Manual d'usuari i de manteniment
Bioscena: Simulador d'entorns biològics evolutius

David Garcia
vokimon@jet.es

4 de juliol de 2000

Índex

Índex de figures

Índex de taules

Capítol 1

Introducció

1.1 Sobre Bioscena

`Bioscena` és una aplicació que permet simular entorns biólogics evolutius, és a dir, biosistemes on viuen organismes que evolucionen al llarg de generacions competint pels recursos.

L'eina ha estat desenvolupada al Departament d'Informàtica d'Enginyeria la Salle amb l'objectiu de fer-la servir de base per a futurs desenvolupaments de sistemes similars de Vida Artificial. Per aquest motiu, ha estat dissenyada pensant sobretot en la flexibilitat i en la facilitat d'ampliació futura.

El sistema en sí permet configurar molts paràmetres i presentar als organismes escenaris molt diversos.

A més, els mòduls del programa estan acoblats molt dèbilment, el que permet crear i intercanviar nous mòduls amb la mateixa funció que un d'existent.

Es podria reprogramar, per exemple, la geometria i composició del medi, el funcionament

intern dels organismes...

No es volia lligar els futurs sistemes basats en aquest a una plataforma en concret i el resultat ha sigut una aplicació molt portable. L'eina ha estat provada en els sistemes operatius MS-DOS, MS-Windows, Windows-NT, Linux i Sun-OS.

1.2 Objectius del manual

El present manual cobreix diferents nivells d'ús i coneixement sobre l'eina.

Per un costat, estan els usuaris que simplement volen probar l'eina. Els primers capítols descriuen les indicacions bàsiques d'instalació, i operació.

Si es vol experimentar amb l'eina, és necessari un coneixement una mica més profund del model conceptual del sistema amb la finalitat de reproduir, mitjançant la configuració, les condicions del sistema simulat i interpretar correctament els resultats obtinguts.

Si el que es vol és ampliar o modificar l'eina, per a cada mòdul, s'explica la seva implementació i es donen indicacions molt detallades de com fer-ne modificacions que s'integrin en el disseny modular del sistema.

El present manual es basa en la memòria del projecte de final de carrera de David García Garzón anomenat *Bioscena: Simulació d'un sistema biològic evolutiu amb interacció entre els individuus*. La finalitat dels dos textos es diferent. La memòria del projecte explicava el que s'havia fet, com i perqué. Aquest text ha esta reestructurat, actualitzat i adaptat per que serveixi de guia als usuaris i futurs mantenidors del sistema.

Capítol 2

Instalació de l'eina

2.1 Contingut de l'arxiu comprimit

`Bioscena` es distribueix en una arxiu comprimit. Cal descomprimir-ho mantenint la estructura de directoris que conté. Hi ha diversos directoris:

`Bin/` Conté l'executable per a MS-DOS generat amb DJGPP i els arxius de configuració. També hi ha alguns arxius de configuració alternatius d'exemple.

`Docs/` Conté alguna documentació útil pel manteniment com ara l'arxiu `TODO.txt`.

`Src/` Conté el codi font i els arxius necessaris per compilar en diferents entorns.

`Manual/` Conté aquest manual i els arxius necessaris per generar-ho.

`Utils/` Conté aplicacions complementàries.

2.2 Preparant l'execució

Per a l'execució del programa, només són necessaris l'executable i els tres arxius de configuració `bioscena.ini`, `agents.ini` i `opcodes.ini`. Els arxius de configuració han d'estar en el directori de treball de l'executable.

És possible que no estigui disponible l'executable per la plataforma escollida. Es pot generar tot seguint les instruccions de compilació de l'apartat 5.

L'executable necessita una consola de text de 80x50 que interpreti correctament seqüències de terminal ANSI.

Els següents apartats expliquen com fer-ho a diferents sistemes. Si, a la plataforma destí, no és possible treballar amb un terminal 80x50, encara seria possible canviar les coordenades dels objectes gràfics a l'arxiu font `BiosistemaProves.cpp`.

2.2.1 Terminal ANSI de 50 linies sota Windows 95

Per suportar les seqüències de control ANSI, cal haver iniciat l'ordinador o una sessió MSDOS amb la línia

```
DEVICE=C:\WINDOWS\COMMAND\ANSI.SYS
```

dintre del config.sys.

Per posar la pantalla a 80x50, si és un executable de DOS (DJGPP), cal crear un acces directe i, a les seves propietats, al separador 'Pantalla' especificar 50 linies.

Si és un executable Win32 per a consola (compilat amb Microsoft Visual C++), automaticament es posa a 80x50.

2.2.2 Terminal ANSI de 50 linies sota Windows-NT

Per fer servir les seqüències ANSI amb l'executable MS-DOS, cal seguir les següents passes:

- Si ja existeix l'arxiu `config.nt` al comprimit, cal copiar-lo al directori del executable, i saltar-se els dos següents passos.
- Copiem l'arxiu `c:\WINNT\SYSTEM32\config.nt` al directori de l'executable.
- L'editem i afegim la línia:

  ```
  DEVICE=$WINNT$\SYSTEM32\ANSI.SYS
  ```

- Obrim les propietats de l'executable DOS
- Premem el botó 'Avanzada' del separador 'Programa'
- A la capsa pel `config.nt` posem l'encaminament del personalitzat.

Per canviar el nombre de linies de la pantalla, si ho fem al mateix lloc que cal fer-ho a Windows 95, no en fa cas. Cal seguir les següents passes:

- Executar el programa
- Accedir a l'opcio propietats del menu de sistema de la finestra. Surtirà un diàleg de propietats diferent que el de Windows 95.

- Cal canviar a una lletra suficient petita, si no, ignorarà la resta de canvis.
- Augmentar el tamany del buffer de sortida i de l'àrea de pantalla fins a 50 línies com a mínim.
- En aceptar, posar que guardi les opcions per a proximes execucions.

Si tot va bé, la proxima vegada que ho executem, sortirà bé.

Un executable Win32 es posarà automaticament a 80x50, però, no s'ha provat encara com fer funcionar l'ANSI.SYS amb un executable Win32 sota Windows NT.

2.2.3 Terminal ANSI de 50 linies sota Linux

A linux ja hi ha ANSI per defecte tant a terminals en mode text com a X-terms.

Per obtindre un terminal 80x50 aquí hi han unes solucions.

Xterms	Simplement, cal augmentar el tamany de la finestra fins que hi capiguen 80x50 linies.
Terminals ordinaris	Cal especificar al programa d'arranc `lilo` el mode gràfic que fan servir els terminals. Consulta el manual.

Capítol 3

Model conceptual

3.1 Diagrama de blocs

El sistema consta de diversos elements principals, representats a l'esquema 3.1, cadascun, amb unes funcions determinades dintre del sistema.

Biosistema: És l'objecte coordinador de la resta d'elements. Les seves funcions són:

1. Multiplexar l'execució concurrent dels diferents organismes.
2. Demanar als organismes les instruccions que volen executar.
3. Realitzar les accions sobre la resta d'elements del sistema que aquestes instruccions requereixin.
4. Mantenir dintre d'uns mínims la població de la comunitat introduint nous organismes quan aquesta baixa.
5. Accionar els agents ambientals encarregats de variar el medi al llarg del temps.

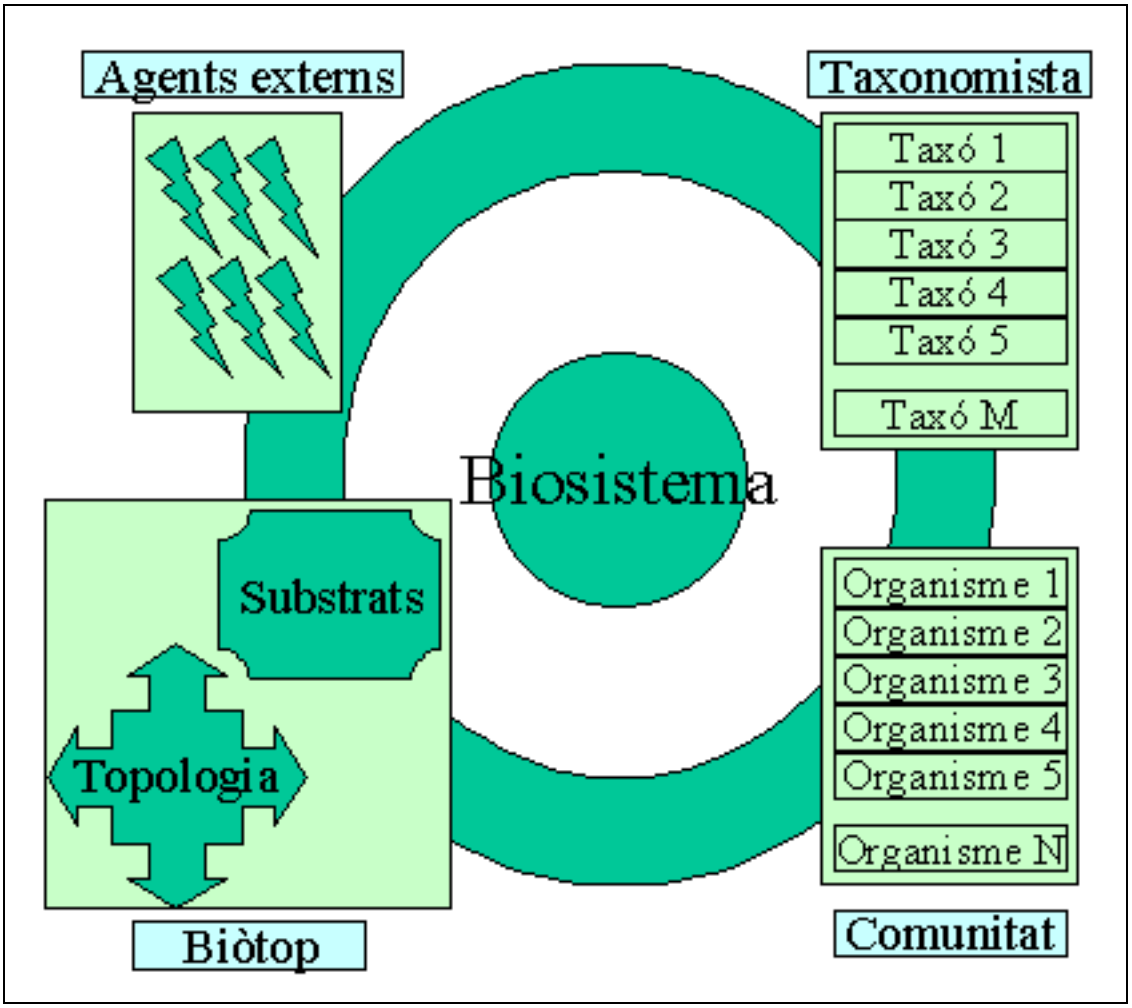

Figura 3.1: Esquema conceptual del sistema

Topologia: Determina la geometria del medi on viuen els organismes. Les seves funcions són:

1. Associar un identificador numèric a cada posició dins del substrat
2. Establir interconexions entre les parceles de substrat
3. Determinar moviments, direccions, camins... i tota l'operativa que té a veure amb la geometria (topologia) del medi segons aquestes interconexions.

4. Proporcionar l'accés, mitjançant l'identificador de posició, a les propietats del medi en aquesta posició.

Substrat: Determina les propietats del medi en una posició donada. Les seves funcions són:

1. Determinar si la posició l'ocupa un organisme i, en cas afirmatiu, quin és l'organisme ocupant.
2. Contenir els nutrients lliures al medi.
3. Altres característiques associades a la localitat que es vulguin afegir més endavant.

Agents Ambientals: Determinen l'evolució de certs paràmetres (posició, composició, probabilitat, estacionalitat...) que intervenen en les propietats dels elements del sistema al llarg del temps. Les seves aplicacions són:

1. Afegir o eliminar els nutrients lliures del medi de forma controlada.
2. Generar expontàniament organismes (no implementat encara).
3. Mutar els organismes depenent de la posició (no implementat encara).

Comunitat: Representa al conjunt d'organismes que viuen al biòtop. La comunitat compleix amb les següents funcions.

1. Associar un identificador a cada organisme dintre de la comunitat
2. Afegir-ne o extreure'n organismes.

3. Mantenir la informació referent a l'organisme que el relaciona amb el seu entorn, com ara, la posició, el grup reproductiu al que pertany... (Informació externa de l'organisme).
4. Proporcionar l'accés, mitjançant l'identificador d'organisme, tant a la informació externa com al propi organisme.

Organismes: Representen als individus que viuen al biosistema. Contenen la informació genètica i les estructures internes que els fan anar. Les seves funcions són:

1. Oferir instruccions al biosistema del que volen fer.
2. Proporcionar al biosistema accés als registres de fenotip.
3. Proporcionar al biosistema operacions per modificar el seu estat intern.
4. Generar descendents.

Taxonomista: Discrimina entre diferents grups reproductius a dins de la població i manté informació sobre ells per facilitar l'anàlisis. Les seves funcions són:

1. Discriminar els diferents grups reproductius.
2. Mantenir informació històrica sobre l'aparició de grups reproductius.
3. Mantenir un cens de la població de cada grup reproductiu.
4. Altres funcions d'anàlisi que es vulguin afegir en el futur, per exemple, per esbrinar la dieta o els agressors normals de cada grup reproductiu.

3.2 Funcionament

Un cop identificats els diferents elements, s'explica a continuació un resum de com funciona el sistema.

El món implementat és una matriu de caselles amb les vores conectades formant un torus. A cada casella hi cap un organisme i un nombre limitat de nutrients. Cada nutrient té un número associat que indica el seu tipus.

El sistema de control dels organismes es basa en l'execució condicional de seqüències d'instruccions. Cada seqüència d'instruccions s'executa, o no, depenent del contingut actual d'uns registres fenotípics que té cada organisme. El biosistema modifica aquests registres en executar les instruccions la qual cosa permet realimentar el sistema de control.

Cada organisme té un pap on acumula nutrients. Els pot fer reaccionar químicament per obtindre energia o nutrients de diferent tipus. També pot expulsar-los fora del seu cos.

Un organisme pot detectar a un altre organisme tot mirant el contingut dels registres fenotípics. Pot detectar nutrients del seu entorn o identificar els nutrients del pap. Aquestes accions sensorials modifiquen el contingut dels registres fenotípics propis amb informació que pot fer-se servir en posteriors instruccions.

El sistema de control d'un organisme es transcriu a partir del seu cariotip (conjunt de cromosomes). Els organismes es reprodueixen mitjançant la instrucció de mitosi. Durant la mitosi hi ha una certa probabilitat de que el cariotip muti en copiar-se al fill. El fill es porta part dels nutrients i l'energia del pare. Durant la mitosi, el pare escriu una primera dada a un dels registres de fenotip del fill.

Aquesta petita explicació hauria de ser suficient per entendre qué està pasant a la simulació. El comportament descrit, es pot modificar tot reconfigurant el sistema o modificant el codi font, però, per fer-ho s'aconsella llegir els capítols que parlen més a fons del model per configurar-ho o els capítols que parlen de la implementació per modificar el codi.

Capítol 4

Operació normal

4.1 La pantalla

Quan executem el programa, un cop carregats els fitxers de configuració, s'ens presenta una pantalla d'ajuda a la que podem tornar sempre que volguem prement 'H' i return. Aquesta pantalleta explica breument les comandes que hi ha. Per començar a executar la simulació premem return.

El que es pot veure es una pantalla semblant a la de la figura 4.1. El de la esquerra és un mapa del món, el de la dreta és un gràfic comparatiu dels organismes que hi ha vius en cada moment. De banda d'aquests dos elements, hi ha un indicadors del tamany de la població, el temps transcorregut i les coordenades de la posició de la cantonada superior esquerra del mapa.

Per entrar qualsevol comanda, podem entrar una tecla, en aquest moment ens sortirar un prompt `Comanda:` que ens permet visualitzar i corregir la nostra entrada. Un cop feta premem return i s'executarà.

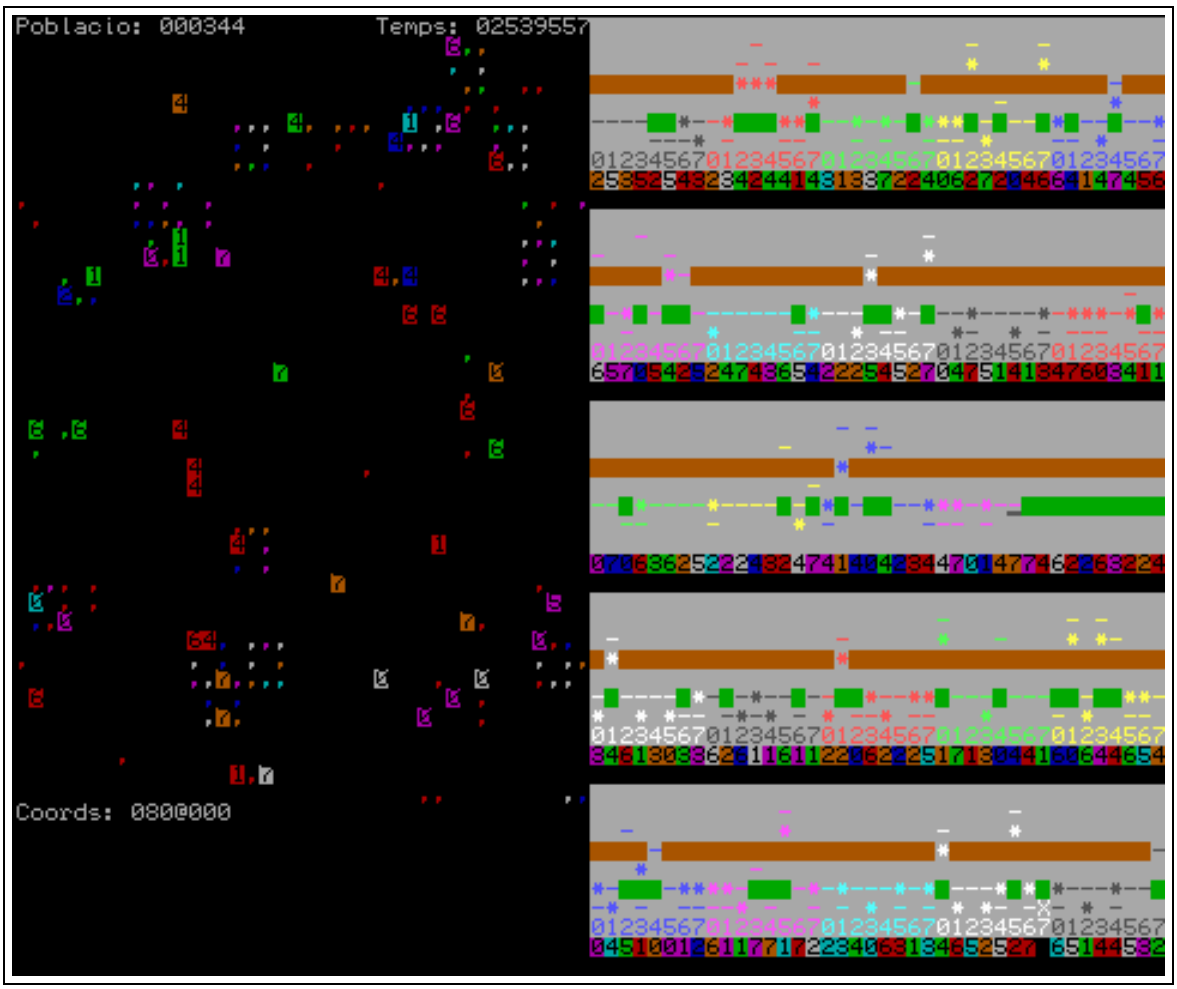

Figura 4.1: Pantalla Mapa/Comparativa

4.2 El mapa del món

El mapa representa dues coses:

- Els organismes, representats per números de colors. Un número i un color diferencien cada grup reproductiu (espècie).
- La quantitat de nutrients en cada posició, representada per un punt de color. Per

ordre de magnitud, els colors són: negre(0), vermell(1), verd(2), taronja(3), blau fosc(4), lila(5), cyan(6) i blanc fosc(7).

Si el món representat és més petit que la zona del mapa, aquest es repetirà en forma de mosaic, la qual cosa ajuda a veure les conexions de la superfície toroidal. Si és més gran, podem fer servir el teclat per veure les altres zones en direcció nord(W), sud (S), est(D) i oest(A). La lletra en minúscula avança una posició, en majúscula avança una pantalla sencera.

L'indicador de coordenades ajuda a saber on som.

Per desactivar el mapa i deixar més espai pels altres elements gràfics cal premer a la 'M' majúscula i retorn. Per tornar-ho a activar es prem la 'm' minúscula.

4.3 El gràfic comparatiu d'organismes

En quant al grafic conparatiu, representa per a cada organisme

- l'energia (amb el símbol '$*$'),
- l'edat (el símbol '$-$'), i
- el grup reproductiu (espècie), amb la mateixa codificació que el mapa

L'escala que es fa servir per l'energia i l'edat es logarítmica o linial segons s'hagi configurat. Per defecte, és logarítmica per tots dos valors.

El factor d'escala de les gràfiques és dinàmic, això vol dir que canvia a mesura que el valor màxim creix o decreix. Per tenir una referència, la banda de color verda que apareix al gràfic representa el valor 255 per la edat i, la taronja, 65535.

Cada organisme té durant la seva vida un lloc fixe al gràfic, que es correspon amb un número en base octal que l'identifica. El numero que es trova a la primera filera de números és el digit octal de menys pes, el color indica el segon. Podem veure un organisme per dins prement la 'v' i escrivint el seu codi octal.

