typu (tzv. Windows Forms Application). Ovládá se plně pomocí jednotlivých formulářových a ovládacích prvků, vstup je realizován výhradně pomocí myši a/nebo klávesnice, výstup pomocí oken.

Obecně jde o určitou odrůdu diskrétní simulace. Jde o klasický způsob se simulačním kalendářem, ten je ale rozčleněn po pevném kroku jednoho roku – typicky se totiž v jednom roce odehraje větší množství událostí, u nichž nezáleží na pořadí. Do určité míry jde tedy o něco mezi spojitou a diskrétní simulací se snahou o využití výhod obou.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Aplikace obsahuje několik formů, nicméně srdce aplikace je umístěno v souboru `Program.cs`. Ten obsahuje jednak statickou třídu `Program`, jednak srdce celé aplikace ve statické třídě `Populace` a v několika dalších třídách.

Dílčí metody, které se starají především o vstupně-výstupní operace, jsou pak umístěny v dalších souborech, reprezentujících jednotlivé formy. Vzhled formů je především naklikaný v designéru, zčásti pak dynamicky generovaný.

Jednotlivé úseky kódu jsou obvykle v rozumné míře komentované (včetně C# XML komentářů). Kód je navíc (vedle normálního třídního členění) členěn pomocí direktiv `#region` a `#endregion` na logické bloky; toto členění mohlo být zčásti provedeno i pomocí dalších tříd a podtříd, ale nebylo. Jednak proto, že by mi to už připadalo trochu přehnané, a jednak asi také proto, že je tohle mé první větší objektové dílo, a nejsem tedy na toto pojetí programování ještě zcela zvyklý. Pokud bych to považoval za významné, celý kód bych znovu prošel a některé položky ještě více zapouzdřil, ale jsem přesvědčen, že v současné podobě je to v pořádku a nezpůsobuje to žádné závažné komplikace.

Většina položek v aplikaci má oprávnění `public`. Vím, že to není úplně tak jak by to mělo být, ale aplikace není příliš rozsáhlá, takže dokážu ohlídat, odkud na co sahám – obvykle to je řádově na několika místech. Některé položky pak musely získat oprávnění `public` proto, že pracují s jinými položkami, které mají oprávnění `public`, a Visual Studio hlásilo chybu v nekonzistenci oprávnění. Nicméně během práce jsem si do určité míry uvědomil některé výhody zapouzdřování, takže u příští objektové aplikace takovéhoto nebo většího rozsahu budu jistě již důslednější a oprávnění `public` budu používat jen tam, kde to bude skutečně nutné.

V následujícím textu rozebírám podrobněji jednotlivé části programu; podrobnost je volena podle zajímavosti a důležitosti, některé málo důležité a zřejmé (nezajímavé) součásti nejsou popisovány vůbec.

třída Populace

Tato statická třída reprezentuje model jako takový. Obsahuje informace o populaci i o stavu modelu populace v daném okamžiku a metody pro práci s modelem populace.

Datové položky

Jde o položky popisující stav populace v daném okamžiku.

jedinci

Instance třídy Jedinci(), reprezentující žijící jedince.

cas

Čas modelu, počítaný od `0` (stvoření populace) v letech.

modelBezi

Boolean, vyjadřuje, zda probíhá modelování – podle toho umožňuje nebo znemožňuje některé akce (např. do běžícího modelu nelze přidávat geny, krok modelu lze provést pouze u běžícího modelu, ...).

sledovaneGeny*

Skupina datových položek reprezentujících sledované geny a jejich projevy. Sledované geny jsou během fáze nastavování ukládány do `Listu`, neboť tato struktura má flexibilní vlastnosti („nafukovací“ velikost, možnost vypouštění prvků). Při spuštění modelu jsou překopírovány do pole, neboť za běhu modelu se nebudou měnit a práce s polem je rychlejší.

Pole `sledovaneGenotypy` a `sledovaneFenotypy` obsahují počty jedinců daného g/fenotypu pro každý možný g/fenotyp; indexují se genotypovým hashem (viz třída `Genotyp`). `sledovaneGenotypy` jsou aktualizovány při narození a úmrtí jedince; obsah pole `sledovaneFenotypy` je dopočítávám na požádání z pole `sledovaneGenotypy`.

vlastnosti

Instance třídy `VlastnostiPopulace`, reprezentuje vlastnosti populace.

podtřída Kalendar

V dvouvrstevné struktuře pracuje s událostmi. Ke své práci pochopitelně využívá třídy `Udalost` a `UdalostiRoku`.

První vrstvou je `List udalosti`. Obsahuje položky typu `UdalostiRoku`, každá položka leží v `Listu` na indexu odpovídajícímu roku, ke kterému se vztahuje. Pro zachování této vlastnosti je nutné, aby při přidávání položky pro daný rok byly nejprve (jako prázdné) přidány položky pro všechny roky předcházející.[1] Také nesmějí být použité položky odstraňovány, nicméně mohou být (a jsou) nastavovány na `null`[2]. `List` jsem zvolil právě proto, že je v něm asi nejjednodušší a časově nenáročné splnit výše popsanou vlastnost.