Quan un lloc en la gràfica ja no està ocupat per cap organisme, s'indica amb una `X`.

Per desactivar la comparativa d'organismes i deixar més espai pels altres elements gràfics cal premer a la 'O' majúscula i retorn. Per tornar-ho a activar es prem la 'o' minúscula.

4.4 El gràfic comparatiu de grups reproductius

El gràfic comparatiu de grups reproductius no es veu quan comença el programa donat que comparteix la zona en la pantalla amb el gràfic comparatiu d'organismes.

Per activar la comparativa de grups reproductius, cal prémer 't' i retorn. Per desactivar-ho i deixar més espai pels altres elements gràfics cal premer a la 'T' majúscula i retorn.

Els grups reproductius estan ordenats al gràfic per antiguitat. Generalment la part final del gràfic l'ocupen grups reproductius que encara no estan consolidats i que segurament desapareixeran.

La gràfica reflecteix altres dos dades una és el nombre total d'organismes del grup repro-

ductiu que han existit al llarg de la simulació, i l'altre dada és el nombre tota d'organismes que són vius en aquell moment donat.

4.5 Detall de les accions

Per veure un log amb el detall de les accions que es produeixen al biosistema, cal prèmer la tecla 'l' minúscula. Per treure el log, cal prèmer la 'L' majúscula.

4.6 Pas de visualtització

La visualització enlenteix molt el sistema, si es vol anar més ràpid es pot saltar un nombre de intervals de temps determinats. Per indicar un pas de visualització més relaxat cal prèmer la 'j' i després el número de passos que ens volem saltar entre visualització i visualització.

A les proves, s'ha pogut comprovar que, en un Pentium II, un pas de 10 fa que els moviments dels organismes semblin més naturals. Un pas de 100 no et permet veure en detall qué és el que passa però, és suficient per advertir a temps els canvis bruscos que es donen sovint en la població.

Cal recordar que les unitats de temps que es calculen són les de simulació, i poden durar més o menys segons el tamany de la població.

4.7 Important i exportant biosistemes

Les simulacions són molt llargues i sovint és molt útil poder reempendre una simulació interrompuda. S'ha implementat un mecanisme senzill d'exportació i importació de biosistemes que permet fer-ho.

Si en un punt de l'execució escrivim al prompt `Xmimundo.bio Bioscena` escriurà un arxiu anomenat `mimundo.bio` amb tot el contingut actual del sistema amb excepció dels paràmetres de configuració i els agents.

Si en una darrera execució volem recuperar aquest estat només cal fer `xmimundo.bio` amb la 'x' en minúscula.

Aquest sistema és provisional i té un parell de deficiències. Els paràmetres de configuració i l'estat dels agents, no estan inclosos en el sistema de serialització. Una incoherència entre els paràmetres del sistema guardat i els paràmetres actuals, pot provocar errors. Es recomana mantenir una còpia del fitxer `bioscena.ini` amb els paràmetres del biosistema exportat.

4.8 Visualitzant, important i exportant organismes

Com que la interfície és molt primitiva, és una mica complex referenciar els organismes. Un organisme es referencia amb la seva posició en la comunitat. Per poder deduir aquesta posició de forma més ràpida a partir de la informació que dona la gràfica comparativa d'organismes, es codifica la posició en notació octal.

Per exemple, imaginem que volem visualitzar l'ultim organisme que apareix a la grafica comparativa que surt a la figura 4.1 identificat amb un 7 gris. El 7 ens dona el primer dígit octal, el gris ens indica que el segon dígit és un zero, segons la relació de colors: gris(0), vermell(1), verd(2), groc(3), blau(4), magenta(5), cyan(6) i blanc(7).

Com pertany al quart grup de color gris del gràfic, la resta de dígits és un 3 (el primer grup es zero). Així doncs aquest organisme té un identificador octal `307`.

Per obtindre més informació sobre aquest organisme concret cal posar al prompt `v307`

Una altra funcionalitat molt útil de cara a l'experimentació és la possibilitat d'aïllar un organismes pel seu anàlisis o per introduir-lo en un altre biosistema.

Si s'escriu al prompt `C12mibicho.org` estem exportant l'organisme que ocupa la posició octal 12 en la comunitat cap a l'arxiu `mibicho.org`.

Per recuperar-ho en aquest o altre biosistema s'escriu `cmibicho.org` amb la `c` en minúscula.

4.9 Modificant la configuració durant la simulació

El que es troben el organismes al medi al llarg del temps de simulació ve determinat pels fitxers de configuració `agents.ini` (agents ambientals).

A vegades, és complex programar una configuració d'agents per fer que variï molt al llarg del temps. Pot resultar més fàcil modificar-ho quan ho necessitem, carregant un arxiu de configuració d'agents amb un contingut diferent en mig de l'execució.

La tecla `r` recarrega el fitxer de configuració d'agents `agents.ini` i `R`, en majúscules, carrega

el fitxer de configuració d'agents que s'indiqui a continuació. Si no s'especifica cap o un d'erroni, donarà un error i es continuarà sense cap agent que actui sobre el biosistema. Això és molt útil per simular un cataclisme o una crisis, doncs desapareixen tots els agents que diposisten nutrients.

La forma d'actuar recomanada es tenir un arxiu `agents.ini` amb una situació normal, i diversos arxius amb configuracions d'agents que implementin situacións excepcionals.

4.10 Altres comandes

p	Congela el biosistema, permetent fer altres operacions sense que la situació canviï
P	Descongela el biosistema
e	Fa que els grups reproductius de pare i fill siguin diferents en fer una mutació
E	Fa que els grups reproductius de pare i fill siguin iguals encara que hi hagi una mutació
H/h	Presenta la pantalla d'ajut
Q/q	Surt del programa

Capítol 5

Compilació

Aquest capítol explica com compilar el codi font per generar un executable. És possible que a la distribució ja vingui un executable per a la seva plataforma i no calgui generar-ho.

5.1 GCC/DJGPP

Aquest compilador i, sobretot, la seva versió per a DOS, el DJGPP, ha sigut el més testejat. De cara a compilar cal una versió superior a la 2.7 i, per fer-ho sense problemes cal la 2.95. Cal assegurar-se que estan instalades les llibreries estàndars de C++.

Per aquest compilador, es proveeix un fitxer `makefile`. El procediment és el següent:

1. Adaptar les següents variables del makefile als noms dels executables al sistema on es compila:

   CC Nom del compilador de C. (`gcc`, `cc`, `egcs`...)

   CPPC Nom del compilador C++. (`c++`, `cxx`...)

RM Comanda per esborrar arxius

EXEC Nom del executable generat

2. Si cal, comenta les crides a `buildnum` que es fan als makefiles (veure apartat 5.4)

3. Si es vol optimitzar per una maquina en concret, cal afegir '-march=ix86' a la variable CFLAGS del makefile; on x=3,4,5,6

4. Executar la comanda 'make'

El makefile té en compte de forma automàtica qualsevol arxiu `.CPP` i `.C` que s'afegeixi al directori. Si es modifiquen les dependències cal executar la comanda `make dep` per actualitzar el fitxer `.depend`. Si aquest fitxer no existeix, cal crear-ho manualment (`echo >.depend`) per que el makefile no doni errors i després executar `make dep`.

5.1.1 Problemes

- Generals:
  - Assegurat de que el fitxer `.depend` existeixi, encara que sigui buit. Si no crea'l amb: `echo >.depend; make dep`
  - Si es fa servir una versió del compilador o de les llibreries de C++ inferior a la 2.8, dóna errors a una línia de la funció `CCariotip::ocupaCromosomes` del fitxer `Cariotip.cpp`. A sota de la mateixa línia hi ha un fragment de codi comentat que pot substituir la línia del error, i indicacions de com arreglar-ho.
  - Per problemes amb relacionats amb el programa de generació de números de compilació veure l'apartat 5.4.

- Sistemes Unix:

  - Els arxius de text del comprimit estan en format MS-DOS. Per això, cal treure els salts de carro adicionals que impideixen la compilació. La forma més ràpida de fer-ho és fer servir l'opció `-a` quan es descomprimeix el comprimit amb la utilitat `unzip`.

5.2 Microsoft Visual C++

Dins del directori `Src/` hi ha els tres arxius del projecte per a Microsoft Visual C++: `Bioscena.dsw`, `Bioscena.dsp` i `Bioscena.opt`.

Aquest darrer arxiu no és necessari però, és útil pel manteniment perqué ordena les classes en carpetes al *Infoviewer* segons la seva funció. És imprescindible si es vol treballar amb les més de 70 classes que hi ha.

5.2.1 Problemes amb versions antigues

Amb versions antigues del compilador es donen un parell de problemes:

- Hi ha problemes amb la biblioteca estàndard C++ que proporciona l'entorn degut a conflictes entre la implementació dels templates que fa el compilador i la forma de fer-los servir en la biblioteca, implementada per HP. Es pot compilar l'eina si es modifiquen els fonts de la biblioteca, ficant entre comentaris les funcions que donen errors.

- No suporten massa bé punters a funcions membres. El codi que fa servir punters a funcions membres, tot i que compila, genera un assembler no massa correcte i dóna errors d'execució.

5.3 Compilant amb altres compiladors

Per compilar amb altres compiladors, es requereix com a mínim un compilador que suporti templates i punters a funcions membres i que tingui implementada de forma suficientment amplia la llibreria estàndar C++ segons s'especifica en l'estàndard ISO/ANSI [ISO]. Concretament es fan servir les capçaleres estàndard `<algorithm>`, `<functional>`, `<list>`, `<vector>`, `<map>`, `<deque>`, `<string>`, `<iostream>`, `<fstream>`, `<iomanip>` i `<strstream>`.

5.4 Generació de números de compilació

Cada compilació porta associada un número d'ordre i la seva data i hora d'inici.

Per generar números de compilació, cal el programa `buildnum` que es troba al directori `Utils` de la distribució. El codi font també està disponible a la pàgina web http://www.salleurl.edu/~is04069/Codders.

Si no es vol fer servir aquest programa, cal treure les crides que s'hi fan al `makefile` o als fitxers de projecte.

Capítol 6

Exemple d'execució: Biosistema amb múltiples climes

6.1 Condicions de la simulació

Els fitxers de configuració que venen amb l'executable modelitzen una simulació d'exemple.

És un biosistema de 160x80 on s'han diferenciat algunes regions amb l'objectiu que els diferents organismes s'especialitzin, a la regiò on viuen. Es vol demostrar que cada grup d'organismes, potser i tot d'origens genètics comuns, es pot especialitzar per viure en cada zona.

Així doncs s'han diferenciat, mitjançant els agents ambientals quatre zones amb *climes diferents*:

A Una taca de 20 posicions de radi que dipositava nutrients de tipus A amb molta freqüència, però, que tenia períodes d'inactivitat.

B Una taca de 20 posicions de radi que dipositava nutrients de tipus B amb un xic menys de freqüència, però, de forma constant.

C Una taca molt dispersa però molt ampla (60 posicions de radi) que dipositava nutrients de tipus C.

D Per la resta del biòtop es deixen petits grupùscles molt densos de forma aleatòria.

6.2 Primigenis

El primer que cal dir és ha de passar cert temps de simulació perqué l'atzar faci aparèixer un organisme vàlid que es sapiga reproduir. Fins llavors els organismes, apareixen aleatòriament i desapareixen quan se'ls acaba l'energia.

La millor forma veure quan apareix un organisme vàlid és anar mirant el gràfic de grups reproductius. Al començament podem trobar quatre tipus d'organismes:

1. Organismes inestables que no trascendeixen.
2. Organismes inmortals que sobreviuen però no es reprodueixen
3. Organismes que es reprodueixen amb una tasa de creixement al voltant de 1.
4. Organismes que es reprodueixen amb una tasa de creixement major que un.

Els organismes inmortal sobreviuran, però, segurament no sobreviuran quan es desenvolupin organismes del quart tipus.

Els organismes amb tasa de creixement al voltant de 1, són organismes que poc després de reproduir-se es moren o que el seus descendents no sobreviuen. Aquest tipus d'organisme té l'esperança de que un fill seu muti i esdevingui un organisme primigeni, però amb tota probabilitat aquesta mutació acabarà sent negativa.

Un cop apareix un grup reproductiu tal com el quart cas, aquest primer grup acaba sent el dominador del biosistema amb més o menys velocitat, donat que la probabilitat de generar al atzar un individu bó és molt petita, com hem vist, i la competència en aquest punt ja seria massa ofegant.

Així doncs, els principis de supervivència que fan servir els primers organismes que aprenen a reproduir-se eficientment són aquells en els que es basaran la majoria dels organismes que apareixin a partir d'aquell moment.

Si es vol obtenir organismes amb principis de supervivéncia diferents, un mètode que s'aconsella és fer diverses simulacions i exportar els organismes que aprenen a reproduir-se en cadascuna. Si els importem a un altre biosistema, tenen més probabilitat de conviure.

6.3 Especialització

Degut al caracter no determinista de les simulacions, els resultats poden diferir molt, tot i així, el que es fa a continuació és resumir alguns aspectes observats a les simulacions basades en aquesta configuració que es donen de forma majoritària o que resulten interesants.

Entre la zona C i D al llarg del temps no hi ha masses diferències tret una diferència en la densitat dels organismes. Diguem que el tipus d'organisme que trionfa a D (que

normalment són els que millor cerquen l'aliment) acaba dominant en C.

A les altres dos zones, el resultat més clar es que a la llarga (aproximadament 1.500.000 d'unitats temporals) els organismes de A i B, que sempre són molt voraços i competitius, acabaven no movent-se. Com que la font de menjar que tenien no es mou, si ells es mouen es temps que perden de menjar.

A les zones A i B, al començament hi ha molta densitat d'organismes la qual cosa crea que en sorgir comportaments d'agressió aleatòria, és a dir sense saber si hi ha ningú a on ataquem, aquests siguin beneficiosos.

En A els comportaments d'agressió, de banda d'un complement per la nutrició resultaven una forma de debilitar a la competència.

En canvi, en B, l'augment de la voracitat dels organismes causa que durant el període de temps en que no es diposita aliment no hi hagi res a menjar. En aquesta situació, l'agressió deixa de ser un complement i passa a ser vital. De tal forma que en B, els organismes que fan servir els sensors de presència comencen a tindre avantatge i es desenvolupen comportaments d'aquest tipus.

Després d'això, la densitat en B, descendeix i els organismes que hi sobreviuen s'alimenten del medi, quan hi ha menjar i dels organismes que, atrets per la quantitat de nutrients, s'apropen des de C i D.

6.4 Pressió selectiva depenent de la densitat

Les primeres experiències amb el prototip van servir, per clarificar un dilema que em plantejava al començament de les proves: Cal fer un biosistema molt restrictiu per que hi hagi pressió evolutiva i així es desenvolupin comportaments bons o cal fer-ne un de menys restrictiu per que es puguin desenvolupar els organismes primigenis amb major facilitat.

Realment vam comprovar que configurant un biosistema on els primigenis puguessin desenvoluparse amb facilitat, quan la població començava a pujar, els mateixos organismes, en la competència pels recursos limitats es creaven la pressió evolutiva.

6.5 Distribució d'edats i organismes inmortals

Un altre tema relacionat és el de la distribució de les edats i el problema dels organismes inmortals apuntat per Todd en [Tod93]. El problema es basa en el fet de que, si no limitem artificialment l'edat dels organismes, ens podriem trobar situacions en les que hi haguessin organismes que no morissin mai.

Beleg i Menezer [MB95] apunten el fet de que sense limitar artificialment l'edat dels organismes a LEE, emergeixen distribucions d'edats estables semblants a les corbes de la equació clàssica d'Euler-Lotka. Això és degut al fet de que els organismes comparteixen recursos finits i, en conseqüència, com hem vist a l'apartat anterior, la pressió selectiva depen de la densitat de la població.

Quan la població es manté estable deixa de crèixer, la distribució en edats tendeix a ser tal que la fracció d'organismes amb una determinada edat equival a la probabilitat de que un

organisme arribi a aquesta edat. [Ste]

En el nostre cas, el problema és que, a diferència del que passa al sistema LEE, els organismes de Bioscena no es reprodueixen automàticament quan arriben a un nivell energètic, sinó que han de desenvolupar en el seu codi genètic la capacitat de reproduir-se només quan tenen prou energia.

Mentre que no hi ha cap organisme que sapiga reproduir-se adientment, a grans trets hi ha dos tipus d'organismes. Uns que proven de reproduir-se abans de tenir l'energia necessària i en fer-ho moren. D'altres que no proven mai de reproduir-se i només menjen sense que ningú els faci competència.

Aquests darrers es podrien considerar organismes inmortals. Però, en el moment en el que apareix un organisme que es sap reproduir correctament, la població puja explossivament i arriba al punt en que comença a haver pressió selectiva. En aquest punt, ja es donen les condicions perquè aparegui el comportament emergent descrit pels autors del LEE i com he pogut contrastar es dóna.

Capítol 7

Representació del medi

7.1 Visió general

El present apartat descriu el funcionament del medi on viuen els organismes i alguns detalls de disseny i d'implementació. De cara a permetre ampliar fàcilment el model, s'ha volgut fer un disseny del medi que permeti adaptacions a futures necessitats de modelatge. Es descriu el model genèric, les particularitzacions bàsiques implementades i els passos a seguir si es volgués implementar particularitzacions pròpies.

Malgrat que el model és general, està restringit a biòtops (medis) que contènen posicions discretes. Això implica dues coses:

- Les posicions dins del biòtop estan quantitzades. No hi ha més que un nombre limitat de posicions a diferència de l'espai continu real amb infinites posicions possibles.
- Es pot considerar una posició discreta com a representació d'una zona limitada del substrat continuu real, però, les propietats dins d'aquesta zona del substrat es man-

tenen uniformes.

La discretització de l'espai juntament amb la discretització del temps és una característica comuna a la majoria de sistemes de simulació i vida artificial. L'única forma de limitar els efectes artificiosos que això pot crear és fer una quantització tan petita que el seu efecte quedi minimitzat.

El model general per al biòtop és un model que consta de posicions discretes amb les seves corresponents propietats i que ténen relacions de veinatge amb les altres posicions segons una topologia.

Així doncs, tenim dos elements que podem modelar independentment.

- **Posicions del substrat:** Elements discrets que indiquen les propietats d'una zona del biòtop.
- **Topologia:** Controla les relacions de veinatge i la identificació de les posicions per part de la resta del sistema.

Combinant aquests dos elements, es pot obtindre un conjunt molt ric de biòtops.

7.2 Topologies

La topologia determina les relacions de veinatge entre les cel·les. Si les posicions del biòtop fòssin nodes d'un graf, la topologia representaria els vertexs que els uneixen.

Per exemple, podem adaptar la topologia per convertir-la en una topologia 2D, molt vàlida per simular biosistemes terrestres sense estratificar, o podem adaptar-ho a una topologia 3D que és més realista per simular medis fluids, com l'aigua o estratificats com els boscos.

Dissenyar una topologia comporta decidir que és el que representen geomètricament els identificadors de posició i els identificadors de desplaçament, i, segons aquesta decisió, especificar quin ha de ser el comportament de les funcionalitats de la topologia que relacionen posicions i desplaçaments.

Aquestes funcionalitats es basen en dos relacions principals: Donats un desplaçament i una posició, la topologia ha de poder determinar quina és la posició destí, i, donades dos posicions, la topologia ha de poder determinar el desplaçament entre elles.

Com que el nombre de cel·les sovint serà limitat, el conjunt de cel·les formaran una regió limitada. En aquests casos, el significat geomètric dels desplaçaments tambe inclou la decisió de quines són les veïnes de les cel·les situades a les vores.

Per exemple, considerem una topologia 2D rectangular. Si féssim que les cel·les limítrofes no tinguin veïnes més enllà dels límits ens trobarem davant d'una regió limitada. Si fem que les cel·les d'un dels costats es conectin amb les cel·les del costat oposat, obtindrem una topologia de superfície cilíndrica. Si fem el mateix amb tots quatre costats obtindrem una topologia de superfície toroidal.

TODO: Solucions per a les vores

Figura 7.1: Conexions de les vores

El conjunt de serveis que una topologia ofereix als seus clients són:

- Donar l'identificador de la posició destí a partir de l'identificador d'una posició origen i d'un desplaçament. Possibilita els moviments.
- Donar el desplaçament que apropa una posició d'origen donada a una posició destí per l'itinerari més curt. Possibilita l'orientació per objectiu.
- Donar el desplaçament oposat a un altre, quan sigui aplicable. Possibilita l'orientació per evasió.
- Donar l'identificador d'una posició escollida aleatòriament. Possibilita el posicionament aleatori.
- Donar la posició destí despres de n desplaçaments aleatoris des d'una posició origen Possibilita el moviment aleatori en l'entorn.
- Determinar si un identificador de posició és vàlid dintre de la topologia. Facilita la depuració (debugging) i implementar el posicionament aleatori de forma general però, poc òptima.

7.3 La topologia toroidal

La topologia bàsica que ja és implementada al prototip és una topologia 2D toroidal.

S'ha escollit una topologia en 2D per al primer prototip donat que és molt fàcilment representable en pantalla i a més, els càlculs dels desplaçaments surten molt senzills i òptims. L'optimització de les funcions de la topologia és crucial donat que s'utilitzen de forma intesiva al sistema.

S'ha triat fer-la toroidal perqué, en els primers prototipus s'ha preferit no afegir complexitat al medi tot afegint situacions excepcionals per als organismes com ara són les vores.

Per optimitzar la implementació, hem fet que la veïna directa d'una posició a l'extrem dret sigui una posició a l'extrem esquerre però, no pas la que està a la mateixa línia sinó la que està una línia abaix. Així, tots els desplaçaments es resolen amb una única suma o resta i només cal ajustar quan la posició destí es surt del domini de les posicions existent.

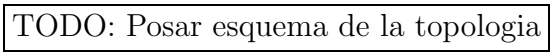

Figura 7.2: Topologia toroidal

7.3.1 Desplaçaments

Cada posició té 8 cel·les veines directes. Un desplaçament a qualsevol d'aquestes 8 cel·les es pot codificar amb 3 bits com indica la figura 7.1

100	000	001
101	Origen	010
110	111	011

Taula 7.1: Veïnes directes d'una posició

La concatenació de N desplaçaments bàsics aleatoris tendeix a formar una distribució normal entorn al centre. Com els vectors de desplaçament tenen 32 bits podriem codificar fins a 10 desplaçaments bàsics consecutius en un vector de desplaçament. Però, com ens serà molt útil poder activar i desactivar cada desplaçament bàsic, farem servir un quart bit per a cada bàsic per dir si està habilitat o inhibit el desplaçament, i en un vector hi caben, doncs, 8 desplaçaments bàsics amb els seus bits d'inhibició.

h0	d0	h1	d1	h2	d2	h3	d3	h4	d4	h5	d5	h6	d6	h7	d7
1	101	1	101	1	010	1	110	1	110	1	110	1	110	1	110

Taula 7.2: Codificació dels desplaçaments al biòtop

Si cal considerar cap ordre en el càlcul dels desplaçaments bàsics, es fa de més significatiu a menys.

La codificació dels desplaçaments bàsics de la taula 7.1 s'ha fet de tal manera que, si invertim bit a bit un vector de desplaçament, obtenim un desplaçament invers. És a dir, si invertim tots el bits d'un vector de desplaçament `desp` a excepció dels bits d'inhibició amb l'expressió `(desp XOR 0x77777777)`, en dòna el desplaçament invers.

7.3.2 Posicions

L'altre funció important de la topologia és assignar a cada posició un identificador únic dins de la topologia. La resta del sistema farà servir aquest identificador per referenciar una posició i, en cas de necessitar-ho, demanarà a la topologia quin és el substrat per aquesta posició.

En la present implementació s'han fet servir enters del 0 a N-1 com a identificadors de les posicions, on N es el nombre de cel.les. Aquests identificadors ens permeten fer els desplaçaments de forma molt òptima si assignem el números per ordre a les cel·les de cada fila. Per calcular l'identificador de la posició destí, només cal afegir (o treure), a l'identificador de la posició origen, un número, que depèn del desplaçament, i ajustar el resultat en cas de sortir-se de límits.

7.4 Substrats

Un substrat particularitza les propietats del medi per a una posició. El substrat pot tenir propietats diferents segons la complexitat que desitjem per al medi. No hi ha cap restricció pel disseny dels substrats de cara a que es pugui muntar un biòtop amb ells.

A continuació, es descriu com està definit el substrat bàsic implementat a l'aplicatiu. Les dos propietats més importants del substrat són qui ocupa el substrat, i quins nutrients hi han.

7.4.1 Ocupants

S'ha restringit l'ocupació de les posicions per part dels organismes a un sól organisme per posició i a una sola posició per organisme. La raó ha estat fer possible referenciar un organisme per la seva posició. Això simplificarà, a les relacions entre organismes, com referir l'altre organisme. També, com a avantatge adicional, simplifica la interfície amb l'usuari per seleccionar un organisme seleccionant la seva posició.

7.4.2 Nutrients

En quant els nutrients que hi pot haver en una mateixa posició del substrat, cada posició tè un nombre de nutrients màxim definit. Quan s'afegeixen nutrients per damunt d'aquest nombre, els nutrients més antics desapareixen.

Els nutrients estan diferenciats qualitativament amb un sencer que indica el seu tipus.

La recollida de nutrients, es fa amb un patró de cerca pel tipus de nutrient i una tolerància a nivell de bit segons la funció de compatibilitat estàndard. Com s'explica a l'apartat 14.8.1, aquesta funció retorna cert si

```
((Patro^Clau)&Random&~Tolerancia)==0
```

, de tal forma que un 1 a una posició de la tolerància significa que, encara que no es correspongui aquest bit no es tindrà en compte. La cerca es fa des dels més nous fins els més vells.

Capítol 8

Agents ambientals

8.1 Trets generals

Els agents ambientals són objectes que disparen una acció determinada quan són cridats. La majoria d'accions es produeixen sobre el biòtop, sobre l'estat d'altres agents ambientals o sobre paràmetres globals de configuració.

El paper dels agents ambientals a dins del sistema és permetre a l'usuari controlar com evolucionarà l'entorn on es mouran els organismes de cara a obtindre resultats que s'hi puguin contrastar. Per sí mateix, el conjunt d'agents implementats permet configuracions molt complexes, però, a més, ofereix un seguit d'eines molt útils per que l'usuari-programador pugui ampliar aquests agents.

Quan un agent es cridat per realitzar la seva acció, es diu que l'agent ha sigut **accionat**. Quan algú requereix un valor contingut en l'estat de l'agent es diu que l'agent ha sigut **consultat**.

Direm que un agent A té com a **subordinat** a un altre agent B si és A qui acciona a B. Ho representarem així: $A \rightarrow B$. L'estructura de subordinació ha de ser un arbre on els subordinats són els fills d'aquell a qui es subordinen.

Direm que un agent A és **depenent** d'un altre agent B si A, quan és accionat, necessita consultar l'estat de l'agent B. Ho representarem així: A(B). La consulta no ha d'implicar cap modificació ni recàlcul d'estat en l'agent consultat. Això permet que no hi hagi restriccions en l'estructura de dependència i que puguin existir dependències creuades sense provocar cicles infinits. [1]

Tots els agents duen un nom associat que, per defecte, coincideix amb un prefixe i un número de sèrie únic entre tots els agents. Aquest nom es pot canviar per un que sigui més mnemotècnic per a l'usuari.

8.2 Agents Subordinadors

Els agents subordinadors són agents que, quan són accionats, accionen tot un seguit d'agents subordinats.

8.2.1 Agents Múltiples

L'agent múltiple acciona una i només una vegada cadascun dels agents subordinats cada cop que és accionat.

[1] Tot i així, és important preveure que l'ordre d'accionat entre agents interdependents podria implicar variacions en els resultats.

8.2.2 Agents Temporitzadors

Els agents temporitzadors són agents multiples que no sempre que reben un accionat el propaguen cap els subordinats. Estableixen dos períodes, un actiu i un altre inactiu. Els accionats només es propagen als subordinats durant el període actiu.

Els períodes es defineixen mitjançant tres paràmetres: El període mínim és el número d'accionats que dura el període com a mínim. Aquest mínim es pot augmentar de forma no determinística el resultat de sumar-li n vegades un número aleatori en l'interval [0,m] (els corxets indiquen que els extrems estan inclosos). n és el número de daus, i m és el valor màxim o magnitud del dau.

D'aquesta especificació es pot deduir algunes dades, pot ser, més intuitives per a l'usuari:

- El valor màxim que pot adoptar el període és el mínim més n*m
- Un sól dau equival a una distribució uniforme entre els límits
- A mesura que incrementem el nombre de daus, la distribució dels períodes s'aproxima a una distribució normal entorn al centre entre el valor màxim i mínim, amb un desviació típica cada vegada menor.

Per defecte, els paràmetres que introdueixen indeterminisme en els temporitzadors, com són els daus, estan ajustats de manera que el seu efecte sigui nul. Si no es toca res més que els períodes mínims, actuarà de forma determinista. De la mateixa manera, els períodes mínims dels cicles estan ajustats, per defecte, perquè sempre s'estigui en un cicle actiu. D'aquesta forma, si no es configura res, l'efecte d'un temporitzador és el d'un agent

múltiple ordinari.

Paràmetres per defecte i una execució:

- Cicle Actiu (mínim=1 daus=0 magnitud=0)
- Cicle Inactiu (mínim=0 daus=0 magnitud=0)
- Període Actual (actiu)
- Període Restant (1)

Paràmetres ilustratius:

- Cicle Actiu (mínim=0 daus=2 magnitud=3)
- Cicle Inactiu (mínim=4 daus=0 magnitud=0)
- Període Actual (actiu)
- Període Restant (1)

A continuació hi ha una execució dels paràmetres anteriors. Els guions representen accionats durant el període inactiu i les O's representen accionats durant el període actiu.

```
OO----OOOOO----OO----O----OOOO----O----OOO----OOOO----OOO----OOOO----O----OO----
OOO----OOOO--------OOO----O--------OOO----OOOO----O----OOOO----OOOO--------OOOO-
---OOO----OOO----OOO----OOOOO----O----OOOOOO----OOOO----O----OOOO----OOO----OO--
--OO----OOOO----OOO----OOO----OOOOO----OOO----OOOO----OOO----OO----OO----OO----O
OOOOO----OOO----OOOOO----OOOO----OO----OOOOO----OOOO----OOO----OOO----OOOOO----O
```

8.2.3 Agents Probabilitzadors

Els agents probabilitzadors són també agents múltiples que controlen si l'accionat es propaga cap els subordinats o no. Però, a diferència dels agents temporitzadors, ho fan mitjançant una llei probabilística. Si es dóna la probabilitat, s'accionen els subordinats, si no es dóna, no s'accionen.

La probabilitat es defineix amb el nombre de vegades que es donaria la probabilitat en un tamany de mostra. Per exemple, podem definir una probabilitat dient que es dóna 3 de cada 14 vegades. 14 és el tamany de mostra i 3 les vegades que es donaria en la mostra.

Els paràmetres estan ajustats per defecte a valors que fan del probabilitzador un agent múltiple ordinari.

Paràmetres ilustratius:

- Probabilitat (mostra=40 encerts=25)

A continuació hi ha una execució dels paràmetres anteriors. Els guions representen accionats en els quals no s'ha donat la probabilitat i les O's, accionats en els quals sí s'ha donat.

```
OOOO-O-O-O-OO----OOOO-OOOOOOOO-OOO-OOOOOOOOOOO-OOOOOOOOO---O-OOOOOOO-O--OOOO-OO-
OOO-OOO-O-OOOO-OOO-O-OOO-OOOOOOOOOOO-OOOOOOO-OOOOOO--OOO-OOO-OO-O-OOOOO--O---O-
OO--O---OOO-OOOOOOOOO-O-OOO-O-OOO-O--OOOOOOOO-OO-O-OO-OOO-OO-OOOO-OOOOOO-OOO-OOO
--OOOO-OOOO-O-OOO-OO-OO-O--OO-OOOO--O--O-O-OO--OO-O-O-OOO---O-O-OO------OO-OO-O-
OOO-OO-O-OO-OO-OOOOO----OO-OO-----O-OOOOO--O---OOOOOOOOOO-O-OO-O-OOOOO---OOO-OO-
```

8.2.4 Agents Iteradors

Els agents iteradors són agents múltiples que no limiten els accionats que arriben als seus subordinats, sinó que el que fan és multiplicar els accionats que hi arriben.

Més concretament, quan un agent iterador és accionat, els seus subordinats, són accionats un número de vegades que es calcula a partir d'un mínim i uns daus com els que feiem servir per als períodes dels temporitzadors.

Per defecte, la part indeterminística (els daus) no té cap efecte, i la part determinística (el mínim) està posada a un valor (1) que el fa equivalent a un agent múltiple ordinari.

Paràmetres ilustratius:

- Iteracions (mínim=2 daus=2 magnitud=4)

A continuació hi ha una execució dels paràmetres anteriors. Els parèntesis agrupen els accionaments dels subordinats que es fan sota un mateix accionament de l'iterador.

```
(OOOOOO)(OOOOOOOO)(OOOOOOO)(OOOOO)(OOO)(OOOOOOO)(OO)(OOOOOOO)(OOOOOOO)(OOOOOO)(O
OOOOOOO)(OOOOO)(OOOOOO)(OOO)(OOOOOO)(OOOO)(OOOOOO)(OOOOOO)(OOO)(OOOOOO)
```

Amb dos accions subordinades una execució quedaria com això:

```
(OEOEOEOEOEOEOEOEOEOE)(OEOEOEOEOEOEOEOEOE)(OEOEOEOEOEOEOE)(OEOEOEOE)(OEOEOEOEOEO
EOEOE)(OEOEOEOEOEOEOEOE)(OEOEOEOEOEOEOEOEOEOE)(OEOE)(OEOEOEOEOEOEOE)(OEOEOEOEOEO
EOE)(OEOEOEOEOEOE)(OEOEOEOEOEOE)(OEOEOEOEOE)(OEOEOEOEOEOEOE)(OEOEOEOEOEOE)(OEOEO
EOEOEOE)(OEOEOEOEOEOE)(OEOEOEOEOE)(OEOEOEOEOE)(OEOEOEOEOE)
```

on les E's representen l'execució de la segona acció subordinada.

8.3 Agents Posicionadors

Els agents posicionadors controlen una posició en la topologia del biòtop. No tènen subordinats, però, generalment hi ha agents que en depenen del seu valor i segons el tipus de posicionador per recalcular la seva posició fan servir altres agents dels quals depenen.

Posicionador Bàsic: No modifica la seva posició si és accionat. A menys que, per configuració, es fixi a una posició concreta, s'inicialitza amb una posició aleatòria vàlida dins de la topologia,

Posicionador Aleatori: Cada cop que és accionat pren una posició aleatòria vàlida dintre de la topologia.

Posicionador Zonal: Cada cop que és accionat pren una posició aleatòria dintre d'una zona. La zona es defineix per una posició central, determinada per un altre posicionador de qual depèn, i un radi, que no és més que el nombre de desplaçaments aleatoris que es fan a partir d'aquesta posició central per trobar la posició final. Les posicions tendeixen a adoptar una distribució normal en l'entorn de la posició central.

Posicionador Seqüencial: Cada cop que és accionat pren la posició del següent posicionador que hi ha en una seqüència de posicionadors. Els agents posicionadors de la seqüència són dependència del posicionador seqüencial.

Posicionador Direccional (Itinerari): Cada cop que és accionat pren la posició que en resulta d'aplicar-li un desplaçament a la posició anterior. El desplaçament el determina un agent direccionador que és dependència. Els agents direccionadors s'expliquen al següent apartat.

8.4 Agents Direccionadors

Els agents direccionadors controlen una direcció per calcular desplaçaments dintre de la topologia del biòtop. La seva utilitat principal radica en controlar la posició d'un posicionador de tipus direccional, però, no es descarten altres aplicacions futures.

Direccionador Bàsic: No modifica la seva direcció si és accionat.

Direccionador Aleatori: Cada cop que és accionat pren una direcció aleatòria.

Direccionador Seqüencial: Cada cop que és accionat pren la direcció del següent direccionador que hi ha en una seqüència de direccionadors. Els agents direccionadors de la seqüència són dependència del direccionador seqüencial.

Per obtindre aquests resultats ha calgut fer servir subordinadors perquè el posicionador s'accionés més que el direccionador. La topologia de l'exemple és toroidal.

8.5 Agents Actuadors

Els agents actuadors són els que finalment modifiquen el substrat. Els actuadors depenen d'un agent posicionador que els indica la cel·la que han de modificar.

La majoria dels agents que hem vist fins ara eren molt independents davant de les modificacions en la topologia i en la composició del substrat que es puguin fer més endavant. Les especialitzacions dels actuadors, en canvi, han de dependre per força del substrat i la seva composició, per que actuen sobre ell. Segueixen sent independents, però, de la topologia.

Aquí a sota, expliquem alguns actuadors vàlids pel substrat implementat en aquest treball.

8.5.1 Agents Nutridors

Aquests actuadors depositen nutrients al substrat. Un tipus de nutrient es codifica amb un enter de 32 bits sense signe. El tipus de nutrient que es depositarà s'especifica amb dos nombres enters de 32 bits sense signe. El primer enter indica el número del tipus bàsic, i el segon indica els bits del tipus bàsic que poden variar aleatòriament.

Per exemple, el parell $\begin{matrix}\text{element bàsic:} & \text{0x0000000000000000} \\ \text{variabilitat:} & \text{0xFFFFFFFF00000000}\end{matrix}$ genera elements que tenen la part baixa igual que l'element bàsic (a zero) i la part alta al atzar.

8.5.2 Agents Desnutridors

Els desnutridors són molt semblants als nutridors, però, en comptes de afegir nutrients, en treuen. Es pot treure selectivament cercant un element químic que s'apropi al que s'indica o es pot especificar una tolerància per a certs bits.

8.6 Arxius de configuració d'agents

L'arxiu de configuració d'agents és un arxiu de text que conté l'estat i l'estructura dels agents d'un biosistema, que es pot extreure en un moment donat i restaurar-ho posteriorment.

Els agents a un biosistema, com s'ha dit abans, formen una estructura d'arbre segons les seves relacions de subordinació. Cada arxiu de configuració conté un arbre d'agents

subordinats partint d'un agent arrel.

Bàsicament, primer es posen els noms dels agents i el seu tipus i, més abaix, es configuren els paràmetres de cadascun. El primer agent del que s'indica el tipus funcionaria com a arrel. L'apèndix A.3 especifica el contingut d'aquests arxius. Un exemple d'arxiu de configuració d'agents seria:

```
* Agent_0000 Agent/Multiple
* Agent_0002 Agent/Posicionador/Direccional
* Agent_0005 Agent/Multiple/Iterador
* Agent_0004 Agent/Actuador/Nutridor
* Agent_0003 Agent/Posicionador/Zonal
* Agent_0006 Agent/Multiple/Temporitzador
* Agent_0001 Agent/Direccionador/Aleatori

+ Agent_0002
- Posicio 1271
- Radi 1
- Direccionador Agent_0001

+ Agent_0004
- Posicionador Agent_0003
- Composicio 31 0

+ Agent_0003
- Posicio 8
- Radi 1
- Posicionador Agent_0002

+ Agent_0005
- Accio Agent_0004
- Accio Agent_0003
- Iteracions 20 0 0

+ Agent_0001
- Direccio 2192479406
```

```
+ Agent_0006
- Accio Agent_0001
- CicleActiu 1 0 1
- CicleInactiu 5 0 1
- CicleActual 4 Inactiu

+ Agent_0000
- Accio Agent_0002
- Accio Agent_0005
- Accio Agent_0006
```

Capítol 9

Els organismes

9.1 Visió externa dels organismes

Un organisme ofereix a la resta del sistema un conjunt de funcions.

- Permetre l'accés a un conjunt de registres que formen el fenotip de l'organisme. El biosistema pot modificar-los i consultar-los segons ho requereixin les instruccions d'aquest o d'altres organismes.
- Expedir codis d'instrucció que indiquin al biosistema les accions que pensa realitzar segons el contingut dels registres de fenotip.
- L'organisme ha de proveir al biosistema d'un protocol d'accés al seu sistema metabòlic i a d'altres funcions vitals. Aquestes funcions vitals, no poden accedir directament al fenotip.
- També han d'implementar mecanismes per tal de crear nous organismes. De forma aleatòria o a partir de la forma d'altres ja existents.

9.2 Components bàsics

Els organismes es composen de quatre parts principals:

- *Sistema d'herència (cariotip):* Conté la informació genètica que es passa de pares a fills.
- *Sistema de control (genotip):* Determina les accions que realitza l'organisme.
- *Sistema de memòria (fenotip):* Fa d'interfície entre l'entorn i l'organisme.
- *Sistema metabòlic (pap+estat energètic):* Conjunt de recursos que permeten a l'organisme obtenir i fer servir l'energia.