Druhou vrstvu pak tvoří jednotlivé objekty typu `UdalostiRoku`, které již obsahují konkrétní nesetříděné `Udalosti` pro daný rok, ukládané do spojového seznamu.

Třída obsahuje důležitou metodu `provestUdalostiRoku()`, která pro zadaný rok postupně provede všechny příslušné události; přitom během provádění je možné do probíhajícího roku přidávat nové události, budou také provedeny.

1 U běžné populace ale existuje jen málo let, během kterých nenastanou žádné události, takže tento přístup považuji za adekvátní. Jediným problémem může být spuštění modelu s dlouhými intervaly událostí (např. vysoká délka života jedinců), zde dojde k časové prodlevě a alokaci většího množství paměti. Nicméně toto u modelování běžných populací nenastane, a před modelováním nějakých bláznivých populací je uživatel varován v Návodu k použití.

2 `null` moc místa v paměti nezabere, jde jen o nějakou tu režii, a ta není tak velká, aby u modelování běžných populací způsobovala problémy.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Metody

Většina metod se týká běhu modelu; z ostatních metod stojí za malou zmínku snad jen metoda `nova()`, která v podstatě provede jakýsi měkký restart – zahodí všechna data a nastaví výchozí hodnoty.

modelSpustit

Provede jakousi inicializaci modelu – nastaví některé datové položky `Populace` a vytvoří výchozí jedince dle nastavení (jednak podle vlastností populace, jednak podle vlastností jednotlivých genů).

modelKrok

Vykoná jeden nebo více kroků modelu. V každém kroku inkrementuje čas a zavolá metodu kalendáře `provestUdalostiRoku()`, která vykoná všechny události daného roku.

třída VlastnostiPopulace

Tato třída obsahuje asi 10 položek ovlivňujících život jedinců v populaci. Některé jsou paušálně platné pro celou populaci, jiné (`prumernaDelkaZivota`, `porodnost`) představují výchozí hodnoty, které se u konkrétních jedinců mohou odlišovat v závislosti na expresi genů. Z komentářů a názvů položek je víceméně jasné, k čemu slouží, proto zde vyberu jen několik z nich.

Vlastnosti populace je možné měnit i za běhu modelu, což umožňuje propracovanější modelování.

maxPocetJedincu

Jakási závora, jeho dosažení způsobí dočasné zastavení rozmnožování. Nejde zdaleka o nějaký umělý limit! Tato hodnota zastupuje v modelu to, co není přímo modelováno, přestože to má zásadní vliv na vývoj populace – vlivy prostředí. Další projev vlivů prostředí najdeme u vlastností genů (u exprese genu).

vybiravost

Hodnota vyjadřuje, z kolika potenciálních partnerů si jedinec vybírá při události `svatba`. Podrobnější popis viz `Jedinec.svatba()`.

oblast Udalost, UdalostiRoku

UdalostiRoku

Spravují všechny události jednoho roku. Události se ukládají do spojového seznamu – nezáleží u nich totiž vůbec na pořadí, takže je bez problémů možné přidávat novou událost vždy na konec a brát události vždy ze začátku a tedy s časovou složitostí O(1). Postačuje obyčejný spojový seznam.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Udalost

Událost je vždy vázaná na jednoho jedince. Může být čtyř `typů`; každý typ v důsledku znamená zavolání stejnojmené metody příslušného jedince. Jako určité kontrolní číslo obsahuje událost také int pocetPartneru – při vytvoření události se sem vloží stejnojmenná hodnota z jedince. U událostí, které jsou vázané na partnera, se kterým jedinec byl, když byla událost naplánována (`svatba`, `rozmnozovani`, `rozvod`) je takto kontrolováno, zda událost ještě platí – např. když si jedinec naplánuje, že se se svým partnerem za 30 let rozvede, myslí tím toho aktuálního; pokud k rozvodu dojde dříve, pozbývá tato událost smysl. Speciální hodnota `-1` znamená, že událost na partnerovi nezávisí (`smrt`).

oblast Gen, Alela, AlelovyPar, Genotyp

Tato oblast představuje datové jádro aplikace – na této úrovni probíhá samotné rozmnožování, odsuď vycházejí projevy jednotlivých genů, apod.

Gen

Zastupuje jeden sledovaný gen a zároveň na něm závislé znaky. Jednotlivé položky korespondují s možným nastavením ve formu `PridatGen`. `String`ové položky jsou vcelku jasné, některé `int`ové jsou zajímavější.

atr, porodnost, delkaZivota

Tato pole jsou všechna podobná. Obsahují hodnoty, které pozitivně nebo negativně ovlivní (aditivně) vypočítanou atraktivitu, porodnost a délku života jedince. Uspořádání pole je dáno `enumem AlelovyPar.Typ – { Aa, AA, aa }`.

zastoupeniAlel

Toto pole vyjadřuje poměr (v promile) jednotlivých alel u výchozích jedinců. Na základě těchto hodnot jsou pak náhodně vytvořeny genotypy prvních jedinců.