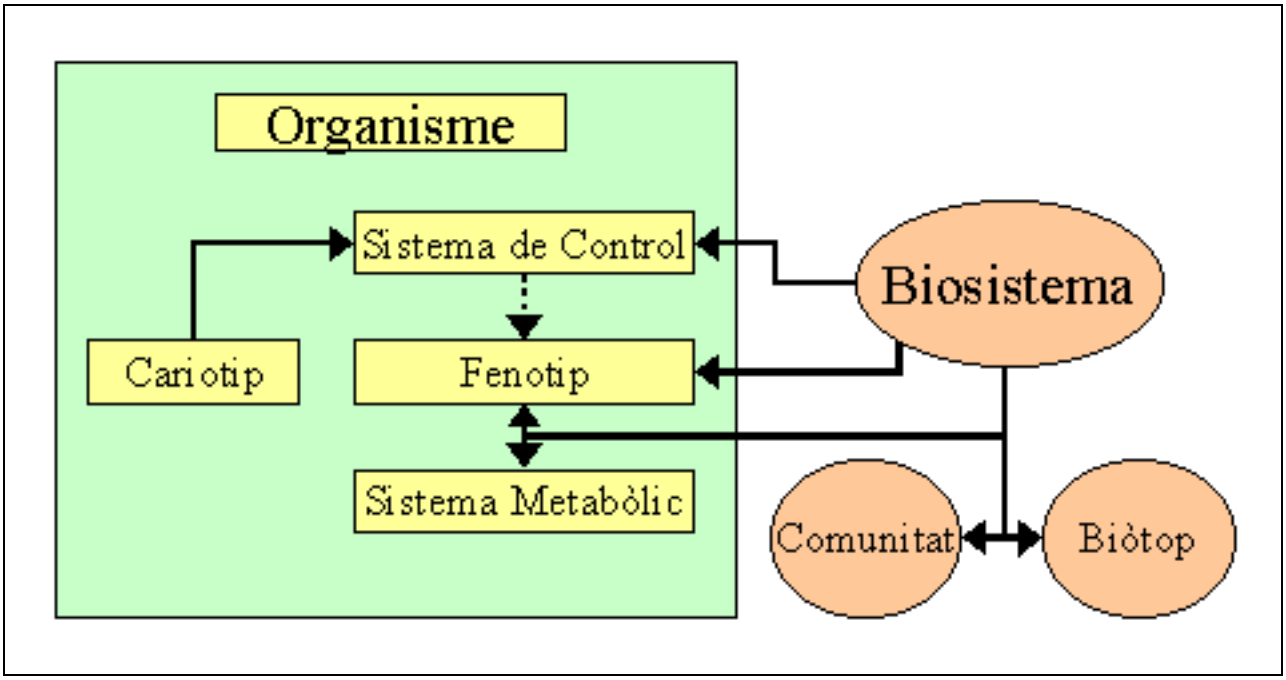

Figura 9.1: Model d'execució d'instruccións

Les interrelacions entre els diferents mòduls es descriuen a la figura 9.1. A grans trets, el funcionament és el següent:

1. Quan un organisme neix, es genera el sistema de control a partir del cariotip. El fenotip i el sistema metabòlic s'inicialitzen independentment del cariotip.
2. Un cop introduït l'organisme dins de la comunitat, el biosistema pot demanar-li instruccions al sistema de control.
3. El sistema de control consulta el fenotip i decideix quina instrucció cal executar.
4. El sistema de control retorna al biosistema la instrucció en qüestió.
5. El biosistema completa la instrucció amb paràmetres que es troben dins del fenotip.
6. El biosistema executa la instrucció. L'execució pot comportar consultes i/o modificacions sobre l'estat de:
    - La comunitat d'organismes i el biòtop
    - El fenotip i el sistema metabòlic del propi organisme.

Sense trencar les interfícies descrites en aquest esquema general podem alterar disseny dels organismes en diversos aspectes:

- Es pot substituir el sistema control de l'organisme sense cap implicació en la resta del sistema.
- Es pot substituir de forma completa el sistema metabòlic tot adaptant les accions que fa el biosistema sobre ell.

- Es poden fer variacions sobre el sistema d'herència tot adaptant les rutines de traducció del control.

9.3 El fenotip

9.3.1 Concepte genèric

S'anomenat *fenotip* en aquest sistema a un conjunt de 16 registres de 32 bits que té cada organisme. És el mecanisme d'intercanvi d'informació entre el sistema de control i el biosistema.

Les instruccions expedides al biosistema pel sistema de control, depenen del contingut del fenotip i, com a resultat de l'execució d'instruccions, el fenotip es modifica amb la informació a la qual el biosistema pot i deixa arribar. D'aquesta manera, el sistema de control es realimenta contínuament.

En resum, el fenotip és el mecanisme principal d'interacció que tenen els organismes.

El que s'aconsegueix amb el fenotip és aillar el sistema de control de la resta d'elements donant una interfície uniforme sigui quina sigui la representació del sistema metabòlic o del biosistema en global.

9.3.2 Ús del fenotip en la present implementació

En la present implementació les úniques instruccions que modifiquen el fenotip són les del propi organisme. Les instruccions generades pel sistema de control d'un altre organisme

no el poden modificar tot i que el poden consultar.

La informació amb la que s'omple el fenotip pot vindre del contingut del medi, de l'estat del sistema metabòlic, del contingut antic del mateix fenotip o del contingut dels fenotips d'altres organismes. Aquesta darrera font d'informació, és un dels dos mitjans que tènen els organismes per comunicar-se, juntament amb la detecció de mol·lècules excretades.

En nèixer un nou organisme, el fenotip s'inicialitza a valors que traduïts a desplaçaments siguin desplaçaments nuls. El pare té el dret excepcional d'inicialitzar un dels registres a un valor donat quan el fill neix.

9.4 El sistema de control

9.4.1 Trets generals

El sistema de control d'un organisme consulta la informació dels registres del fenotip, i, en funció del que troba, en retorna un codi d'instrucció. Aquesta és la seva funció bàsica. En realitat, el sistema de control no sap el significat del codi d'instrucció que ha generat.

Per a la resta del sistema, la implementació interna del sistema de control és indiferent. Podria ser una xarxa neuronal, un programa estructural, un usuari humà ... En aquest primer prototip, s'ha implementat un sistema de control basat en els mecanismes de regulació de l'expressió gènica que es donen a la natura.

9.4.2 Regulació de l'expressió gènica

A la natura, perque un gen s'expressi necessita que l'enzima que transcriu l'ADN tingui afinitat amb la part previa al gen en el cromosoma (la zona operadora). Sovint la presència de sustàncies o d'enzimes generades per altres gens facilita o dificulta aquesta afinitat. D'aquesta forma es regula la producció de les proteïnes que en deriven de cada gen, variant el comportament gènic de l'organisme davant de situacions diferents.

El sistema de control implementat emula aquests mecanismes de regulació.

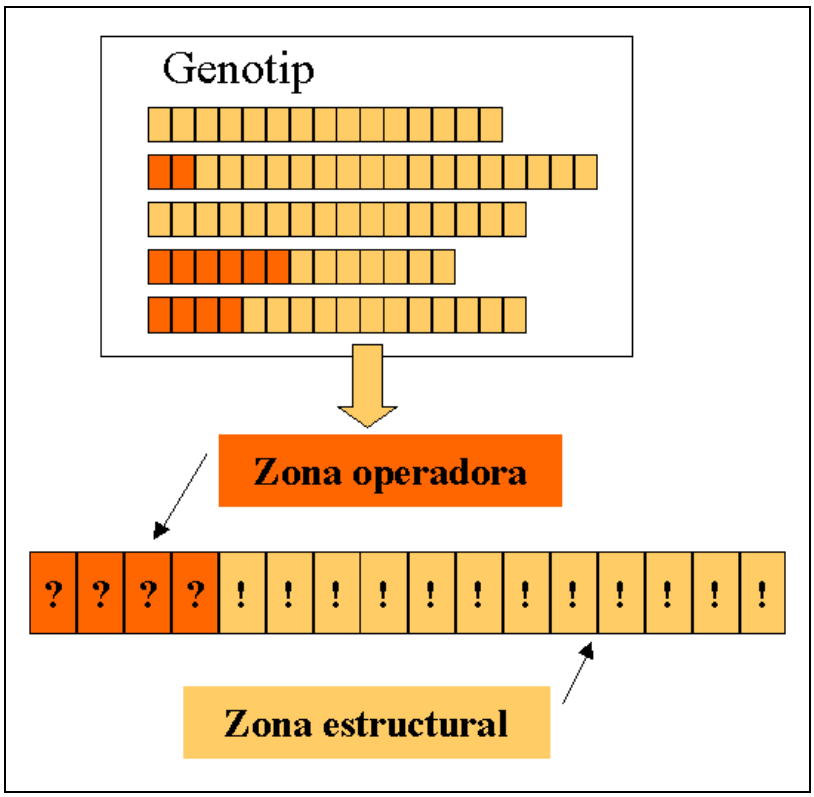

Figura 9.2: Genotip, gens, zona estructural i zona operadora

El sistema de control està format per un conjunt de gens (el **genotip**). Cada gen té dues zones: una zona estructural i una zona operadora. La zona estructural conté una seqüència

de codis d'instruccions i la zona operadora codifica un conjunt de condicions que s'han de donar perqué la seqüencia de la zona estructural es pugui executar.

9.4.3 Zones operadores

Cada codó (part elemental de traducció) de la zona operadora codifica una condició. El gen s'expressa si es compleixen una a una les condicions de cada codó de la zona operadora.

L'avaluació de les condicions del operador depenen del contingut dels registres de fenotip de forma que, en cada moment el fenotip pot ser favorable o no a que un gen s'expressi.

La condició implementada és:

- **Similitud entre registres:** La condició es compleix si dos registres tenen un contingut similar tot tenint en compte un patró de tolerància indicat per un tercer registre. Cada codó codifica una O lògica entre dos condicions d'aquest tipus.

Es podrien implementar altre tipus d'operacions tot fent servir els bits que en sobren com a discriminador del tipus de condició.

9.4.4 Transcripció

La trascripció del cromosoma en gens constarà de diverses parts:

1. Indentificació de la zona promotora (indica l'inici d'un locus)
2. Indentificació de la zona terminadora corresponent (indica el final d'un locus)

3. Identificació (justament després de la zona promotora) i interpretació de la zona operadora (indicarà quan cal executar el gen traduït).

4. Eliminació d'introns (sequencies de codons no significatius) entre promotora i terminadora (procés de maduració)

5. Traducció de la zona estructural (la que porta les instruccions que s'executaran amb el gen)

Per raons d'optimització, la transcripció, maduració i traducció es fa només una vegada durant la vida de l'organisme, encara que, a la natura, la transcripció de l'ADN es fa contínuament. Això no té implicacions massa importants, donat que ens guardem amb el gen el significat de la seva zona operadora que ens serveix per simular el comportament temporal de la transcripció.

9.5 Sistema d'herència

9.5.1 El cariotip i els cromosomes

El sistema d'herència és el cariotip. El cariotip és un conjunt de cromosomes de tamany variable. Cada cromosoma conté una seqüència de bases (bits).

El cariotip és informació en brut organitzada en paquets (els cromosomes). Per extreure'n el significat, per construir el sistema de control associat, cal interpretar aquestes dades. Això és el procés de traducció.

La forma en que s'ha implementat la traducció del cariotip, segueix molt a prop el model

biològic.

Si els cromosomes que formen el cariotip són unitats estructurals del material genètic, els gens que formen el genotip en són les seves unitats funcionals.

Aquest diferenciació es justifica perquè:

1. Permet un model multicromosómic, amb un nombre i longitud de cromosomes variable que possibilita solucions creatives [Kar97].

2. Permet implementar, sobre els cromosomes, operadors de mutació i creuament planers.

3. La traducció de cariotip a genotip, ens permet detectar promotors i terminadors [May97], proporcionar operadors, eliminar introns [TS97]... Facilita d'aquesta forma la implementació d'aquests fenòmens.

4. És més semblant al comportament biològic.

9.5.2 Mutacions

El cariotip és el responsable de les mutacions que passaran a la descendència.

Tot i que la probabilitat de mutació ha de dependre, en gran mesura, dels agents mutàgens del medi, els organismes han de poder controlar genèticament la probabilitat de mutació per adaptar-la a la seva situació. En posar en mans de l'evolució la probabilitat de mutar estem confiant en que l'coevolució castigarà els organismes que no mutin en front dels que mutin doncs els primers no podran millorar.

El control sobre la probabilitat de mutació d'un organisme es fa amb una clau que té el propi organisme. Aquesta clau es pot adaptar, en major o menor mesura, al llarg del procés evolutiu, a una d'equivalent que hi ha a cada posició del substrat.

Un cop es dóna, a un organisme, la probabilitat de mutar, cal decidir com es fa la mutació, es a dir, amb quin operador de mutació es fa. Cada organisme té codificada genèticament una ponderació per a cada operador de mutació.

$$Probabilitat(operador_i) = \frac{ponderacio_i}{\sum_j ponderacio_j} \tag{9.1}$$

Els operadors de mutació són objectes que podem aplicar a un cariotip per aplicar la mutació. Els operadors de mutació implementats es basen en el funcionalment de les mutacions naturals i es divideixen en tres categories segons el seu abast:

- *Mutació Cariotípica:* Modifica el cariotip.
  - *Mutació per fusió:* Fusiona dos cromosomes.
  - *Mutació per escisió:* Parteix en dos un cromosoma.
  - *Euploidia positiva:* Duplicació total del cariotip.
  - *Aneuploidia positiva:* Duplicació d'un cromosoma.
  - *Aneuploidia negativa:* Eliminació d'un cromosoma.
- *Mutacio cromosómica o estructural:* Modifica l'estructura d'un cromosoma.
  - *Mutació per delecció:* Elimina un fragment del cromosoma.
  - *Mutació per desplaçament:* Desplaça un fragment de lloc dins del cromosoma.

  - *Mutació per inserció aleatòria:* Insereix al cromosoma un fragment aleatori.
  - *Mutació per inserció replicada:* Insereix al cromosoma un fragment replicat del mateix cromosoma.

- *Mutació gènica o puntual:* Modifica el contingut d'una base o conjunt de bases.
  - *Mutació puntual dràstica:* Canvia una paraula (codó) per un altre.
  - *Mutació puntual binària:* Inverteix una base (bit) per un altre.
  - *Mutació puntual binària en distribució gaussiana:* Inverteix aproximadament 4 bits d'un codó (el nombre de bases modificades següeix una distribució normal).

9.6 Model metabòlic dels organismes

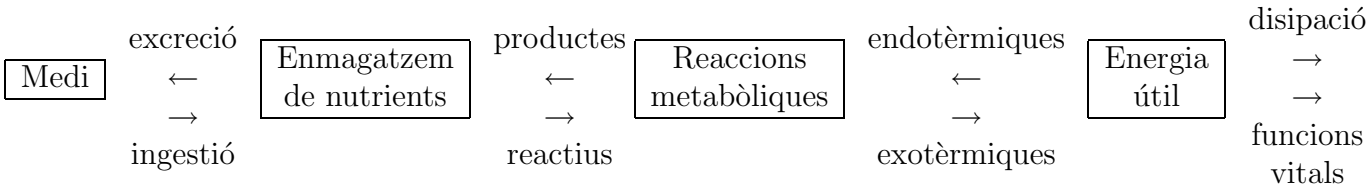

Figura 9.3: Model metabòlic dels organismes a Bioscena

Dintre de l'organisme hi ha dos formes de tenir l'energia:

- En forma d'energia útil.
- En forma de nutrients.

No podem passar d'una a l'altre, si no és mitjançant un procés metabòlic. Per un costat, l'energia útil és l'única que pot fer-se servir a la majoria de processos vitals. Per un altre,

aquesta energia útil té una caducitat de forma que, quan passa un cert temps de la seva obtenció, es disipa. La figura 9.3 representa els cicles energètics dins d'un organisme.

9.6.1 Fluxe de nutrients

Com s'ha explicat a l'apartat 7, els nutrients tenen un identificador o patró qualitatiu que indica quin tipus de nutrient és. Per seleccionar un nutrient dins d'un grup, només cal oferir una clau o patró de cerca i una tolerància respecte a aquest patró. Cal recordar que la funció de compatibilitat entre claus no és determinista (apartat 14.8.1) i l'execució d'una mateixa cerca pot donar resultats diferents.

Un organisme pot introduir un nutrient en el seu cos des del medi per ingestió o des d'un altre organisme en atacar-lo. Anàlogament, els nutrients es poden extreure del cos cap al medi per excreció i cap un altre organisme en rebre un atac.

El nombre de nutrients que hi caben al pap d'un organisme pot estar limitat per configuració. De fet es recomana, donat que alguns organismes tendeixen a acomular-ne i se'n deriva un alt consum de memòria. Al igual que succeeix al substrat, quan es sobrepassa el límit, els nutrients més antics s'eliminen.

Per fer una reacció metabòlica, s'extreuen els nutrients reactius del pap de l'organisme. D'aquesta reacció s'obte un balanç d'energia i uns productes que son introduïts al pap novament.

9.6.2 Fluxe d'energia útil

Segons els reactius, les reaccions metabòliques acaben sent exògenes o endògenes, és a dir, proudeixen energia o en consumeixen.

El més normal segons el model és obtindre energia útil a partir de les reaccions metabòliques exògenes. Però, és possible configurar el biosistema per que certes accions, com el simple fet d'ingerir un nutrient, en generin energia útil. No dependre del metabolisme per obtindre energia permet implementar sistemes més simples, que facilitin la feina als organismes per sobreviure.

L'energia útil es consumeix per tres motius:

- Les contribucions energètiques a reaccions endògenes
- El cost de les accions realitzades
- La disipació

Es pot configurar el cost associat a cada funció vital. Equilibrar els costos i guanys energètics de les diferents operacions és crucial per obtindre comportaments complexos en els organismes. Cal forçar-los a que desenvolupin estratègies complexes i que tinguin flexibilitat de maniobra per evolucionar estrategies no òptimes.

La disipació de l'energia s'implementa amb un seguit de contenidors cadascun dels quals té una caducitat. L'energia obtinguda es fica en el contenidor més nou, mentres que l'energia que perdem l'extreiem dels contenidors més vells. Quan el contenidor més vell caduca, l'energia que hi conté es perd (es disipa) i s'afegeix un contenidor nou buit.

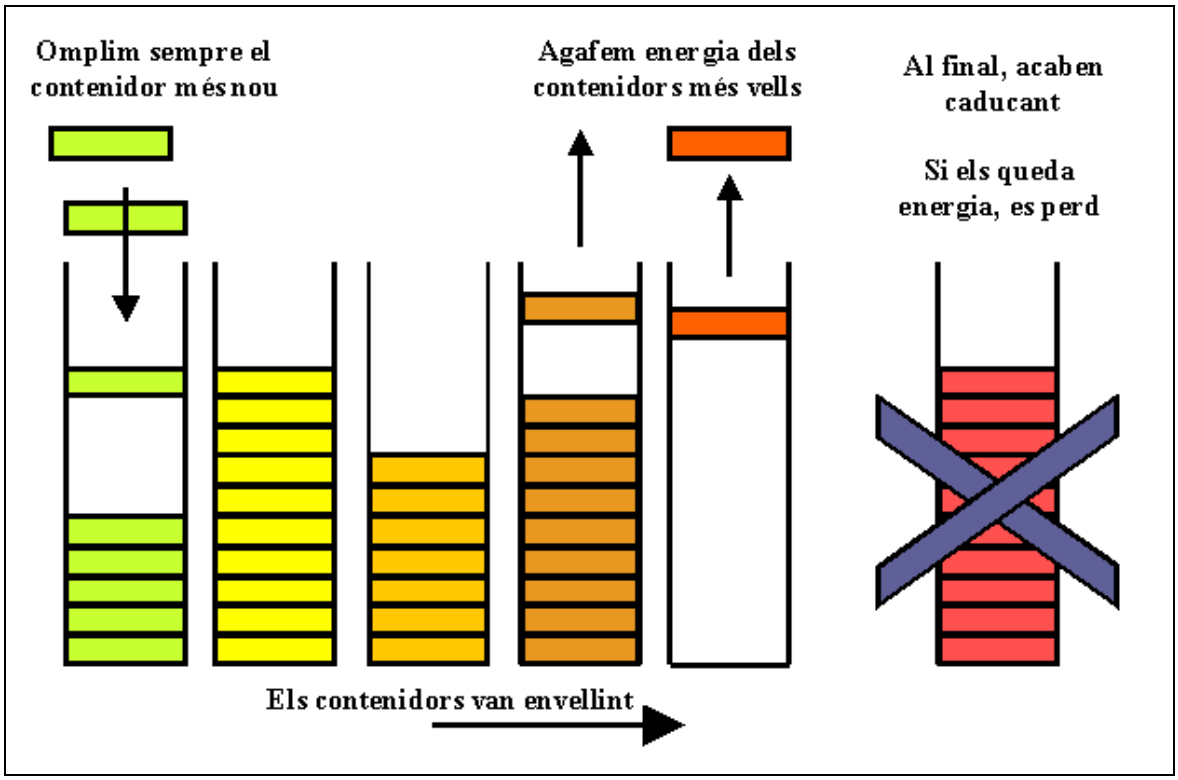

Figura 9.4: Disipació de l'energia obtinguda per l'organisme

Capítol 10

Mecanismes d'especiació i anàlisi

10.1 Els taxonomistes

Per que l'usuari pugui extreure una informació útil, cal que poguem agrupar els organismes en espècies. És clar que a la nostra comunitat, al igual que a la natura, l'especiació és un fenòmen que cal que emergeixi, L'espècie, no és quelcom intrínsec a tot organisme; és a dir, que el concepte d'espècie no estarà implementat al codi genètic o als mecanismes de funcionament comuns dels organismes sinó que caldrà observar-ho en el seu comportament.

Els taxonomistes són objectes que agrupen organismes per afinitat evolutiva segons els conceptes abans mencionats.

Com que, abans de definir objectius, em plantejava implementar organismes amb reproducció sexual, vaig construir un taxonomista que soportava tota la complexitat que comporten els creuaments i que he mencionat abans.

Com finalment, els intercanvis sexuals es van excloure dels objectius del projecte, les rela-

cions evolutives es limiten a un arbre i vaig implementar un taxonomista molt més senzill, que simplement discrimina organismes amb diferent cromosoma quan es produeix una mutació, però, que compleix el mateix protocol que el més elaborat implementat anteriorment de forma que es podrien intercanviar.

El taxonomista simple és el que he fet servir a les proves donat que era molt més ràpid i la informació que dona ja és suficient per la complexitat dels sistemes generats.

Tot i així, documento el taxonomista per organismes sexuals de cara a ilustrar millor el protocol i donat que serà útil en futures ampliacions del sistema si s'afegeixen creuaments.

10.2 Qué es vol solucionar

El concepte clàssic d'espècie considera que dos organismes són de la mateixa espècie si són capaços de donar descencència fèrtil.