AlelovyPar

Nejmenší dílek genetické informace, se kterým aplikace pracuje (během rozmnožování ještě pracuje přímo s jednotlivými alelami, ale alela jako třída neexistuje, je předávána pouze informace o typu alely).

Obsahuje mnoho metod pro pohodlné zjišťování různých informací o páru.

Genotyp

Reprezentuje genotyp jednoho konkrétního jedince (především datovou položkou `pary`). Obsahuje několik konverzních metod, na nichž je zajímavé především použití hodnoty `hash`. Práce jednotlivých metod pak již ničím nepřekvapí, jde o vcelku jednoduché převody.

Hash genotypu a fenotypu

`Hash genotypu` jsem zavedl zejména pro možnost indexace pomocí genotypu; přímý způsob by totiž vytvořil mnohadimenzionální pole. `Hash` je obyčejný `int`, takže potřebné pole je pak lineární. Hash je vlastně interpretací genotypu v trojkové soustavě, kde nultý řád čísla představuje gen na indexu 0 v poli `sledovaneGenyPole`; jednotlivým typům alelového páru pak odpovídá prosté přetypování příslušného `enum`u `{ Aa, AA, aa }` na `int`. Vztah prvků množiny genotypů a fenotypů je tedy 1:1, což umožňuje převod tam i zpět. Zajímavým rysem je, že kompletní heterozygot má za všech okolností `hash` roven nule.

Odvozený `hash fenotypu` funguje prakticky stejně (a např. pro neúplnou dominanci alel naprosto stejně), jen jsou v něm některé hodnoty zastíněné jinou hodnotou se stejným projevem (stejně jako jsou některá políčka formu při přidávání genu neaktivní); vztah tedy již není 1:1 a některé indexy (při použití hashe pro indexaci) nebudou využity. Převod fenotyp -> hash je opět bez problémů, opačný převod ale není jednoduše možný; proto jsou při výpisu ignorovány všechny indexy, ve kterých má pole hodnotu 0. Tím pádem nejsou zobrazeny tyto indexy navíc, a jako vedlejší účinek nejsou zobrazeny ani fenotypy, které nejsou v populaci v dané chvíli zastoupeny.

oblast Jedinec, Jedinci

V této oblasti je vše, co se týká jedinců.

Jedinec

Reprezentuje jednoho jedince žijícího v populaci a pochopitelně jde o jednu z nejkomplexnějších tříd v aplikaci. Obsahuje mnoho datových položek několika kategorií i nemálo metod; rozeberu jen ty, které dle mého názoru nemusí být zřejmé nebo jsou něčím zajímavé.

Názvy jsem volil spíše s ohledem na lidskou populaci; nicméně to nebrání používání aplikace pro modelování i jiných populací než lidských, jen je v tom případě třeba si pod těmito názvy představit příslušnou odbornou terminologii.

atraktivita

Atraktivita je vypočítaná z genotypu a představuje zde kritérium umělého výběru. Prohlídkou metod `svatba()` a `chceteMe()` lze snadno zjistit, jak přesně v této aplikaci atraktivita funguje.

V metodě `svatba()` se jedinec snaží si najít co nejatraktivnějšího partnera, který ho chce.

V metodě `chceteMe()` se jedinec (mimo jiné) na základě atraktivity nápadníka rozhoduje o tom, zda ho chce nebo ne.

porodnost

Porodnost je vypočítaná z vlastností populace a z genotypu a přestavuje počet dětí, které se jedinci narodí (pokud jsou k tomu podmínky). Protože se vždy rozmnožují společně dva jedinci, je třeba konkrétní porodnost pro daný pár nějak vypočítat. To jsem se rozhodl provádět tak, že pro

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

každé rozmnožování je vypočítána porodnost jako náhodná hodnota z uzavřeného intervalu porodností obou rodičů. To v důsledku znamená, že i prakticky sterilní jedinec může mít s určitou minimální pravděpodobností potomka; což koneckonců vcelku odpovídá reálnému světu.

pocetPartneru

Počet partnerů, které jedinec již měl (při změně partnera se inkrementuje). Slouží pro rozhodnutí o platnosti událostí vázaných na partnera, viz `Udalost`. Speciální hodnota `-2` znamená, že jedinec je již mrtvý (na takového mohou stále ukazovat události, takže je třeba toto ošetřit - `-2` nikdy nebude odpovídat hodnotě uložené v události, takže se událost s mrtvým jedincem nikdy neprovede).

minulyPartner

Poslední partner, kterého jedinec měl, od něj se "odpíchne" hledání nového partnera – viz metoda `Jedinci.nahodnyJedinec()`.

Jedinec()

Konstruktor zde plní vícero funkcí, proto se o něm zmiňuji. Těmito funkcemi jsou:

- pochopitelně vytvoření jedince s nastavením dat podle parametrů
- vypočítání položek `atraktivita`, `porodnost` a `delkaZivota`
- přidání jedince do struktury `Populace.jedinci` a úprava pole `Populace.sledovaneGenotypy`
- naplánování vlastní svatby a smrti

chceteMe()

Jednoduchá metoda, která simuluje rozhodování se jedince, zda chce či nechce mít za partnera zadaného nápadníka; dala by se pravděpodobně ještě vylepšit, ale nejde mi ani tak o model zcela dokonalý, jako spíš o model poskytující rozumné a realistické výsledky.

svatba()

Jedinec si vybírá co nejatraktivnějšího jedince, který ho chce (uplatňuje se zde vlastnost `vybiravost` – je součástí souboru vlastností populace). Přitom mu nijak nevadí, pokud je jeho idol zadaný – přece se kvůli němu rozvede! Pokud ho nikdo nechce, zkusí to znovu příští rok (tento interval – 1 rok – by případně také bylo možné zahrnout do vlastností populace).

rozvod()

Korektně se rozvede se svým partnerem a případně si rovnou naplánuje svatbu. (Pokud se rozvádí, protože to měl v plánu, plánuje rovnou svatbu. Pokud se ale rozvádí kvůli jinému jedinci, tak ho svatba už čeká a nemusí ji už plánovat.)

rozmnozit()

V určitém smyslu nejdůležitější metoda (společně s pomocnými metodami `vyberAlelu()` a `alely2par()`). Pokud má partnera, stanoví si společně počet dětí, které se pokusí počít (metodika viz `porodnost`). Poté každému dítěti po jednotlivých alelách jednotlivých genů sestaví genotyp a porodí jej. (A nakonec si nezapomene naplánovat další rozmnožování.)

Pomocné metody zde slouží pro „převod alelový pár <-> alela“. `vyberAlelu()` pro daný `AlelovyPar` každého rodiče vybere jednu z alel, kterou rodič předá potomkovi (v podstatě simulace redukčního dělení). `alely2par()` naopak takto získané dvě alely spojí v `AlelovyPar`.

Jedinci

Reprezentuje jedince žijící v populaci. Původně jsem experimentoval s různými `Listy` a podobnými strukturami, až jsem si uvědomil, že mi stačí obyčejný spojový seznam (tedy ne úplně obyčejný, ale obousměrný a cyklický). Na jednotlivé jednice se totiž přistupuje pouze dvěma způsoby.

Prvním způsobem je přístup z událostí, které si nesou přímo ukazatel na konkrétního jedince, případně když jedinec manipuluje sám se sebou. Toto se ve spojovém seznamu tohoto typu udělá naprosto triviálně, a tedy tyto metody nebudu ani rozebírat.

Druhý typ přístupu se vyskytuje v metodě `svatba()` při výběru partnera – bylo tedy třeba implementovat nějakou metodu, která náhodně vrátí nějakého jedince. To ve spojovém seznamu nejde přímo, a tedy jsem musel najít nějaký náhradní způsob – funkční, a zároveň ne zbytečně náročný.

nahodnyJedinec()

Klíčem k úspěchu metody je vstupní parametr, kterým je jedinec. Metoda pak od tohoto jedince provede několik[3] kroků po spojovém seznamu směrem vzad a vrací nalezeného jedince. Ten, kdo metodu volá (metoda `Jedinec.rozmnozit()`), má pak povinnost zajistit, že předá metodě vhodného výchozího jedince.

Zde se to děje tak, že je metodě předán vždy buď poslední partner[4] (který buď byl nalezen tímto postupem a tedy od něj můžeme postupovat dále dozadu, nebo jde o víceméně náhodného jedince), nebo (při několikerém volání po sobě) je vždy předán předchozí výsledek, takže hledání pokračuje z místa, kde skončilo.

Není to samozřejmě zcela náhodné (název metody je trochu zavádějící), ale vcelku to odpovídá realitě: jedinci se většinou spárují s jedinci, kteří žijí někde blízko; postupem času se pak případně dostávají dál a dál od místa, kde začali – někdy se ovšem do tohoto místa zase vrátí. S rozumně velkou pravděpodobností (řekněme nadprůměrnou) také občas vznikne znovu pár, který už zde jednou byl – to je ale také v pořádku, občas se stává, že se k sobě bývalí partneři vrátí.

3 Několik zde znamená náhodně 1 až 4.

4 Pokud tento neexistuje, předá jedinec sám sebe.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

třída Program

Tato statická třída obsahuje především několik metod a datových položek, které spíše než ke konkrétní třídě či objektu patří celému programu. Podrobněji je rozeberu podle oblastí (`#region`).

oblast Main()

Standardní funkce Main, jsou zde navázány handlery pro ošetření výjimek.

oblast Formy

Obsahuje jakési ukazatele na formy – ty totiž musí být dostupné v celém programu. Dále obsahuje metody pro aktivaci a obnovování formů a metodu `ZadejtePocet`: ta vyzve uživatele k zadání počtu něčeho (kroků programu) a získanou hodnotu předá příslušné funkci (`Populace.modelKrok()`).