A la biologia moderna es considera que la diferenciació de les espècies no està tant en la capacitat sinó en el fet mateix de reproduir-se. En això pot influir:

- Compatibilitat genètica
- Compatibilitat d'acoplament
- Compatibilitat geogràfica
- Altres

També, cal adonar-se de que totes aquestes consideracions es refereixen als organismes

que es reprodueixen sexualment i es creuen. Còm es poden identificar les espècies dels organismes que es reprodueixen asexualment (com és el cas implementat)? En quin punt es considera que dos descendents d'un mateix organisme sòn d'una espècie diferent?

A més, degut a la seva qualitat emergent, l'especiació no estarà sempre ben definida. També pot ser que la selecció a l'hora de reproduir-se tingui en compte altres mecanismes que no pas l'especiació.

Tota aquesta conjuntura ha obligat a deixar de banda el concepte d'espècie cap al concepte, una mica més relaxat, de grup reproductiu que, tot i la relaxació, segueix proporcionant informació útil sobre les diferents poblacions del biosistema. Considerem que un grup reproductiu és un conjunt d'organismes que es creuen entre sí o provenen d'ancestres comuns o ancestres que s'han creuat entre sí dintre d'un cert període de temps o d'un cert nombre de generacions.

10.2.1 Taxonomista d'organismes sexuals

La política que determina els grups reproductius es basa en un marcatge històric.

Cada individu s'associa amb un taxó que no és més que una seqüència de marques amb diferent antiguitat. Les marques es traspasen idèntiques a la descendència via mitosi.

Cada cert temps, es fa una discriminació que consisteix en fondre les dos marques més antigues de cada taxó en una sola i afegir una marca nova que diferenciarà els individus que fins llavors compartien les mateixes marques i que pendrà importància a mida que adquireixi antiguitat.

Cóm es produirà aquesta discriminació? Imaginem que existeixen individus amb els taxons següents:

A A A A A A A A A A A A A B A A A A A B B A A A A B B A A A B A A A A A	⇒	**A** A A A A **A** A A A A **B** A A A A **B** B A A A **B** B A A A **C** A A A A	⇒	A A A A A **A** A A A A A **B** B A A A A **A** B B A A A **A** B B A A A **B** C A A A A **A**
Taxons originals dels organismes abans de la discretització de la població		Fusió dels dos taxons més antics		Discriminació dels individus amb les mateixes marques amb una nova:

D'altra banda, hem de considerar el que passa quan es creuen dos individus que pertanyen a diferents taxons.

Quan dos individus es creuen, s'asimilen les marques des de la marca més antiga fins a la primera que els diferencia als dos (marca discriminant). Es a dir, a tots els taxons cal revisar les marques. Pot ser es veu més clar amb un exemple. Considerem el següent conjunt de taxons.

A A A A A A
A A A A A B
A A A A B B
A A A B A B
A A A B B A
A A A B B B
A A B A A A

Si es creuen AAAABB i AAABBA, cal considerar equivalemts les subseqüencies de marques AAAA i AAAB equivalents. Tots els taxons que comencin per AAAB els canviem per AAAA sense oblidar-nos de modificar la següent marca més jove que la discriminant amb

l'objectiu de que els taxons asimilats mantinguin el sentit.

A A A A A A --→ A A A A A A
A A A A A B --→ A A A A A B
A A A A B B --→ A A A A B B
A A A B A B ⇒ A A A *A* *C* B
A A A B B A ⇒ A A A *A* *D* A
A A A B B B ⇒ A A A *A* *D* B
A A B A A A --→ A A B A A A

De cara a la implementació, he considerat separar la major part del procés associat a la determinació de grups reproductius en un objecte independent anomenat taxonomista. Aquest objecte s'encarrega de mantenir els taxons al dia mitjançant una interfície estreta que manté amb el processador.

A dins de la comunitat es manté una informació mínima: l'identificador del taxó al que pertany cada individu. El tràfec d'informació a l'exterior del objecte taxonomista es basa exclusivament en el pas d'aquests identificadors. La interfície oferida permet al processador:

- Incrementar o decrementar la poblacio assignada a un taxó
- Creuar un parell de taxons
- Generar un nou taxó (Per individus generats espontàniament)
- Envellir les marques i discriminar la població que comparteix el mateix taxó
- Determinar el grau de parentesc entre dos individus

Quan hem de discriminar o quan fussionem dos taxons, necessitem que la Comunitat i el Taxonomista cooperin.

Quan cal discriminar la població, la Comunitat demana al Taxonomista, per a cada individu, un nou taxó, basant-se en el taxó antic i el número de individus que en queden sense discriminar d'aquest.

El nou taxo duu les marques X.X.X. ... .X.N on N és el número de queden sense discriminar.

Quan, fruit d'un creuament, es fusionen dos taxons, un dels dos taxons és assimilat per l'altre i, en conseqüència, els individus associats al taxò assimilat, cal associar-los al taxó assimilador.

Per dins, el taxonomista està compost per una llista indexada de taxons (taxonari). Els números d'índex es referencien des de cada individu pertanyent a la Comunitat.

Una alternativa al taxonari hagués sigut una implementació en arbre en comptes de la llista indexada. En cada node hi hauria una marca i a les fulles els identificadors de cada taxó. La implementació en arbre simplifica molt la lògica dels algorismes de discriminació i creuament però complica altres operacions internes que amb la llista indexada són trivials.

La llista indexada permet un accés directe als taxons i, a més, els manté ordenats de tal forma que les cerques de grups de parentesc tenen un cost temporal mínim.

Capítol 11

Coordinació del biosistema

L'objecte `CBiosistema` conté i coordina tots els elements que formen part del nucli del simulador.

El cicle intern del biosistema és una funció que un controlador pot executar de forma iterativa. A continuació es detallen les accions que el biosistema pot fer durant una iteració.

11.1 Manteniment de la població mínima i la variabilitat genètica

El biosistema té la capacitat de generar organismes aleatoris i inocular-los al sistema.

A cada cicle, el biosistema prova d'inocular organismes aleatoris en tres casos:

- Quan es dona una probabilitat configurada
- Quan la població no arriba a un cert mínim configurat

- Quan no hi ha població

El procediment per fer-ho no sempre és exitós donat que es poden donar condicions per que la inoculació no sigui correcta, per exemple, si la posició del biòtop escollida és ocupada, com passaria segur si no hi hagués cap posició del biosistema lliure.

11.2 Control del quàntum i canvi de context

Com que es vol simular un sistema paral·lel en una màquina seqüencial haurem d'implementar tècniques de temps compartit. El quàntum és el nombre d'instruccions d'un organisme que s'executen seguides abans de canviar a un altre organisme.

Sovint les tècniques de temps compartit introdueixen artificis en els sistemes de simulació i, en concret, de vida artificial. Amb l'objectiu de dispersar el seu efecte hem introduït les següents tècniques:

- No intercanviar els organismes de forma ordenada, sino aleatòriament.
- Fer servir un quantum despreciable, (si no pot ser 1), respecte les instruccions que necessita l'organisme per reproduir-se.
- Introduir cert indeterminisme en la determinació del quàntum.

L'encarregat de fer el "canvi de context" és un mètode del biosistema que té com a postcondició surtir sempre amb un organisme com actual. Fins i tot, si la comunitat és buida, en genera un organismes aleatoris fins que ho deixi de ser.

Si, per un costat, es bo fer petit el quàntum per dispersar els artificis, fer el quàntum petit també carrega més el sistema doncs cal fer més canvis de context.

També, de cara a visualitzar les instruccions pas a pas, a vegades també va bé fer el quàntum un xic gran per veure més instruccions juntes d'un mateix organisme i entendre millor el seu sentit.

11.3 Defuncions

Si l'organisme actual no té energia, s'executa el procediment de defunció. Aquest fa tres accions:

- Buidar la posició del biòtop on es trobi l'organisme.
- Indicar al taxonomista que un organisme de determinat taxó ha mort.
- Extreure l'organisme de la comunitat.

Després d'una defunció, evidentment, cal canviar a un nou organisme amb la funció de canvi de context i tornar a comprovar si aquest nou organisme també està sense energia.

11.4 Expedició d'instruccions

Un cop tenim un organisme amb suficient quantum i prou energia per continuar, se li demana una instrucció i s'executa. L'apartat 12 detalla una mica més com s'executen les instruccions.

11.5 Temps simulat

De cara a mesurar el temps transcorregut en el sistema, la idea més simple és comptar el cicles del biosistema, això és, el nombre d'instruccions executades.

Però, es vol trobar una mesura del temps que, d'alguna forma, sigui regular des del punt de vista dels organismes. Regular des del punt de vista de l'organisme vol dir, que un organisme, en n unitats de temps un organisme executi sempre, aproximadament, i instruccions.

El problema que porta considerar els cicles del biosistema com a unitat de temps és que la població varia durant la simulació. Com que la població varia al llarg del temps, donat un nombre n d'instruccions executades entre tots els organismes, no es manté una proporció p més o menys constant d'instruccions d'un mateix organisme.

Així doncs, si comptem el nombre de cicles, des del punt de vista dels organismes es veurà que, quan hi ha més població, el temps passa més poc a poc i, quan hi ha menys població, el temps passa mès ràpid.

El problema ve del fet que, en la realitat, un nombre variable d'organismes executarien instruccions en paral·lel, i, en aquesta simulació tots els temps es seqüencien com indica el gràfic següent. En ser una població variable, els intervals oscilen.

A la figura 9.4, anomenem *temps real o temps seqüencial* al temps que es pot mesurar en cicles del biosistema i que és el que nosaltres percebem per la pantalla, i *temps simulat o temps paral·lel* al temps que veuen els organismes que és el que veurien en la realitat.

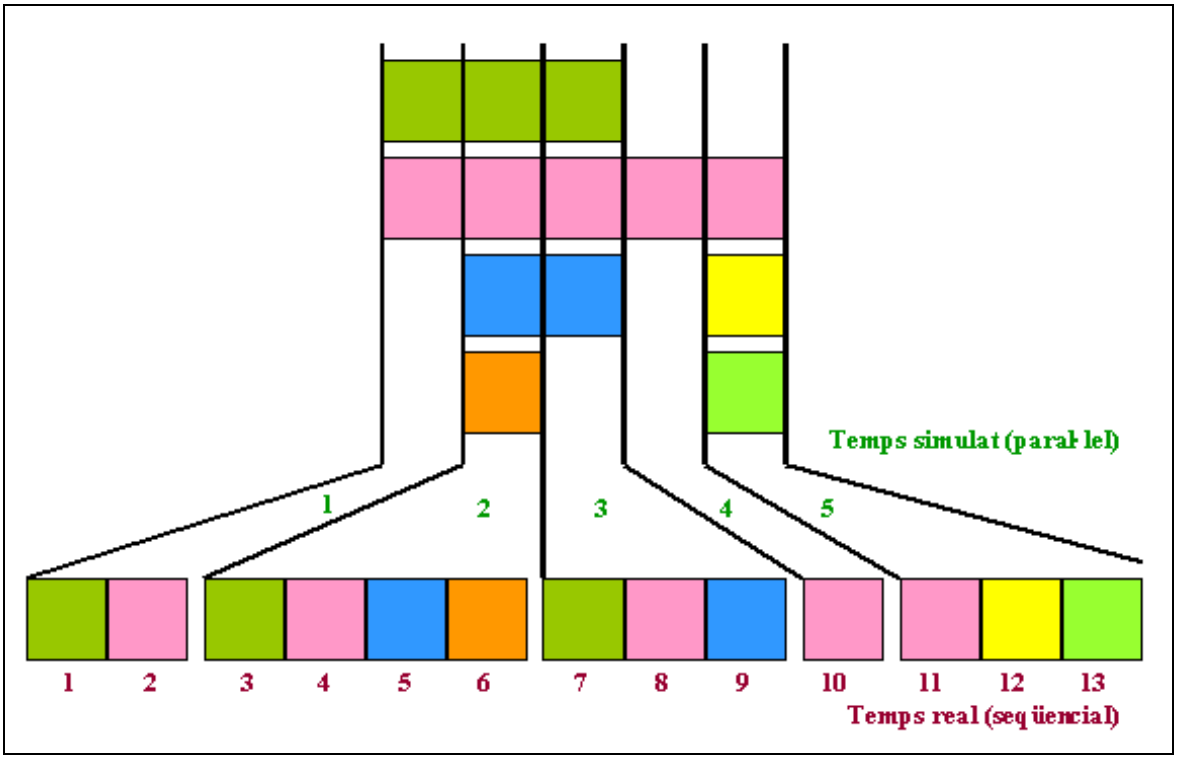

Figura 11.1: Temps seqüencial i temps paral·lel

Al esquema seqüencial de la figura 9.4 estem imaginant que el model de compartició de temps entre els organismes seqüencia en ordre els organismes de la comunitat. En aquest cas, es podria mesurar fàcilment el temps paral·lel considerant com unitat de temps el que triga el sistema en atendre a cadascun dels organismes.

A Bioscena no es pot fer exactament aquesta solució, donat que s'ha escollit seqüenciar els organismes de forma aleatòria per disipar els artificis de la seqüencialitat. Però, podem fer una aproximació bastant equivalent que consisteix en comptar la població en un determinat instant i esperar tants cicles del biosistema com aquest número per incrementar el temps.

Cal tenir en compte que aquesta mesura no és gaire perfecte durant períodes de fluctuacions en la població donat que la població la estem mirant, només en un instant.

11.6 Accionament dels agents ambientals

El biosistema acciona l'arbre d'agents ambientals que actuen sobre ell cada unitat de temps simulat. D'aquesta forma els organismes i els agents ambientals parlen una mateixa llengua en qüestió de temps.

Com a mitja es dóna que, per cada vegada que s'accionen els agents, cada organisme ha executat una instrucció.

Capítol 12

Conjunt d'instruccions

El biosistema pot configurar quins opcodes es relacionen amb quines operacions mitjançant un arxiu de configuració especial. Mitjançant aquesta relació, el biosistema pot saber, a partir dels números d'instruccions que ofereix l'organisme, quina és la instrucció que cal executar.

Aquest apartat detalla com s'executen les diferents instruccions que pot executar un biosistema. Els paràmetres de les instruccions, són nibbles de 4 bits que generalment indexen un dels 16 registres de 32 bits que formen fenotip de l'organisme actual, d'on s'extreu o a on s'escriu el valor.

12.1 Instruccions fenotípiques

El següent grup d'instruccions, són instruccions en les que només intervé el fenotip. Ni el sistema metabòlic ni els altres elements del biosistema. Són operacions que comencen i acaben al fenotip.

- `And dest op1 op2`: Fica a `dest` la and bit a bit de `op1` i `op2`
- `Or dest op1 op2`: Fica a `dest` la or bit a bit de `op1` i `op2`
- `Xor dest op1 op2`: Fica a `dest` la xor bit a bit de `op1` i `op2`
- `Not dest op1`: Fica a `dest` la negació bit a bit de `op1`
- `Oposa dest op1`: Fica a `dest` el desplaçament en sentit contrari al desplaçament expressat per `op1` (veure 7.1)
- `Carrega dest`: Omple el registre amb el valor de la següent instrucció
- `Random dest`: Omple el registre amb un valor al atzar
- `ShiftL dest op1 num`: dest=op1<<num (num és directament el nibble, no pas el valor del registre indexat pel nibble)
- `ShiftR dest op1 num`: dest=op1>>num (num és directament el nibble, no pas el valor del registre indexat pel nibble)

12.2 Instruccions sensorials

Són les instruccions que modifiquen el fenotip segons algun element del biosistema:

- `SensorQ`: Sensor químic (nutrients al medi)
- `SensorP`: Sensor de presència (altres organismes)
- `SensorI`: Sensor intern (nutrients al pap i estat energètic)

Tant el sensor químic com el de presència, cerquen una posició en una zona del biòtop que acompleixi una condició. El primer cerca per un nutrient que tingui una clau compatible amb el patró i la tolerància indicades com a paràmetres. El segon cerca un organisme que, al registre indicat, tingui un contingut compatible amb el patró i la tolerància indicades.

Totes dos indiquen una posició relativa que marca el centre de la zona de cerca i un radi que indica el número màxim de salts als que es pot trobar l'objectiu.

Es comprova un nombre configurable de vegades una posició aleatòria dintre d'aquesta zona. Si la posició compleix la condició, als dos registres de destí es posen el valor trobat i el desplaçament relatiu que em dirigeix a aquesta posició.

El sensor intern simplement posa, a un registre destí, la clau del nutrient dins del pap que és compatible amb el patró i la tolerància donades, si n'hi ha cap, i, a un altre, l'energia total actual.

12.3 Instruccions motores

Les instruccions motores són les que modifiquen altres coses que no només el fenotip. En resum són:

- `Ingestio`: Incorpora al pap nutrients lliures al medi
- `Excrecio`: Allibera al medi nutrients del pap
- `Moviment`: Mou l'organisme a una posició donada
- `Mitosi`: Crea un fill de l'organisme

- `Agressio`: Manlleva nutrients d'un altre organisme
- `Catabol`: Parteix un nutrient del pap en dos (reacció exògena)
- `Anabol`: Fusiona dos nutrients del pap en un (reacció endògena)

Les instruccions motores poden fallar. Quan ho fan, tenen un cost adicional. Aquest cost adicional per fallada és comú a totes les instruccions motores i es pot configurar.

En principi, en el sistema proposat, l'energia útil s'extreu només de reaccions metabòliques. Però, amb l'objectiu de simular entorns més senzills (sense metabolisme), és possible associar un guany energètic a la ingestió. També ho hem estés a la resta de funcions motores i sensores, de forma que es permet associar guanys i costos extres d'energia a cada instrucció.

12.3.1 Ingestió

Per introduir un nutrient lliure al medi dins del cos d'un organisme, cal precissar el lloc d'on vol extreure-ho, un patró i una tolerància. Si existeix l'element que compleixi aquests requisit, el nutrient és introduït a l'interior de l'organisme.

En principi, en el sistema proposat, l'energia útil s'extreu només de reaccions metabòliques. Però, amb l'objectiu de simular entorns més senzills (sense metabolisme), és possible associar un guany energètic a la ingestió.

12.3.2 Excreció

L'excreció mou un nutrient (especificat per un patró i una tolerància) del pap cap al medi a una posició indicada.

Aquesta instrucció s'ha introduit donat que és una forma en que l'organisme pot modificar el medi. És possible que, al llarg de l'evolució doni peu a comportaments complexos com el següents:

- Proporcionar a la descendència nutrients via excreció.
- Detectar als membres d'una espècie o el seu estat per les mol·lecules excretades.
- Transport de nutrients.
- ...

El caracter obert que poden adoptar les soluccions dels organismes implica que la llista anterior pot no ser tancada, o pot ser els organismes no arribin a cap dels punts anteriors.

De la mateixa manera que la ingestió es pot associar un guany energètic adicional, l'excrecció es pot configurar amb un guany o amb un cost adicional.

12.3.3 Agressió i defensa

Un organisme, de banda d'obtindre nutrients del medi, també els pot obtindre d'altres organismes. Per fer-ho, només cal indicar una posició on suposadament hi ha un altre organisme, un element base i una tolerància, i una clau d'atac. Aquesta clau d'atac, en

enfrentar-la a la clau de defensa de la victima en resulta la força final de l'atac. Aquesta força indica el nombre de vegades que es provarà d'extreure un nutrient del pap de la victima segons l'element base i la tolerància.

Com es pot deduir, d'un atac es poden obtindre molts nutrients al mateix temps la qual cosa la fa més profitosa que la ingestio de nutrients del medi. Això prova de compensar els desavantatges de desenvolupar una conducta depredadora i haver de dependre de les preses.

Es pot associar un guany energètic proporcional al nombre de nutrients extrets a un altre organisme. De la mateixa forma també es pot associar un cost energètic proporcional per a la victima.

12.3.4 Moviment

La instrucció per moure's és ben simple, només cal indicar una posició destí. Es pot realitzar si la posició està lliure. Té associada un cost configurable.

12.3.5 Mitosi

La instrucció `Mitosi` crea un fill de l'organisme a una posició indicada. Aquesta posició ha de estar lliure, si no, l'instrucció no s'executa.

Es crea el cos del nou organisme a partir del cariotip del progenitor, amb una certa probabilitat de mutació. Es situa al biòtop, i es demana per un identificador de taxo al taxonomista tot aportant el identificador de taxó del pare i indicant si ha mutat o no.

En el moment en el que s'introdueix l'organisme en la comunitat, aquest és elegible pel biosistema per ésser executat.

La mitosi té molts factors energètics configurables, donat que és una de les instruccions clau quan es vol equilibrar els costos per obtindre comportaments complexos.

Es requereix que hi hagi una certa energia disponible abans de reproduir-se. Si no es posés un límit d'aquest tipus, no seria necessari tenir una vida llarga per reproduir-se i amb una vida excessivament curta (en algunes proves els organismes arribaven a reproduir-se en les cinc instruccions que se'ls hi donaven de quantum), no hi ha lloc per a trobar comportaments interesants.

Llavors hi ha un mínim d'energia per poder iniciar la reproducció. Aquesta energia mínima no és el cost real de la instrucció, sinó que aquest és configurable independentment.

Una altra cosa que cal que sigui configurable és l'energia útil amb la que comença el nou organisme. Per ser consistents, és important que sigui bastant menor que el cost de reproduir-se perquè sinó els organismes no tindrien perquè menjar.

També s'especifica els nutrients del pap que passen a la descendència amb un patró, una tolerància i un número d'intents.

És possible penalitzar en aquesta instrucció al pares que tinguin un cariotip superior a un nombre de codons. Aquesta penalització es proporcional a la quantitat que es passa.