oblast Datové zdroje

Obsahuje několik datových položek používaných v programu. U většiny jde o jakési „pole pseudo-`enum`“ – kdyby `enum` mohl obsahovat položky s mezerami v názvu, použil bych `enum`, ale takhle mi nezbývá než uložit hodnoty do pole a v programu používat reprezentaci typem `int` představujícím index položky. Dále obsahuje související data a generátor náhodných čísel.

oblast Metody

Obsahuje metodu `hw()` a metody pro zpracování výjimek (ty nejsou příliš zajímavé).

hw()

Z podílu jedněch homozygotů spočte podíl heterozygotů a druhých homozygotů (v promile) tak, aby populace byla v genetické rovnováze; jde o vzorce $p + q = 1$ a $p^2 + 2pq + q^2 = 1$ (viz Hardy-Weinbergův zákon). Používá se ve formu `PridatGen`.

Formy

Aplikace obsahuje množství formů, plnících různé funkce, které lze snad intuitivně uhodnout podle názvy formu. Není tam moc zajímavého kódu, stručně tedy k jednotlivým formům.

Hlavni

Hlavní form aplikace slouží k ovládání celé aplikace. Obsahuje menu, pomocí kterého jsou dostupné veškeré funkce. Tyto mají také přiřazeny klávesové zkratky. Každá funkce má svou ikonku, která se objevuje v souvislosti s ní v celé aplikaci, např. jako ikonka reprezentující příslušný form.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Většina volání metod z menu vede buď na některou z metod třídy `Populace`, nebo na vyvolání některého formu.

PridatGen

Form pro přidávání genů do modelu. Cílem je vytvoření genu voláním `new Gen()`. Obsahuje několik rozsáhlejších metod, ale jde vesměs jen o různé hrátky s formulářovými prvky.

Zajímavější je leda zamykání a odemykání položek nastavováním `Tagu` na `0` nebo `1`. Pokud totiž s formulářem nepracuje uživatel, ale některá z metod formu, je někdy potřeba zabránit vykonávání operací, které jsou typicky navázané na události typu `ValueChanged` apod. To se děje právě pomocí tohoto zámku.

Form využívá metodu `Program.hw()` pro počítání rozložení alel v populaci podle Hardy-Weinbergova zákona.

Jedinci

Hlavní výstup programu, vypisuje složení populace. Typ výpisu je jedna z položek pole `Program.moznostiZobrazeniJedincu`. Využívá především pole `Populace.sledovaneGenotypy` a `Populace.sledovaneFenotypy`.

VlastnostiPopulace

Umožňuje nastavit vlastnosti populace – viz třída `VlastnostiPopulace` a datová položka `Populace.vlastnosti`. Obsahuje metody obsluhující formulář.

Většina položek formuláře je typu `NumericUpDown`, kde vlastnosti `Minimum` a `Maximum` reprezentují rozsah povolených hodnot. Minimum je voleno tak, aby dávalo smysl – 0, 1, 2 nebo 3. Maximum je obvykle nastaveno jako 999 999 999. Hodnoty jsou voleny tak, aby se vešly do typu `int32` a bohatě převyšují požadavky na modelování běžných reálných populací.

SledovaneGeny

Vypisuje sledované geny (s využitím položky `Populace.sledovanyGeny`), umožňuje je odstranit.

AboutBox

Standardní form „O aplikaci“.

ZadejtePocet

Form pro zadání počtu – slouží pro obsluhu metody třída Program.ZadejtePocet. Je postaven jako univerzální, ale slouží pouze pro obsloužení volby „Více kroků...“.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Uživatelská část

Před používáním programu je vhodné mít určitý genetický základ. Minimálně je dobré tušit, co je to gen, alela a znak, jak se alely projevují a jak se předávají potomkům.

Genetický základ

Aplikace pracuje s genetikou ve zjednodušené podobě a ta zde také bude popisována. Zájemce s hlubším zájmem o genetiku může najít velké množství informací (v angličtině) na internetu.

Pro seznámení s genetikou dobře poslouží [1]. Zde volně cituji některé základní informace.

Geny

Gen je specificky uložená jednotka dědičné informace. Z genetického hlediska jde o informaci, která ovlivňuje podobu jednoho znaku jeho nositele.

Dědičné znaky

Dědičné znaky jsou vlastnosti organismu vzniklé expresí genů. Jejich soubor v rámci jednoho organismu se nazývá fenotyp. Některé mohou být pozorovatelné, některé jsou zjistitelné pouze za pomoci speciálních vyšetření. Znaky lze dělit podle jejich „měřitelnosti“ na kvalitativní a kvantitativní – program umožňuje sledovat pouze znaky kvalitativní.

Kvalitativní znaky jsou znaky neměřitelné, tvoří několik odlišných variant. Příkladem mohou být některé **geneticky podmíněné choroby** – jedinec může být buď **zdravý** nebo **nemocný** (u některých chorob může být zdravý jedinec „přenašečem“ – choroba se u něj neprojevuje, ale může se vyskytnout u jeho potomků).

Alely

V diploidní buňce existují pro každý gen **dvě alely**, tedy konkrétní formy genu. Pokud jsou tyto stejné, označuje se takový jedinec jako **homozygot**. Pokud jsou různé, označuje se jako **heterozygot**.