Aquesta penalització s'ha introduït per corregir l'efecte que produien alguns operadors de mutació que sovint incrementaven el tamany del cariotip afegint més redundància de la que era necessària per mantenir la variabilitat. Els cariotips grans afecten en gran mesura

la velocitat i la memòria ocupada pel sistema. Però, tampoc semblava correcte limitar el tamany restrictivament perqué algun cop pot arribar a ser útil aquest increment. Així doncs s'ha adoptat la solució també present a la natura: Un cariotip excessivament llarg costa més de replicar que un de curt. Això possibilita permetre els llargs sempre que serveixin per codificar millors solucions.

12.3.6 Anabolisme

L'anabolisme que implementa aquest prototip extreu dos nutrients del pap (mitjançant els respectius patrons i toleràncies) i els junta per formar un tercer, donant un balanc d'energia negatiu.

$A + B \rightarrow^{-E} C$

```
C = A & B
cost = PAnabol * (comptaUns(C)- min(comptaUns(A),comptaUns(B)))
```

A la fórmula, `PAnabol` representa un factor configurable.

12.3.7 Catabolisme

El catabolisme que implementa aquest prototip extreu un nutrient del pap (mitjançant un patró i una tolerància) i obté dos nutrients mitjançant una clau catabòlica.

$A \rightarrow^{+E} B + C$

El catabolisme requereix, a més una clau catabòlica T que es fa servir per calcular els productes.

```
B=A & T;
C=A & ~T;
energia = PCatabol * (comptaUns(A)- max(comptaUns(B),comptaUns(C)))
```

A la fórmula, `PCatabol` representa un factor configurable.

Capítol 13

Tecnologia emprada

Aquest capítol explica breument els conceptes tecnològics que han estat objecte de consideració, de cara a implementar l'eina.

13.1 Llenguatges orientats a objectes

Vaig escollir el paradigma de programació orientada a objectes degut als següents motius:

- S'ajusten molt bé a les aplicacions de simulació donat que existeixen objectes molt identificables.
- S'ajusta molt bé als conceptes de modularitat i testabilitat que es donen al llarg de la carrera.
- Volia poder aplicar algunes tècniques de disseny molt útils que havia estat estudiant aquest anys i que es basen en la programació orientada a objectes. Les explico més endavant.

En considerar el llenguatge de programació a utilitzar, em vaig plantejar quin dels llenguatges de programació orientada a objectes fer servir. Vaig preferir escollir un llenguatge que coneixés a fons i en el que tingués experiència per reduïr els imprevistos que em pogues trobar a causa de no conèixer a fons els seus mecanismes. Això deixava com a opcions més dessitjables el C++ i l'Smalltalk.

Smalltalk era el llenguatge que més bé s'adaptava a les tècniques que volia aplicar, permetia un desenvolupament ràpid i evitava molta reescriptura de codi. A més, molts conceptes de orientació a objectes no són directament aplicables a C++ sinó que requereixen alguns recargolaments per fer coses que són directes a Smalltalk i altres llenguatges dissenyat des del principi amb una mentalitat orientada a objectes.

Tot i així, vaig optar per fer-ho en C++ perquè:

- L'eina ha de poder ser modificada per tercers. Tot i que Smalltalk és prou fàcil d'apendre, el fet d'estar implementada en un llenguatge no massa extés, pot disuadir a gent de provar-ho.
- El codi en Smalltalk no és tan eficient com pot arribar a ser-ho en C++.
- Hi ha entorns C++ per a més plataformes.
- Els últims drafts de l'estàndard C++ inclouen algunes biblioteques que implementen d'una forma estàndard part de la funcionalitat de la extensa biblioteca de classes d'Smalltalk.
- Vaig trobar alguns mecanismes fixos que apropaven la forma de treballar C++ a la pròpia d'Smalltalk.

Així doncs, per poder apropar el C++ a la forma de programar d'Smalltalk, es van haver d'implementar alguns mecanismes interns d'Smalltalk que queden ocults al programador. També es fa un ús intensiu dels membres estàtics.

Algunes de les tècniques aplicades impliquen polimorfisme. C++ permet que dos funcions membre s'anomenin igual però el polimorfisme complert, el que permet que intercanviem un objecte d'un tipus per un altre, implica l'ús de *mètodes virtuals* i fer servir punters o referències, que en C++ implica, bé inseguretat (els punters no tènen perquè apuntar a un objecte vàlid), bé constància (l'objecte apuntat per la referència no es pot modificar).

Aquests impediments s'han resolt programant explícitament part dels mecanismes que implicitament fa servir Smalltalk per mantenir segures les referències.

Altres impliquen l'us massiu de membres (funcions i objectes) estàtics a les classes. No és que C++ no suporti aquest tipus de membres, el que passa és, al contrari de que passa amb Smalltalk, c++ no en fa un ús generalitzat i està molt mal documentat en la bibliografia C++. Alguns aspectes, com ara la inicialització de objectes membres estàtics els hem hagut d'extreure directament del draft de l'estàndard.

13.2 Biblioteca estàndard de C++

Al projecte, es fan servir de forma intensiva les llibreries estàndard de C++. L'ús que se'n fa normalment es limita a les classes d'entrada i sortida, però, un estudi més a fons del que s'ofereix pot donar molt de sí.

Entre les funcionalitats utilitzades al projecte en destaquem les següents:

Classes contenidores: Implementen estructures de dades d'ús comú: Llistes, cues, piles, vectors dinàmics, estructures d'accés per contingut... Aquestes estructures de dades es poden fer servir per contenir qualsevol tipus d'objectes mitjançant l'ús de templates suplint, en part, la funcionalitat de les *Collections* d'Smalltalk.

Cadenes de caracters: D'una forma estandaritzada s'encapsula un objecte per contindre text amb operacions consistents.

Objectes funcionals: Permeten manegar funcions com a objectes, operar amb elles (emmagatzemar, combinar, assignar paràmetres) i aplicar-les quan convé. És a dir, permeten la codificació dinàmica suplint de forma molt rudimentària els blocs de codi d'Smalltalk tot i que són molt més potents que els simples punters a funció de C.

Mecanismes de passivació: Encapsulen els receptors de dades en forma serialitzada. De banda de l'entrada/sortida comuna, es pot aprofitar el protocol per obtindre unes altres funcionalitats. Per exemple, en l'eina un objecte que fa servir el mateix protocol que `iostream` per visualitzar text en caixes de diàleg o de forma diferida en qualsevol altre lloc fent servir en el entremig una estructura de dades que ho memoritza.

13.3 Entorn de programació

S'han fet servir dos entorns de programació en diferent proporció durant el temps que ha estat desenvolupant-se l'eina.

Al començament el desenvolupament es feia de forma integra en el Microsoft Developer v5.0. El programa ens oferia ajuda ràpida sobre el lleguatge i la llibreria i ens permetia

organitzar la multitud de classes en una estructura jeràrquica l'`Infoviewer`.

Tot i així la compilació era molt lenta per la qual cosa es va començar a fer servir el GCC i les altres utilitats GNU en la seva versió per a DOS, el DJGPP.

Tot i així, es va estar fent servir el MSD5 com editor, debugger i ajuda, fins al final del projecte on la necessitat d'ajuda era molt ocasional. Llavors es va començar a fer servir l'editor VIM, un clònic millorat de l'editor VI de UNIX amb moltes més funcionalitats d'ajut a la programació i que té una versió GUI per a Windows.

Cal senyalar que ens hem trobat alguns problemes amb les característiques avançades del futur estandard C++. Les primeres versions de GCC provades i la versió 5 de MSVC++ no implementaven molt correctament algunes d'aquestes característiques. Sobretot els templates, els punters a funcions membre i la biblioteca estàndard.

Les últimes versions del GCC, ja implementent tot el necessari per que compili l'eina.

13.4 Tècniques de disseny

A continucació s'expliquen algunes tèniques de disseny que s'han aplicat.

13.4.1 Inicialització mandrosa

La inicialització mandrosa consisteix en no inicialitzar una dada fins que no es necessita. És molt útil quan aquesta inicialització és molt costosa i no sempre sigui necessària.

13.4.2 Double Dispatch

La natura polimòrfica dels llenguatges orientats a objectes ens dóna problemes quan cal saber el tipus d'un paràmetre que arriba a un mètode, per poder processar-ho.

Hi han molts casos en que ens és molt ineficient preguntar per tots els tipus de paràmetres que hi podem passar. Dona un codi, molt feixuc i cada cop que afegim un tipus de paràmetre cal modificar el codi del mètode.

La tècnica del Double Dispatch el que fa és simplement passar-li la pilota al paràmetre. Cridem a un mètode secundari al que tota classe succeptible de ser passada com a paràmetre ha de respondre.

13.4.3 Mètodes Constants

Si, dintre d'una classe, en comptes de fer servir una constant, cridem un mètode de la classe que ens la retorna, a les subclasses podrem redefinir aquest mètode per que en retorni un altre valor. Així no cal reescriure l'algorisme que la fa servir.

13.4.4 Policy Classes

Les policy classes (classes de polítiques) són aquelles que encapsulen algorismes sense dades. En comptes de posar l'algorisme a dintre d'un mètode de classe, el posem com a mètode de la seva propia classe, rebent els paràmetres oportuns. D'aquesta manera aquest algorisme:

1. Es pot reaprofitar, cridant-lo en altres contextos.

2. Es pot intercanviar facilment per un altre amb el mateix protocol.

3. Permet modificar el comportament d'un objecte a nivell d'instància sense fer una subclasse, simplement modificant l'objecte policy.

13.4.5 Classes singulars

Una classe singular (Singleton class) és una classe que permet una sola instància.

Per definir un Singleton definim una variable de classe que contindrà aquesta instància única i un mètode de classe per accedir-hi que sol ser *current*. El mateix mètode pot servir per inicialitzar mandrosament el Singleton.

La inicialització de Current també es pot realitzar mitjançant un missatge d'inicialització de classe.

A vegades, es pot fer servir un mecanisme similar als singletons per mantenir una instància com a instància distingida o per omisió, tot i deixant que es puguin crear tantes instàncies com es vulgui. El procediment és pràcticament idèntic. Només que els mètodes de creació no s'invaliden i que, en comptes de fer servir la variable *Current*, per convenció, fem servir la variable *Default* i els noms dels mètodes d'accés canvien en conseqüència.

13.4.6 Classes adaptadores

Si una classe s'adapta funcionalment a les nostres necessitats, però, la interficie no és la desitjada, podem arreglar-ho mitjançant una classe adaptadora. Aquesta classe contindrà, com a variable d'instància, una instància de la classe adaptada a la qual traduirà tots els

missatges que li arribin suplint-li les possibles carències.

Les classes adaptadores serveixen per reutilitzar una classe en un entorn per el qual no estava dissenyada.

13.4.7 Classes representants

Una classe representant és un cas semblant al de una classe adaptadora. Ara, però, no provem d'adaptar els protocols sinó que la classe representant es fa servir com si fós la classe representada amb el mateix protocol. De fet, interesa que el client de la classe no noti diferències substancials entre una i altra i la classe representada no té perquè estar inclosa dintre de la representant.

La representació pot tenir tot un seguit de objectius i funcionalitats:

- **Control d'accés:** La classe representant controla l'accés a la classe representada, per exemple, per fer una comprovació de permisos, per evitar que certs missatges arribin, o per fer comprovacions previes com ara exclusions mutues.
- **Representació remota:** La classe representada no està present físicament. Per això, la classe representant es fa passar per la classe representada de cara al client, mentres que, per un altre costat, prova de fer arribar els missatges a la instància de la classe representada (via xarxa o similar).
- **Simulació:** La classe representant simula el comportament de la classe representada. Molt utilitzat en desenvolupament per fer les proves.

- **Representació mandrosa:** L'objecte representat no existeix de fet fins que el representant el crea mandrosament quan es requereix i s'intercanvia amb el missatge *become: anObject*.

13.4.8 Classes abstractes i classes concretes

Si dos classes tenen en comu característiques però no es pot dir que una sigui subclasse de l'altre perquè totes dues tenen coses diferents, el més elegant és crear una classe que contingui tot allò que és comú a les dues classes: Representació (variables), Comportament (contingut de les funcions) i Protocol (capceleres de funció). Aquesta clase se'n diu classe abstracta degut a que no està dissenyada per generar instàncies si no que és un nexe conceptual entre les seves subclasses.

En contraposició, anomenarem classes concretes a les classes dissenyades per generar instàncies.

13.4.9 Classes abstractes factoria

Una classe abstracta factoria és un tipus especial de classe abstracta que implementa mètodes factoria per a les seves subclasses concretes. L'elecció de la subclasse de la qual es genera la instància pot dependre del contigut del missatge de creació o del entorn.

Si el protocol de la classe abstracte es manté a les classes concretes, el client de la classe abstracte factoria obtindrà indistintament instàncies d'una classe o d'una altre a les que podrà accedir tot de forma transparent.

13.4.10 Classes per portabilitat

Si el nostre codi ha de conviure amb diversos sistemes, per reduir l'impacte de les parts no portables a la nostra aplicació, convé encapsular-les en objectes que ofereixin una interfície uniforme a la resta de l'aplicació.

Capítol 14

Metodologia d'implementació i estil de programació

Aquest capítol està orientat a les persones que vulguin fer modificacions del sistema. Descriu la metodologia d'implementacio i l'estil de programació. És molt aconsellable seguir aquestes directrius.

14.1 Registre de canvis

Cada arxiu d'implementació, porta a l'inici un registre dels canvis que s'anat fent al fitxer (Change Log). Cada entrada d'aquest registre porta la data, un indicador de l'autor de la modificació i una breu explicació d'una o dos línies, suficient per deduir en què consisteix i a quins llocs afecta.

14.2 Registre de coses pendents

El control de les coses pendents (informalment, *TODO's*) resulta molt important per no deixar qüestions deslligades, donada la quantitat de coses que cal tenir presents durant la implementació.

Per tenir-ne constància de les coses que s'han anat deixant pendents, s'han anat mantenint registres a tres punts diferents:

- Les coses pendents que han d'anar a un punt concret a dins del codi s'indiquen al mateix lloc on caldrà afegir-ho. S'indica amb un comentari d'una sola línia que comença amb:

  ```
  // TODO:
  ```

  El text del comentari ha de ser suficientment explicatiu, perquè no es necessiti veure'l en el seu context per entendre'l. Tot això es fa així perquè es pugui obtindre un extracte de totes les modificacions pendents d'aquest tipus només executant la comanda:

  ```
  grep -n TODO *.h *.cpp *.c
  ```

- A l'inici del fitxer d'implementació, a continuació del *Change Log*, es posen els canvis pendents que afectan al mòdul en general i que no es puguin localitzar a cap lloc en concret dins del codi.

- En un fitxer a part anomenat `TODO.txt`, s'han anat recopilant i actualitzant periòdicament

els canvis pendents que persisteixen d'entre els anteriors i alguns altres que afecten a diversos mòduls o a mòduls que encara no s'han construït.

14.3 Control de versions

Cada cop que es compila, es genera de forma automatitzada una entrada a un log de compilacions amb la data i el número de compilació. Al mateix temps es modifica un arxiu font per tal que aquest número de compilació i la data estiguin disponibles per al programa.

Això ens permetrà saber, donat un executable, fins a quin punt està actualitzat, i amb els *Change Logs* quines característiques inclou. El registre de compilacions facilitarà, a més, una millor aproximació del temps d'implementació.

El registre de compilacions el manté el programa `buildnum` el codi font del qual està disponible al comprimit. Cal cridar-lo cada cop que intentem compilar el programa.

El programa modifica els fitxers `build.h` i `buildlog.txt`. El primer és la informació que s'inclou al programa. El segon fitxer és el registre de compilacions.

Per accedir a la informació de l'aplicació, inclosa la que hi ha al `build.h`, és aconsellable la utilització de la classe `CAppInfo`.

14.4 Fitxers

Tot i que la intenció inicial era mantenir per a cada classe un fitxer de prototipus i un altre d'implementació, l'ús massiu de les classes ha obligat a fusionar algunes classes en el

mateix parell de fitxers.

Això sí, només s'ha fussionat en un fitxer classes molt intimament lligades com ara subclasses d'una mateixa classe abstracta factoria en els casos en els que el codi que aportava cada subclasse era molt poc i molt uniforme.

En aquests casos, la classe abstracta factoria té el seu propi fitxer de prototipus de cara a que els seus clients el puguin incloure sense que interfereixi l'existència de les subclasses. La resta s'ha agrupat en un o més.

Generalment el fitxer de la classe abstracta té el nom de la classe en singular i el de les classes derivades en plural. Fent servir un exemple típic, el fitxer `Persona.h` contindria el prototipus de la classe abstracta `CPersona`, i el fitxer `Persones.h` podria contenir les especialitzacions de la classe `CClient` i `CEmpleat` sempre que aquestes classes no afegissin mètodes adicionals al protocol públic de `CPersona`.

14.5 Criteris de nomenclatura d'identificadors

En molts, casos s'han adoptat alguns criteris que es fan servir en la programació d'Smalltalk.

En general, els identificadors que representen diverses paraules hem adoptat el criteri de fer servir les majúscules per separar-les en comptes del símbol de subratllat com és costum entre alguns programadors de C. Així doncs, farem servir `unIdentificadorLlarg` en comptes de `un_identificador_llarg`.

Els identificadors de les funcions, mètodes de classe (estàtics en nomenclatura C) i objectes globals, els hem començat preferentment per una majúscula. També els noms de les classes

i els `namespace`'s.

La primera paraula dels altres identificadors (dades locals o membres, funcions membres no estàtiques...) he adoptat el conveni de començar-la en minúscula.

També he pres alguns convenis estesos en la programació per a windows. Per exemple:

- Preposem una `C` majúscula als identificadors de les classes: `CComunitat`
- Preposem `m_` als identificadors de dades membres no estàtiques: `m_unaVariableMembre`
- Preposem `s_` als identificadors de dades membres estàtiques: `s_unaVariableEstàtica`

14.6 Proves unitàries de classe

Cada classe té una funció membre estàtica anomenada `ProvaClasse` on es deixa tota la bateria de proves unitàries que s'han fet sobre la classe, per, en cas de modificacions, tornar-les a passar.

Els mòduls que no estiguin encapsulats en classes també tindran una funció similar. Generalment per ortogonalitat i per no interferir en l'espai de noms, la funció de proves del mòdul es posa a dins d'un `namespace` sinò hi està posat ja tot el mòdul.

14.7 Funcions dels comentaris al codi

Els comentaris dels fitxers estan tipificats segons la seva funcionalitat. De banda dels que es fan servir per les funcions ja explicades (Change Log i TODO's), tenim altres funcionalitats:

Comentaris de mòdul: Serveixen per explicar qué va al mòdul que encapçalen i si hi ha alguna consideració global que fer en usar-lo o mantenir-lo. Es troben a l'inici del fitxer, juntament amb el Change Log i els TODO globals pel mòdul.

Comentaris de secció: Separen visualment les diferents seccions d'un mòdul. Tots els mòduls estan dividits d'una forma molt semblant i cada divisió es troba, generalment en el mateix ordre per tal de trobar fàcilment les funcions. Exemples de seccions comunes són: Inicialització de variables estàtiques, Construcció/destrucció, Mètodes redefinibles a les subclasses, Operacions (de diferents tipus), Proves... Es distingeixen visualment per estar envoltats per un parell de línies a dalt i a baix com el següent exemple:

```
////////////////////////////////////////////////////////////////////
// Nom de la secció
////////////////////////////////////////////////////////////////////
```

Comentaris d'encapçalament: Es troben just després de l'encapçalament d'una funció o mètode i just abans de que s'obrin els claudàtors del seu cos. Aquests comentaris van adreçats als usuaris potencials de la funció i explica qué és el que fa evitant qualsevol menció als detalls d'implementació. Si cal indicar, precondicions o postcondicions es farà aquí, dedicant, a cadascuna, una linia que vindrà precedida de les partícules `Pre:` o `Post:` segons convingui.

```
int unaFuncio (int param1, it param2)
// Comentari d'encapçalament
// Pre: Precondició
// Post: Postcondició
{
     ...
}
```

Comentaris de manteniment: Aquests comentaris es troben al cos de la funció o mètode (entre els claudàtors), parlen de detalls d'implementació i estan adreçats als mantenidors.

14.8 Eines i ajudes a la implementació

En aquest apartat s'expliquen algunes eines que s'han implementat per tal d'afavorir la implementació de la resta del sistema.

14.8.1 Funció de compatibilitat de claus

Al sistema resulta molt important una funció que determini, a partir de dos claus, quin els el grau de compatibilitat de les dues.

La compatibilitat entre claus es fara servir, per exemple, per a la identificació d'organismes (amb l'objectiu de cercar preses, col·legues, progenitors...), identificació de nutrients (amb l'objectiu d'ingerir-los, evitar-los, detectar excrecions ajenes, controlar els processos metabòlics interns...) i contesa (mecanismes de depredació i defensa). Donat que aquesta funció és una de les més utilitzades al sistema, ha de ser molt poc costosa.

Cal que la funció no tingui en compte la ponderació dels bits que formen la clau i que els tracti tots de la mateixa forma perquè no es converteixi en una optimització numèrica. A més, és desitjable que aquesta funció permeti nivells de tolerància variables i un cert indeterminisme.

Necessitem tenir en compte, doncs, tres elements:

- El grau de compatibilitat entre les claus.
- Una tolerància quantificable sobre les variacions entre claus.
- Un element indeterminístic que permeti resultats diferents amb les mateixes entrades.

Si les claus les representem amb dos enters de 32 bits, el grau de coincidència el podrem obtenir fent-ne la o exclusiva bit a bit i complementant el resultat. Al número obtingut l'anomenarem coincidència (C). El nivell de tolerància també pot ser un enter (T) que ens vindrà donat i l'indeterminisme el pot introduir un altre enter (R) tret d'una funció pseudo-aleatòria.

Opció 1

Els uns del número generat pseudo-aleatòriament (R) es 'filtren' per la coincidència (C) de tal forma que nomes arribin els uns que estiguin en una posició on no hi havia coincidència entre claus. La tolerància (T) indica el número d'uns que admetem com a màxim per aceptar les claus com a compatibles.

$$ComptaUns(R\& \sim C) < T \tag{14.1}$$

El punt negre d'aquest mètode és l'alt cost de la funció ComptaUns, donat que no és una operacio nativa a la majoria de màquines i cal implementar-la a base de desplaçaments i enmascaraments.

La següent gràfica mostra la probabilitat de que dos claus de 32 bits siguin compatibles segons el bits que tinguin igual i per diferents valors de T. La T pot oscilar entre 0 i 32 tot

i que veiem que la distribució no pateix variacions apreciables per valors a partir de 24 o potser abans. Podriem molt be limitar-la entre 0 i 15 sense perdre gaire significat.

Figura 14.1: Probabilitat d'encert amb la funció de compatibilitat número 1

La distribució sembla ideal pel que volem: Per valors de poca tolerància, la probabilitat és gairebé nul·la, per toleràncies molt grans ho deixa passar gairebe tot i per a una sèrie de valors intermitjos on es mantenen tres zones:

- Una zona de pas incondicional, per les coincidències més altes.
- Una zona intermitja on la probabilitat de pas depèn de la coincidència.
- Una zona de tall incondicional, per les coincidències més baixes.

Opció 2

Una altra opció és fer servir la tolerància com una altra màscara. Un bit a un a la tolerància voldria dir que es tolera que aquest bit resulti a un després del filtratge. La condició que determina que dos claus són compatibles quedaria com segueix:

$$(\texttt{R\&} \sim \texttt{C\&} \sim \texttt{T}) == \texttt{0} \tag{14.2}$$

Aquí sí que T agafa tota la franja dels 32 bits. Per fer la gràfica i obtenir un resultat comparable amb l'anterior, s'ha considerat el número de uns a la T en comptes del seu valor.

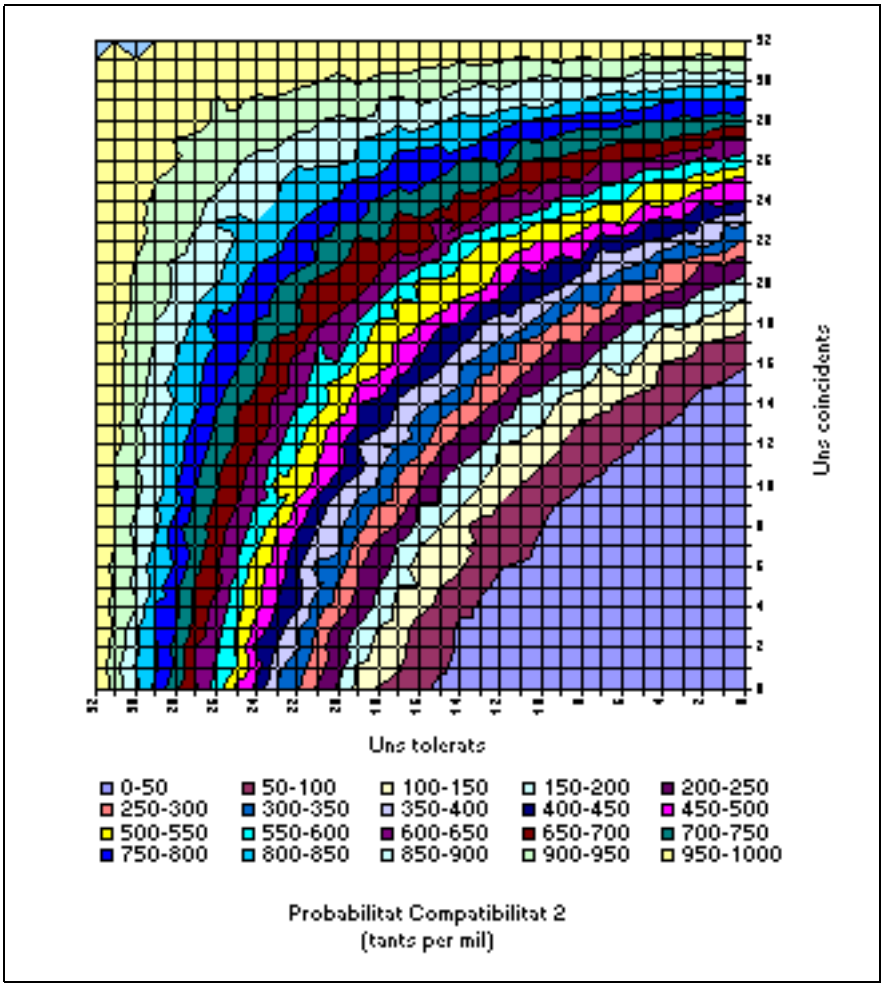

Figura 14.2: Probabilitat d'encert amb la funció de compatibilitat número 2

Observem que hem perdut les zones d'aceptació i rebuig incondicional a les toleràncies intermitges, però, la funció és bastant vàlida pels objectius donat que és una aceptació no determinística on la probabilitat depèn de la tolerància i de la coincidència, i, a més, hem optimitzat moltíssim el cost d'avaluació.

Com a característica afegida, aquesta funció permet, mitjançant la tolerància, un control més acurat de quins bits són els que poden no coincidir. Aquesta peculiaritat pot donar

peu a mecanismes més complexos, més que no pas ho faria una tolerància cega. A mès, tot i que es té en compte la posició dels bits, no els pondera, com les altres fòrmules provades.

Altres opcions desestimades

Altres funcions de compatibilitat han estat provades i del tot desestimades pel seu alt cost i/o per la seva poca idoneïtat.

Per exemple, es va provar la funció

$$ComptaUns(R\& \sim C\& \sim T1) < (T2\&0x5) \tag{14.3}$$

per sintetitzar en una fòrmula els dos conceptes de tolerància que hem vist, una tolerància que dóna significat a la posició dels uns i una altra que permet tolerar globalment un cert nombre d'uns independentment de la posició.

Degut als pocs bits (32) amb els que juga i a que hi havia dos punts on es tolera, la funció, lluny de donar tot el significat que volíem, dóna molt poca variació amb els paràmetres. A més, tornem a tenir el problema de la funció ComptaUns.

$$R >> (T\&0x7) < (C << ((T >> 3)\&0xf)) \tag{14.4}$$

14.8.2 Dispositius d'entrada i sortida portables

Seguint el paradigma model-vista-controlador el nucli del sistema, el model, hauria de ser independent de l'entorn on executem l'aplicació. Tot i així, a dintre del nucli cal fer algunes

operacions d'entrada i sortida, com a mínim per fer les tasques de depurat i els missatges d'error. Per això, ens facilitaria molt les coses que un objecte que tingués un comportament semblant a un `iostream` de C++ però que permeti redireccionar els missatges per gestionar com es visualtitzen depenent de l'entorn destí.

S'ha implementat un objecte `CMissatger` que es comporta de forma molt similar als `iostreams`. Aquest objecte conté una referència a un objecte que pertany a una classe derivada de la clase abstracta `COutputer`. La classe abstracta `COutputer` defineix un protocol molt senzill d'inserció de missatges, per ser controlat per `CMissatger`. Segons la subclasse a que pertanyi l'objecte `COutputer` els missatges insertats es visualitzen d'una forma o d'altra.

Ara mateix estan implementats els seguents `COutputer`'s:

- Consola estàndard
- Control d'edició de MS-Windows
- Caixes de missatges de MS-Windows
- Pop up de la llibreria Curses
- Un scroll limitat de la pantalla fent servir codis ANSI
- Una llista d'strings de la llibreria STL (per imprimir-los en diferit)

En cas de voler un altre dispositiu de sortida, només cal crear el COutputer adient que és una tasca molt senzilla.

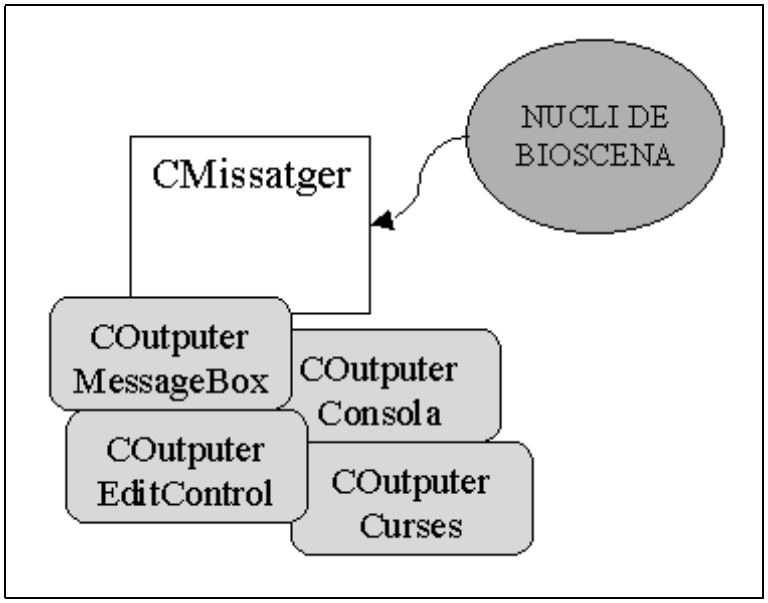

Figura 14.3: Encapsulament dels dispositius de sortida

14.8.3 Seqüències d'escapament ANSI

De cara a obtenir una sortida rica, però, conservar el caracter portable d'aquest primer prototipus, s'ha optat per fer servir seqüències d'escapament ANSI en un terminal de text. Aquestes seqüències permeten fixar colors, posicionar el cursor, netejar la pantalla... i d'altres operacions en terminals que compleixin aquest estàndard. Això inclou els terminals de Linux i les consoles de MS-DOS i MS-Windows després d'instalar el controlador `ANSI.SYS`.

S'ha optat per implementar una biblioteca pròpia per ajudar a insertar aquests codis donat que totes les biblioteques provades que treballaven amb aquestes seqüències, no eren prou òptimes com per l'ús masiu que calia fer d'elles. A més, tampoc feien un enfoc compatible amb els `iostream`'s de C++.

Com que els símbols que defineix la biblioteca de codis ANSI es poden insertar sense

problemes en un `iostream` qualsevol, no hi ha cap problema en enviar-los a un fitxer obert, o la a classe `CMissatger` implementada (veure 14.8.2).

Generalment la inserció optimitzada simplement inserta una cadena predefinida, o, si hi ha paràmetres, substitueix els caracters que varien d'un a l'altre a una còpia de la cadena original. Es podria optimitzar un xic més del que està no copiant la cadena, però, seria a costa de comprometre la integritat del sistema, si algun dia, l'aplicació ha d'executar-se en multithread.

La biblioteca també implementa la classe `CColor` que implementa operacions amb atributs de color pels caracters. Els objectes CColor en ser inserits en un `stream` inserta la seqüència d'escapament corresponent.

En resum, la biblioteca permet expressions com aquesta:

```
cout << clrscr << gotoxy(3,4) << blau.fons(vermell)
     << "Hola Món" << blau.brillant() << "!!!" << endl;
```

que es resolen de forma bastant optima comparant-ho amb altres llibreries de l'estil.

A mode d'exemple posem com hem solucionat cada tipus de seqüència:

```
// Sequencia constant
const char clrscr[] = "\033[2J";


// Seqüència amb parametres de longitud fixa
string color (int fg, int bg)
{
  // Aquesta es la copia necesaria per permetre multithreading
  string ansiseq ("\033[0;30;40m");

  ansiseq[2]=(fg&0x08)?'1':'0';
```

```
  ansiseq[5]='0'+(fg&0x07);
  ansiseq[8]='0'+(bg&0x07);
  return ansiseq;
}

// Seqüència amb parametres de longitud variable
string gotoxy(int col, int lin)
{
  ostrstream str(myBuffer,16);
  str << "\033[" << lin << ';' << col << 'H' << ends;
  return str.str();
}
```

14.8.4 Objecte configurador

Els objectes configuradors llegeixen un fitxer de configuració amb paràmetres numèrics sobre el sistema. Cada paràmetre té associat un identificador pel qual pot ser consultat el paràmetre.

L'avantatge d'aquest procediment per configurar el sistema és que és molt fàcil per el programador afegir paràmetres configurables amb valors per defecte al sistema.

Capítol 15

Notes pel manteniment de l'aplicació

15.1 Implementació del medi. Biòtops

En aquest apartat i en els següents del capítol s'explica com està implementat el medi i com s'hi poden fer modificacions mitjançant programació. Aquests apartats no són d'interes als usuaris que no vulguin modificar el programa.

Per implementar el medi, tenim tres elements: La topologia, el substrat i el biòtop que és l'associació d'una topologia i un tipus de substrat.

Per definir el tipus de substrat cal una classe les instàncies de la qual representin el substrat de cada posició individual, és a dir, una casella.

Totes les topologies deriven de la classe `CTopologia`, que defineix el protocol de mètodes virtuals als que una topologia ha de respondre per definir la seva geometria.

Per poder escollir diferents substrats i topologies sense gaire esforç de programació s'ha creat una classe patrò (*template*) anomenada `CBiotop`.

Un biòtop és una especialització d'una topologia. Aquesta especialització es basa en dos fets:

- Per un costat, un biòtop conté un vector de substrats discrets, als quals es pot accedir amb un identificador de posició. El tipus del substrat és el paràmetre del patrò.
- El comportament que té un biòtop com a topologia, és a dir, de cara a establir la geometria del medi, es pot determinar en temps d'execució tot fent que les crides pròpies de topologia es redireccionin cap a un objecte topologia indicat.

En resum, es pot determinar en temps de compilació el tipus de substrat amb el paràmetre del template i, la topologia, en temps d'execució, tot assignant-li una topologia o una altra a la que redireccionar les crides.

15.2 Programació de nous substrats

Podem dissenyar un substrat sense preocupar-nos de cap altre element del biòtop. Hi ha, però, dependències entre el disseny del substrat i la resta del sistema. Estan localitzades en:

- Execucions de les instruccions dels organismes (Biosistema)
- Interaccions amb els agents ambientals
- Interaccions amb el sistema de monitorització i logs.

Es recomana implementar els procediments `load` i `store` per pasivitzar i recuperar l'estat de la cel·la a un medi serie.

El substrat implementat a l'eina i descrit en l'apartat 7.4 és la classe `CSubstrat`.

15.3 Programació de noves topologies

Si l'usuari necessités crear un nou tipus de topologia, cal que la faci heretar de CTopologia i redefineixi els mètodes del protocol d'accés que CTopologia estableix.

El protocol està pensat per aconseguir que la resta de sistema només hagui de manegar identificadors de posicions i identificadors de desplaçaments. El significat geomètric d'aquests identificadors queda ocult darrera de l'implementacio del protocol.

Quan es deriva de CTopologia, el principal que caldria redefinir, si cal, és:

- Un **constructor** amb els paràmetres significatius per a la topologia. Per exemple, en una topologia rectangular és significatiu indicar l'altura i l'amplada. Cal acutalitzar la variable interna que guarda el nombre de posicions.
- `t_posicio CTopologia::desplacament (t_posicio origen, t_desplacament desplacament)`: Una funció per averiguar la posició destí en aplicar-li un vector de desplaçament a una posició origen. CTopologia, la defineix, per defecte, de tal forma que el resultat és una posició destí aleatòria vàlida.
- `bool CTopologia::esValid(t_posicio id)`: Una funció per saber si un identificador és vàlid. Només cal redefinir-ho si es modifica la correspondència directa entre

identificador de posició i index de casella en l'array de substrats reservada per CTopologia::CTopologia

- `t_posicio CTopologia::posicioAleatoria ()`: Una funció per obtindre aleatòriament una posició vàlida de la topologia. La funció general que no caldria redefinir seria

```
{
    uint32 pos;
    do {pos=rnd.get();} while (!esValid(pos));
    return pos
}
```

però, CTopologia no fa servir aquest algorisme donat que optimitza agafant un número aleatori entre 0 i N. Aquesta optimització funciona mentre es mantingui la correspondència entre identificador i index abans comentada. Si la subclasse la trenca, es quan cal redefinir la funció.

- `bool CTopologia::unio (t_posicio origen, t_posicio desti, t_desplacament & desp)`: Una funció per calcular el primer d'un conjunt de desplacaments que cal fer per anar de l'origen al destí. Retorna cert si el desplacament és suficient per arribar a la posició destí. CTopologia, la defineix de tal forma que el resultat és un desplaçament aleatori i retorna sempre fals (mai hi arriba).

Com a exemple, és molt aconsejable fixar-s'hi en com està feta la classe `CTopologiaToroidal` abans d'implementar una topologia pròpia.

15.4 Programació de biòtops dinàmics o heterogènis

Per la forma en que està dissenyada la classe `CBiotop`, els biòtops que es poden construir actualment ténen diverses restriccions:

- **Controlen un grup estàtic de posicions:** El nombre de posicions està limitat en el moment de creació. La topologia pot controlar el nombre de posicions disponibles però sempre dintre d'aquest límit establert inicialment.
- **El conjunt de substrats és homogeni:** No poden conviure substrats de diferents classes. Poden diferir les propietats però han de ser de la mateixa classe.
- **Es restringeix els identificadors:** Han de estar entre 0 i $N-1$ sent N el nombre de caselles màxim determinat inicialment.

Aquestes restriccions conseqüències directes del fet de que el conjunt de substrats s'hagi implementat amb un vector estàtic.

Si el biòtop que es vol simular no compleix aquestes característiques ni es pot simular el comportament desitjat, llavors caldria modificar la classe CBiotop, o crear-ne una classe nova similar.

15.5 Programació de nous agents

De cara a afegir nous agents al sistema, s'aconsella seguir els següents passos:

1. Llegir per sobre el codi dels agents ja implementats per assimilar les solucions que s'han donat a problemes que segurament es tornaran a repetir als nous agents. També convé mantenir uniforme l'estil de programació i l'ordre intern dels fitxers per fer-ho més mantenible a tercers. El més pràctic es partir d'una còpia d'un agent que tingui, estructuralment, tot o gran part del que interesa implementar.

2. Escollir la classe d'agent de la que volem heretar l'agent nou. Generalment voldrem que el nou agent pertanyi a un dels quatre grans grups funcionals d'agents:

   - Subordinadors (CMultiAgent i subclasses) si controla l'accionat d'altres agents
   - Posicionadors (CPosicionador i subclasses) si controla una posició en el biòtop
   - Direccionadors (CDireccionador i subclasses) si controla una direcció
   - Actuadors (CActuadors i subclasses) si modifica el substrat a una posició

   Si no pertany a cap dels quatre grups, caldria plantejar-se heretar de CAgent directament. En aquest cas, convé fer un esforç i fer una classe intermitja que pugui englobar altres agents en el futur. En endavant, anomenarem CAgentNou al nou agent afegit i CAgentVell a l'agent del qual heretem.

3. Adaptar el constructor de CAgentNou per que proveeixi els paràmetres del constructor de la superclasse. Posicionadors i direccionadors, per exemple, necessiten una referència a una topologia en el constructor.

4. Afegir dins del constructor, la línia.

   ```
   m_tipus+="/ElMeuSubtipus";
   ```

   que afegeix la cadena de subtipus a l'identificador de tipus que hereta de la superclasse.

5. Afegir els nous atributs (variables membre) dels que en depèn l'estat de l'agent i les funcions d'accés als mateixos.

6. Inicialitzar dins del constructor els nous atributs als valors per defecte. Els atributs que siguin dependències amb altres agents, o agents subordinats, es recomana que siguin punters, i no referències, per poder-ho deixar sense especificar al constructor. S'inicialitzen sempre com a punter a NULL. Cal procurar que, si el punter no apunta a un agent vàlid el seu valor sigui NULL i tenir-ho en compte quan hi accedim per evitar accesos il·legals a memòria. Es veu clarament aquesta idea llegint el codi d'alguns agents que ho fan.

7. Afegir dins del destructor, l'alliberament de memòria ocupada pels agents subordinats. Les dependències no s'han de alliberar pas.

8. Redefinir la funció membre `virtual void CAgentNou::operator() (void)` per que faci el que hagi de fer quan l'agent és accionat. Si es tracta d'un actuador, no cal redefinir aquesta sino `virtual void CAgentNou::operator() (CSubstrat & s)` on `s` és el substrat que hem de modificar.