Mezialelické vztahy mezi alelami stejného genu

Diploidní jedinci mají od každého genu 2 alely. U jedince se pak může projevit jedna nebo obě některým z níže uvedených způsobů.

Je zvykem dominantní alelu označovat velkým písmenem (*A*) a recesivní alelu písmenem malým (*a*). Toto označení se často zavádí i u alel s jiným typem vztahu než je dominance a recesivita.

Úplná dominance (a recesivita)

Dominantní alela úplně potlačí projev recesivní alely. Dominantní alela je tedy taková, která se projeví i v heterozygotní kombinaci. Jde o nejčastější typ vztahu alel.

Např. polydaktilie (výskyt nadpočetných prstů) je dominantně podmíněná choroba, jejíž alely si označíme jako *P* (dominantní) a *p* (recesivní). Heterozygot (*Pp*) i dominantní homozygot (*PP*) budou tedy chorobou postiženi (a to ve stejné míře), recesivní homozygot (*pp*) bude zdráv.

Neúplná dominance (a recesivita)

Dominantní alela nepotlačuje recesivní alelu úplně, recesivní alela se také částečně projeví.

Např. barva květů u rostliny kejklířky skvrnité, kde alela *A* způsobuje červenou barvu a alela *a* žlutou barvu. Fenotypy jsou pak následovné: homozygot *AA* - červená barva; homozygot *aa* - žlutá barva; heterozygot *Aa* - oranžová barva.

Kodominance

Obě přítomné alely se u heterozygota projeví v celé míře a navzájem se neovlivňují. Např. u dědičnosti krevních skupin různých systémů.

Superdominance

Heterozygot (Aa) vykazuje silnější formu znaku než oba typy homozygotů (aa, AA). Není příliš častá.

Jak funguje dědičnost

Díky dědičnosti dochází k přenosu určitých znaků z rodičovské generace na generaci potomků. V případě vyšších (eukaryotních) organismů (které jsou diploidní – každý gen je reprezentován dvěma alelami) je dědičnost většinou založena na tvorbě gamet (pohlavních buněk). Ty vznikají redukčním dělením, která dává za vznik haploidním gametám (od každého genu je přítomna jen jedna alela). V praxi to znamená to, že rodič může potomkovi předat pouze některé své alely – od každého genu pouze jednu; přenos alel na potomky podléhá základním pravidlům kombinatoriky.

Jako první vyřešil tuto problematiku Johann Gregor Mendel díky svým známým pokusům s křížením hrachu. Jeho poznatky shrnují 3 Mendelovy zákony.

Mendelovy zákony

Zákony byly sestaveny na základě Mendelových pozorování a pokusů, proto jsou formulovány poněkud obšírněji, než by bylo třeba. Uvádím je zde pro úplnost (a také jako výraz pocty otci genetiky), nicméně jejich poslání lze shrnout asi takto:

Při rozmnožování každý rodič předá svému potomkovi od každého genu jednu ze svých alel, která je vybrána zcela náhodně. U každého potomka se tedy alelový pár každého genu skládá z jedné alely otcovské a jedné alely mateřské.

1. Mendelův zákon - Zákon o uniformitě generace F1 (1. filiální = první generace potomků)

Při vzájemném křížení 2 homozygotů (generace P – parietální, rodičovská) vznikají potomci genotypově i fenotypově jednotní. Pokud jde o 2 různé homozygoty, jsou potomci vždy uniformními heterozygoty.

2. Mendelův zákon - Zákon o náhodné segregaci genů do gamet

Při křížení 2 heterozygotů může být potomkovi předána každá ze dvou alel (dominantní i recesivní) se stejnou pravděpodobností. Dochází tedy ke genotypovému a tím pádem i fenotypovému štěpení = segregaci. Pravděpodobnost pro potomka je tedy 25% (homozygotně dominantní jedinec) : 50% (heterozygot) : 25% (homozygotně recesivní jedinec). Tudíž genotypový štěpný poměr F2 je 1:2:1.

3. Mendelův zákon - Zákon o nezávislé kombinovatelnosti alel

Při zkoumání 2 alel současně dochází k téže pravidelné segregaci. Máme li 2 dihybridy AaBb může každý tvořit 4 různé gamety (AB, Ab, aB, ab). Při vzájemném křížení tedy z těchto 2 gamet vzniká 16 různých kombinací. Některé kombinace se ovšem opakují, takže nakonec vzniká pouze 9 různých genotypů (poměr 1:2:1:2:4:2:1:2:1).

Hardy-Weinbergův zákon

Tento zákon popisuje rozložení alel v populaci, která je v genetické rovnováze. Platí v populacích dostatečně velkých (dokonale platí u populací nekonečně velkých, ale i u populací nad 1000 jedinců se dá použít) a zároveň panmiktických (tedy takových, kde se jedinci kříží náhodně, bez ohledu na genotyp). Protože se týká jednotlivého genu, stačí, když je populace panmiktická v daném genu.

Zákon je vlastně vcelku jednoduché matematické pozorování[5]. Říká toto:

Nechť p je frekvence dominantní alely a q je frekvence recesivní alely (jako poměrné zastoupení, tedy např. $p = 0{,}2$ znamená, že jedna pětina všech alel tohoto genu v populaci je recesivní). **Pak platí: $p + q = 1$** (což je vskutku překvapivé).

Zajímavější jsou důsledky tohoto zákona (které z něj ovšem přímo plynou):

- frekvence dominantních homozygotů je p^2
- frekvence recesivních homozygotů je q^2
- frekvence heterozygotů[6] je $2pq$

5 G. H. Hardy byl matematik, kterého jeho přítel biolog R. Punnet požádal o spolupráci, neboť měl určitá pozorování, ale sám je nedokázal správně matematicky vyjádřit; W. Weinberg byl lékař, který nezávisle na Hardym a pravděpodobně sám dospěl ve stejné době ke stejným závěrům

6 Toto je triviální až na druhý pohled – úvaha vede tudy:
$p + q = 1$
$(p + q)^2 = 1^2$
$p^2 + 2pq + q^2 = 1$ a již je to úplně zřejmé

Pokud předpokládáme konstantní frekvenci jednotlivých alel v populaci, můžeme tedy předpokládat i konstantní frekvenci heterozygotů, dominantních homozygotů a recesivních homozygotů.

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Návod k použití

Použití programu by mělo být víceméně intuitivní. Základní postup při práci s programem je asi takový:

1. Populace

V menu *Populace* nastavte *Vlastnosti populace*. Jednotlivé položky jsou doplněny vysvětlujícími popisy, výchozí nastavení by mělo víceméně odpovídat lidské populaci.

Výchozí a *maximální počet jedinců* jsou velmi důležité položky a jejich nastavení může mít na populaci zásadní vliv. Je rozumné, pokud jsou např. v poměru 1:2 – populace pak víceméně setrvává ve stálém stavu, což je u populací běžné. Pokud nastavíte *maximální počet jedinců* na velké číslo, budete modelovat populaci „v rozkvětu“, nebrzděnou vnějšími vlivy, progresivně se vyvíjející. *Maximální počet jedinců* má svůj zřetelný význam – simuluje totiž vliv prostředí, které obvykle neumožňuje populaci neomezené rozmnožování.

Časovou jednotkou modelu je rok, což se dobře hodí pro lidi i většinu jiných organismů, méně už např. pro prvoky. Pokud Vám tato jednotka nevyhovuje, prostě si pod ní představte např. den, hodinu, minutu...

Vlastnosti populace je možné měnit i za běhu modelu, je ale nutné počítat s tím, že změna ovlivní vývoj populace pouze dopředně a s určitou setrvačností. Takže např. *výchozí počet jedinců* nemá smysl měnit, na modelu se to neprojeví; snížení *maximálního počtu jedinců* pod aktuální počet jedinců nezpůsobí okamžité vymírání populace, ale pouze zastaví rozmnožování; nově nastavená *délka života* či *dospělost* se budou týkat až nově narozených jedinců, atd.

V okně *Jedinci* můžete sledovat jedince, kteří jsou v aktuální populaci naživu; na výběr máte několik možností, jak je zobrazit.

Volba *Nová populace* Vás prakticky dostane do stavu po spuštění programu.

2. Geny a znaky

Aby bylo co modelovat, je potřeba zadat do programu geny, které chcete sledovat. K tomu pochopitelně slouží volba *Přidat gen*. Geny, které přidáte, poté můžete vidět a upravovat přes volbu *Sledované geny*.

Okno *Přidat gen* má mnoho voleb, které jsou automaticky nastaveny na nejběžnější výchozí hodnoty. Pro přidání genu stačí vyplnit *Název* a alespoň jeden *Projev genu (znak)*, ostatní položky je možné ponechat tak, jak jsou.

U některých položek se při najetí myší zobrazí stručná nápověda. Na některé z nich se podrobněji podíváme zde:

- *Alely* – zadejte písmeno či písmena označující dominantní a recesivní alelu. Výchozí označení je podle názvu genu – např. při vyplnění názvu `Krátkozrakost` se alely automaticky pojmenují `K` a `k` (jak je v genetice ostatně běžné).

- *Projev genu (znak)*. Zde vyplňte, jak se u jedince projeví daný alelový pár. Např. pro gen krátkozrakost můžete vyplnit AA: `vidí dobře` a aa: `je krátkozraký`. Pokud je některá položka neaktivní, znamená to, že ze *Vztahu alel* tato hodnota automaticky vyplývá (je shodná s některou jinou). Uvidíte, že se položka automaticky vyplní, pokud vyplníte tu, na které je závislá. Některý *projev* můžete nechat i nevyplněný (tzn. daný alelový pár se nijak neprojevuje), ale je vhodnější i toto nějak slovně popsat. Pokud nevyplníte žádný *projev*, program Vás nenechá gen přidat (gen, který se nijak neprojevuje, nemá smysl sledovat).

- *Atraktivita*. Na základě genetické informace má každý jedinec vypočítanou atraktivitu, která hraje roli při výběru partnera (jedinec se snaží mít co nejatraktivnějšího partnera). Jednotlivé znaky mohou atraktivitu zvyšovat či snižovat – do pole vyplňte, o kolik se atraktivita nositele znaku zvýší (či sníží, pokud vyplníte zápornou hodnotu). `0` znamená, že znak nemá na atraktivitu vliv.

- *Porodnost* a *Délka života* fungují podobně jako atraktivita (mohou se zvýšit či snížit) – změna probíhá oproti hodnotě nastavené ve *Vlastnostech populace*. Dostatečně malými hodnotami (neboli velkými zápornými čísly) lze pak dosáhnout sterility jedince (znak znemožňující rozmnožování) nebo „spontánního potratu“ jedince s daným znakem (znak neslučitelný se životem).

- Ve *Výchozím rozložení genů* si zvolte, jak mají vypadat jedinci, kteří budou vytvořeni po spuštění modelu (běžné znaky mívají víceméně rovnoměrné rozložení, zatímco např. u geneticky podmíněných chorob bývá obvyklé jejich malé zastoupení v populaci). Zaškrtávací políčko Vám umožní nechat hodnoty automaticky dopočítávat tak, aby byla populace v genetické rovnováze (viz Hardy-Weinbergův zákon).

Sledované geny pochopitelně nelze přidávat ani odebírat za běhu modelu.

3. Model

Když je všechno nastavenou, je na čase *Spustit model*. Tím dojde k vytvoření výchozího počtu jedinců, kteří se zobrazí v okně Jedinci (pokud je okno otevřeno). Zároveň se v menu aktivuje položka *Krok*.

Jeden krok modelu představuje jeden rok, který proběhne v populaci. Někteří jedinci se zpárují, někteří se rozejdou, někteří se narodí a někteří zemřou. Všechny tyto změny můžete sledovat v okně Jedinci, čas modelu a počet žijících jedinců se zobrazují i přímo v hlavním okně. Pokud necháte provést více kroků najednou, model zobrazí výsledek po provedení všech těchto kroků.

Také můžete *Zastavit model*, upravit některá nastavení (např. přidat nebo odebrat geny) a znovu jej spustit (bude spuštěn od začátku, neboli znovu bude vytvořen výchozí počet jedinců). Model můžete znovu spustit i přímo za běhu – jde o jakýsi restart modelu.

Aplikace

V menu *Aplikace* může být zajímavá volba *Restartovat*. Je to to samé jako zavřít aplikaci a znovu ji spustit. Pokud aplikace spotřebovává hodně systémových prostředků, ale je ovladatelná, je to zaručená možnost, jak je rychle uvolnit – ovšem za cenu ztráty dat modelu. Podobně funguje volba *Nová populace*, nicméně zde rychlost uvolnění prostředků závisí na dalších faktorech (má určitou setrvačnost).

Může se stát, že nastavíte takový model, jehož provádění bude pro Váš počítač příliš náročné. Vysoké nároky bude mít model s velkým počtem jedinců, ale např. i model, kde jedinci mají řádově vysokou porodnost, věk nebo vybíravost, případně pokud zadáte požadavek na provedení velkého počtu kroků. Uvědomte si prosím, že maximální hodnoty, které je možné nastavit, nejsou vždy fyzicky uskutečnitelné. Pokud je Vás počítač alespoň středně výkonný, mělo by být modelování běžných populací bez problémů. Pokud ale chcete zadávat výrazně ambiciózní data a nemáte zrovna nějaký superpočítač, počítejte s tím, že můžete na výsledek čekat ne milisekundy či sekundy, ale minuty, desítky minut či hodiny. To je dáno prostě a jednoduše tím, že v paměti Vašeho počítače jsou někde všichni jedinci včetně všech informací o nich a Váš procesor musí zpracovat každé narození, každé úmrtí i každé rozmnožování každého jedince. Pokud je těch jedinců třeba miliarda a každý má porodnost třeba v řádu milionů potomků, je logické, že chvilku trvá to zpracovat a že to zabere nějaké místo.

Příjemné a ničím nerušené používání programu a mnoho nových objevů Vám přeje

Rudolf Rosa, autor

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze

Použitá literatura

[1] Azrael: **Genetika - Základy genetiky, dědičnosti a evoluce** (online), *http://genetika.wz.cz/*

[2] RNDr. Eduard Kočárek, PhD.: **Genetika**, edice Biologie pro gymnázia (2004 Scientia, Praha, 1. vydání)

Použité softwarové prvky

- Microsoft Visual Studio 2008, Version 9.0.21022.8 RTM (C# Express)
- „Icons are from *http://www.freeiconsdownload.com*“

a některé další...

© 2008 Rudolf Rosa <rur@seznam.cz>, 1. ročník, letní semestr, programování (IP), prezenční bakalářské studium
Matematicko-fyzikální fakulta Univerzita Karlova v Praze