9. Redefinir la funció `virtual void CAgentNou::dump(CMissatger & msg)` per que cridi a la funció corresponent de la superclasse (CAgentVell a l'exemple) i, després, inserti en el CMissatger les noves línies de configuració dels paràmetres que afegeix l'agent:

```
void CAgentNou::dump(CMissatger & msg)
{
    CAgentVell::dump(msg);
    msg << "- UnParametreNou " << m_valor1 << " " << valor2 << endl;
    msg << "- UnAltreParametreNou " << m_valor3 << endl;
}
```

10. Redefinir la funció `virtual bool CAgentNou::configura(string parametre, istream & valors, t_diccionariAgents & diccionari, CMissatger & errors)` per mirar si el `parametre` és un dels que ha afegit CAgentNou. Si ho és cal parsejar l'istream `valors` en busca dels valors corresponents, reportar els errors que es produeixin pel CMissatger `errors` i retornar cert per dir que el paràmetre era de la classe. Si no ho és, cal cridar a la funció corresponent de la superclasse per que ho pugui interceptar ella. El diccionari serveix per, donat un nom d'agent de l'arxiu, obtindre un punter a l'agent que s'ha creat que, pot ser, té un nom diferent. El diccionari és un `map<string, CAgent*>`, el seu funcionament s'explica a qualsevol manual sobre les Standard Template Libraries de C++. L'estructura general de la funció `configura` quedarà com això:

```
bool CAgentNou::configura(string parametre, istream & valors,
t_diccionariAgents & diccionari, CMissatger & errors)
{
    if (parametre=="UnParametreNou") {
        // Parsing dels valors...
        return true;
    }
    if (parametre=="UnAltreParametreNou") {
        // Parsing dels valors...
        return true;
    }
    // Li deixem a la superclasse que l'intercepti si vol
    return CAgentVell::configura(parametre, valors, diccionari, errors);
}
```

11. Si cap dels atributs (m_dependencia a l'exemple) és una dependència amb altre agent, cal redefinir la següent funció com segueix:

```
list<CAgent*> CAgentNou::dependencies() {
    list<CAgent*> l=CAgentVell::dependencies();
```

```
    if (m_dependencia) l.push_back(m_dependencia);
    return l;
}
```

12. Si cap dels atributs (m_subordinat a l'exemple) és un agent subordinat, cal redefinir la següent funció com segueix:

```
list<CAgent*> CAgentNou::subordinats() {
    list<CAgent*> l=CAgentVell::subordinats();
    if (m_subordinat) l.push_back(m_subordinat);
    return l;
}
```

13. Afegir a l'arxiu `Agent.cpp` un include a `AgentNou.h` i, a la funció estàtica CAgent::CreaAgent(...) una línia com les que ja n'hi ha per cada tipus d'agent, però, per a CAgentNou. Això permet que la funció CAgent::ParsejaArxiu pugui reconèixer el nou tipus als arxius de configuració.

De tots els punts anteriors el que potser és una mica més particularitzat són els atributs i els mètodes d'accés als mateixos, i el mètode d'accionament (o d'actuació en el cas dels actuadors). Per a la resta de coses el més pràctic es fer un cut&paste dels agents ja implementats i retocar el mínim.

15.6 Arxius de configuració d'agents ambientals

Per abocar a disc i per recuperar un agent, tota la feina bruta la fa la classe abstracta factoria `CAgent`.

En l'abocat, `CAgent` engega primer un recorregut en arbre per abocar els noms i els tipus dels agents. Després, en fa un altre per fer el mateix amb els paràmetres.

En la recuperació, la classe `CAgent` llegeix els noms i els tipus. Amb el tipus, crea l'objecte corresponent i després l'introdueix en un diccionari que relaciona els noms del fitxer amb el punter a memòria.

Tot agent té un nom únic al sistema. En un arxiu, aquest nom serveix per fer-ne referència. En memòria les referències es fan directament amb punters i el nom només serveix per quan ho grabem.

Però, quan llegim un arxiu d'agents, és possible que un nom dels que es fan servir al fitxer ja es trobi a memòria i caldrà renombrar-ho. Per això, es mantenen les els noms de referència del fitxer, mitjançant un diccionari.

Un cop creats els objectes `CAgent`, cal modificar els paràmetres per defecte. La classe factoria `CAgent` determina a quin agent del diccionari pertany cada conjunt de línies de configuració i va enviant els noms de paràmetres que troba i els valors perqué l'agent els faci servir per configurar-se.

Donat que hi ha diversos nivells de subclasses als agents i cadascuna té els seus paràmetres a configurar, es segueix la mateixa estratègia que segueix el pas de missatges amb el que Smalltalk implementa l'herència.

Cada nivell d'herència d'una subclasse de `CAgent` té uns paràmetres que reconeix, i que pot configurar. Si un nivell d'herència d'un objecte agent rep un nom de paràmetre i un stream amb els valors associats, mira si el parèmetre és un dels que reconeix, si ho és, parseja

l'stream dels valors. Si no és un paràmetre seu, li passa el paràmetre al següent nivell (la superclasse). Si el paràmetre arriba a `CAgent` i no ha estat configurat es dóna un error.

Per abocar, simplement cada subclasse aboca els seus paràmetres i indica a la seva superclasse que faci el mateix amb els seus.

15.7 Implementar nous sistemes de control

Tot sistema de control deriva de la classe abstracta factoria `CControlOrganisme`. Aquesta classe defineix el protocol d'accés a un sistema de control i mètodes per crear-ne sistemes de control a partir de l'identificador de tipus.

```
class CControlOrganisme
{
// Construccio/Destruccio
public:
    CControlOrganisme() {}
    virtual ~CControlOrganisme() {}
// Operacions
public:
    // Construeix l'estructura de control a partir d'un cariotip
    virtual bool init(CCariotip& c)=0;
    // Dona una instruccio generada pel sistema de control
    virtual uint32 seguentInstruccio(uint32 * fenotip)=0;
    // Representa al CMissatger l'estructura de control (Estat actual?)
    virtual void dump(CMissatger & msg)=0;
// Factoria
public:
    // retorna el nombre de tipus de sistemes de control implementats
    static uint32 Nombre(void);
    // retorna un sistema de control de tipus 'n'
    static CControlOrganisme * Crea(uint32 n);
    // retorna un sistema de control del tipus identificat per 'tipus'
```

```
    static CControlOrganisme * Crea(string tipus);
// Proves
public:
    // No es pot fer virtual pura pero caldria fer-la en totes les subclasses
    static void ProvaClasse();
};
```

Per crear un nou sistema de control és convenient seguir les següents passes:

1. Definir una classe derivada de `CControlOrganisme`, que anomenarem per entendre'ns `CNouControl`.

2. Definir per a `CNouControl`, les tres operacions bàsiques:

   - `init`: Construeix l'estructura interna del sistema de control a partir d'un cariotip.
   - `seguentInstruccio`: Retorna una instrucció a partir del contingut dels registres fenotípics que rep com a paràmetre.
   - `dump`: Visualtiza el contingut del sistema de control a un dispositiu de sortida.

3. Definir un mètode estàtic anomenat `CNouControl::ProvaClasse` amb un conjunt consistent de proves de la nova classe.

4. Adaptar la classe factoria per que sapiga generar el nou tipus de control.

   (a) Incrementar el número retornat per `CControlOrganisme::Nombre`

   (b) Afegir a les funcions `CControlOrganisme::Crea` el codi necessari per crear també controls del nou tipus.

(c) Provar els mètodes factoria fent servir de el mètode de proves de la classe abstracta.

15.8 Modificar del sistema de control actual

15.9 Afegir operadors genètics

15.10 Adaptar el model d'herència

15.11 Eines d'anàlisi

15.12 Arquitectura de la interfície d'usuari

15.13 Afegir nous dispositius de visualització

Capítol 16

Conclusions i línies de futur

16.1 Conclusions

En aquest apartat de la memòria es recullen les conclusions a les que s'han arribat en aquest projecte.

Primer, faig una valoració del que ha estat el desenvolupament de l'eina.

Seguidament, comentaré els resultats que han donant en les experiències els mecanismes d'expressió gènica i el model implementat.

Finalment, s'inclou com un resum dels seus costos econòmics.

16.1.1 Conclusions sobre l'eina implementada

Metodologia de treball

La metodologia d'implementació i les estratègies de disseny han ajudat molt a fer més efectiu el desenvolupament de l'eina.

La interfície

Ha estat un gran inconvenient el fet de presentar una primera interfície portable en consola de text perquè limita molt la quantitat i la qualitat de la informació representable i també resta interactivitat amb l'usuari.

Aquests dos inconvenients s'han compensat, per un costat, amb l'incorporació de seqüències d'escapament ANSI per augmentar la capacitat de representació del terminal, i, per un altre, la possibilitat de carregar fitxers de configuració en calent amb comandes ràpides del teclat.

Tot i així, segueix sent interesant la implementació d'una interfície gràfica, front-ends o altres eines complementàries per crear de forma més interactiva el que ara són els fitxers de configuració o modificar la configuració i el mateix sistema directament.

Possibilitats de configuració

El model implementat és altament parametritzable. Fins i tot, la majoria de paràmetres es poden canviar en mig de la simulació.

És molt destacable en aquest sentit el paper dels agents ambientals. Només amb els agents ambientals implementats en aquest projecte, es poden construir escenaris molt diversos per als organismes. A més, com la programació de nous tipus d'agents ambientals és molt senzilla i està documentada al detall, aquests nous agents poden oferir noves possibilitats de configuració que no hagin estat incloses amb els ja existents.

Components intercanviables

Les interfícies entre els diferents mòduls del nucli implementat, en ser tan estretes i genèriques, permeten no només millorar la testabilitat dels mòduls i encapsular la implementació. També permeten substituir de forma molt simple un mòdul per un altre.

Alguns mòduls anàlegs, fins i tot poden conviure en el mateix sistema. És el cas dels organismes i dels sistemes de control dels organismes. Aquests dos casos, amplien les opcions d'experimentació en el futur, per exemple es podria fer una comparació en competència entre organismes amb diferents elements de control o sistemes metabòlics.

Altres grups de mòduls com els agents ambientals, els operadors de mutació, i les vistes, directament fan servir aquesta característica per complir amb la seva funcionalitat.

Utilitat de l'eina

Donat que, en el Departament d'Informàtica d'Enginyeria la Salle, pot ser no es treballa prou sovint en temes de vida artificial, crec que l'existència d'una eina d'aquest estil, que té prou generalitat com per ser transportada a un conjunt molt ample de problemes, pot contribuir a l'increment dels treballs en aquesta àrea.

16.1.2 Conclusions sobre el model i les experiències realitzades

En quant al model concret d'organisme implementat, cal dir que ha donat proves de ser suficientment flexible al llarg de l'evolució.

En interval de temps entre 10 i 12 hores d'execució s'han trobat alguns comportaments complexos o emergents. Exemples són:

- En zones on hi ha molt diferencial de menjar que atreien als organismes del voltant, s'han observat conductes de depredació que seguien una estrategia 'de trampers'. És a dir, quedar-se quiet on hi ha el menjar i esperar a que arribin les preses.
- En zones on hi ha aliments agrupats en petis montículs s'han observat patrons complexos de cerca de nutrients al medi que prioritzen els pròxims, però, tenint sempre l'opció de cercar-los a distància fent dues cerques seguides sobre el mateix registre amb diferent radi.
- A zones amb suficient menjar uniformement distribuït s'observen comportaments per mantenir la distància amb els altres organismes, ja sigui per evitar competència, o per evitar ser depredat.
- En casos semblants a l'anterior, però, quan la suficiència de nutrients no era una situació perpètua, a vegades havien sorgit organismes que subordinaven el fet de mantenir-se equidistant al fet de trobar-se a una zona amb suficient menjar.
- Altres...

Al començament ha costat molt trobar conductes purament depredadores, però han co-

mençat a aparèixer així com hem posat els incentius energètics i configurant un medi més variable.

Per que els organismes desenvolupin comportaments complexos, cal que els organismes tinguin una vida mitjanament llarga. Si és possible reproduir-se de forma temprana, la vida dels organismes es fa molt curta i en conseqüència simple. Dificultar la reproducció temprana sembla ser la solució.

Primer, vaig intentar compensar-ho posant costos de reproducció molt grans. Vaig observar que funcionava, els organismes acabaven tenint una vida més llarga que donava peu a comportaments més complexos. El problema és que costava molt que els organismes comencessin a reproduir-se de forma correcta, doncs i ho provaven a fer de forma incorrecta morien doncs pagaven igual l'energia.

La conclusió a la que vaig arribat en aquest aspecte és que, més que fer pagar-ho amb energia, que evidentment caldria posar requisits energia però no fer-los pagar.

Per acabar amb els comentaris sobre el model, caldria dir que el model original, preveia comportaments sexuals no lligats directament a la reproducció (intercanvi de fragments cromosòmics com fan els bacteris). No es va fer perquè el model ja era prou complex com per afegir un element més però, crec que hagués millorat molt la capacitat d'adaptació del sistema.

També cal comentar que els significat de les zones operadores és una mica pobre i que generalment tendeixen a fer una funció probabilística més que de dependència.

16.1.3 Estudi econòmic

De cara a evaluar el temps invertit en la realització del projecte, el dividirem en diverses parts. En aquesta taula es representa en hores.

Part	1998.05 - 1999.06	1999.07 - 2000.01	Total
Documentació	120	10	130
Eines i entorn	35	8	38
Disseny	50	5	55
Implementació	110	490	600
Proves	30	10	40
Experimentació	0	80	80
Memòria	25	150	175
Total	370	753	1123

Taula 16.1: Hores destinades al projecte

La documentació del projecte va començar al Maig de 1998. El temps invertit en el projecte en cada part fins al Juny de 1999 és estimat. Des del Juliol de 1999 fins el Gener de 2000 el temps d'implementació, proves, experimentació i memòria està recollit aproximadament, per una eina de Logging.

16.2 Línies de futur

16.2.1 Experimentació amb el model presentat

La principal via de futur d'aquest projecte és la seva aplicació. Això vol dir plantejar experiments que es puguin realitzar sobre aquest sistema.

Com a exemple tenim la bateria d'experiments que s'han anat fent a la Universitat de San Diego sobre el sistema LEE.

16.2.2 Sistema de control

Es pot adaptar el sistema d'una forma molt directa a altres models d'organisme. Per exemple, es podria implementar fàcilment organismes cognitius basats en diferents tipus de xarxes neuronals, de forma similar a com s'ha fet als sistemes LEE, o en sistemes classificadors, entre d'altres.

Dels resultats d'aquesta primera experiència hem resaltat que potser es milloraria el sistema si s'introdueixen o es modifiquen alguns dels seus elements. Uns exemples serien:

Millors zones operadores: El significat que he donat a les zones operadores és un xic pobre. La majoria de casos adquireixen més un significat de probabilitat que no pas de dependència. Si s'enriquissin una mica més, pot ser donaria peu a comportaments molt més elaborats.

Instruccions de test i predicats: Pot ser de cara a millorar el significat de les zones operadores, convindria implementar instruccions de test que modifiquessin bits de predicat, que al mateix temps es facin servir per validar o no una zona operadora.

16.2.3 Sistema genètic

També és possible afegir altres elements interesants a la part genètica.

Operadors de mutació: Els que hi ha implementats potser no són del tot els idonis per un problema donat.

Evolució dels operadors de mutació Donada la hetereogeneitat en la distribució dels gens als cromosòmes, pot ser és útil que el tipus d'operador de mutació, o la probabilitat de mutar amb un o altre, també evolucioni amb el mateix cromosoma.

Sexualitat: Cal recordar que el model presentat no considera sexualitat, la qual cosa, no permet gaire variabilitat genètica. És trivial introduir mecanismes sexuals no lligats a la reproducció com els dels organismes unicel·lulars. Només caldria considerar l

16.2.4 Metabolisme

Les reaccions metabòliques s'han implementat d'una forma molt simple si tenim en compte les possibilitats de configuració que donen els sistemes LEE. Els sistemes LEE permeten establir en una taula de reaccions binàries on cada casella conté una llista de productes i el balanç d'energia per a cada dos possibles reactius.

16.2.5 Interfícies

En aquest projecte, de cara a mantenir la portabilitat, s'ha volgut mantenir una interfície de tipus terminal de text. Malgrat tot, aquesta interfície probablement no li serà prou amigable a un usuari final, a més, la interfície en mode text presenta moltes limitacions en la presentació de gràfics i, sobretot, en la quantitat de dades representables i la seva llegibilitat.

Així doncs, es podria implementar una interfície gràfica que facilités la feina als usuaris del sistema.

Aquesta tasca ve facilitada pel fet de que, durant l'elaboració del nucli, s'ha tingut molt present el paradigma model-vista-controlador:

El nucli és molt independent de la interfície. Fins i tot les traces i els missatges de debug que han d'estar implementats a dintre del nucli, estan preparats per no necessitar d'una plataforma concreta. Fan servir una classe adaptadora que pot visualitzar-les en un terminal, en una finestra de la llibreria Curses, en MessageBoxes o controls d'edició de MS-Windows o potencialment d'X-Windows.

Els pocs objectes visuals implementats en mode text (mapes, gràfics comparatius i gràfics d'evolució temporal) ofereixen un protocol d'accés que podria implementar a sota elements gràfics d'alta resolució.

16.2.6 Eines adicionals

A mida que compliquem el problema, les modificacions dels fitxers de configuració dels agents ambientals es tornen molt feixugues. De forma no massa complicada es pot reutilitzar el codi del nucli per fer una petita eina que permeti editar l'estructura arboriforme dels agents.

Una altra eina, podria ser una que ens permetés editar els individus o el seu material genètic. Editar els organismes d'una forma fàcil facilitaria els experiments dels biòlegs i obtenir un organisme primigeni eficient per estalviar temps d'evolució. Proposant com a cultiu primigeni les cepes dominants de diversos experiments editades convenientment per falcilitar la variabilitat genètica.

16.2.7 Funcionalitats

Donat que les simulacions s'allarguen molt de temps convindria implementar mecanismes per fer persistent el biosistema un cop es finalitza l'execució de l'eina i recuperar l'estat del biosistema en una execució posterior.

També seria interesant una funcionalitat que permetés importar i exportar organismes.

16.2.8 Optimitzacions

Hi ha punts del sistema dels qual encara es pot optimitzar el seu comportament:

Per exemple, es podria fer compartir a tots els organismes amb el mateix material genètic un punter a la mateixa estructura de dades. La millora que representa això en temps i en memòria tot i que és molt clara amb mutació o intercanvis sexuals no reproductius, deixaria de ser útil amb intercanvis sexuals lligats a la reproducció donat que la variabilitat genètica seria molt alta.

Apèndix A

Configuració

Existeixen tres arxius de configuració:

bioscena.ini	Paràmetres del biosistema
agents.ini	Agents ambientals
opcodes.ini	Correspondència entre codis i intruccions

A continuació s'expliquen en detall.

A.1 Arxiu `bioscena.ini`

Aquest arxiu defineix una sèrie de paràmetres numèrics que es fan servir per configurar el biosistema. Consta d'un seguit de línies on cadascuna defineix un paràmetre i tè una estructura semblant a aquesta:

```
* Apartat/SubApartat/../NomDelParàmetre 4
```

A continuació s'expliquen els paràmetres més importants. Entre parèntesis es posa el valor que l'eina adopta per omisió.

Biosistema/Energia/FixeInstruccio (1) Cost que, com a mínim, té una instrucció pel fet d'executar-se.

Biosistema/Energia/AdicionalInutil (1) Cost adicional que té una instrucció motora, si falla la seva execució.

Biosistema/Energia/Engolir (6) Guany que rep un organisme pel fet d'engolir un nutrient.

Biosistema/Energia/Excretar (3) Cost adicional d'excretar un nutrient

Biosistema/Energia/Extraccio (6) Guany per cada nutrient extret per agressio a una victima (i cost per la victima)

Biosistema/Energia/InicialExpontani (100) Energia amb la que comença un organisme aleatori

Biosistema/Energia/Mitosi/Cedida (15) Energia que es cedeix al fill en la mitosi

Biosistema/Energia/Mitosi/Factor (2) Factor pel que es multiplica l'energia que es cedeix al fill per obtindre el cost de la Mitosi

Biosistema/Energia/Mitosi/Minima (20) Mínima energia requerida per entrar a fer la mitosi (no es consumeix)

Biosistema/Energia/Mitosi/CostNoMinima (2) Cost d'haver entrat a fer la mitosi sense l'energia mínima

Biosistema/Energia/Moviment/Fixe (0) Cost adicional per instruccions de moviment

Biosistema/Energia/Moviment/UnitatDeCarrega (16) Nombre de nutrients que calen per omplir una unitat de carrega.

Biosistema/Energia/Moviment/CostUnitatDeCarrega (16) Increment per cada unitat de carrega en el cost energetic de moure's.

Biosistema/Metabolisme/BitsSignificatius (7) Màscara amb els bits significatius de les claus dels nutrients

Biosistema/OpCodes/BitsOperacio (5) El bits del bytecode que són significatius

Biosistema/Quantum/Maxim (4) Màxim nombre d'instruccions que s'executen de cop (El primer quantum que s'acabi mana)

Biosistema/Quantum/Utils (2) Màxim nombre d'instruccions útils que s'executen de cop (El primer quantum que s'acabi mana)

Biotop/CasellesAltitud (30) Posicions que tè el biòtop d'altura

Biotop/CasellesAmplitud (30) Posicions que tè el biòtop d'amplitud

Biotop/Substrat/MaximInicial (7) Maxims nutrients que pot contenir una sola posició del biòtop

Biotop/Substrat/MolleculesInicials/Numero (2) Nombre de nutrients amb els que s'omple cada casella del biòtop en iniciar l'execució

Biotop/Substrat/MolleculesInicials/Element (0) Element base que es fa servir per omplir cada casella del biòtop en iniciar l'execució

Biotop/Substrat/MolleculesInicials/Tolerancia (3) Tolerancia sobre l'element base per omplir cada casella del biòtop en iniciar l'execució

Comunitat/TamanyRegeneracio (27) Tamany de la població per sota del qual el biosistema sempre genera organismes aleatoris.

Comunitat/ProbabilitatGeneracioExpontanea/Encerts (1) Probabilitat de que es generi un organisme aleatori independentment del tamany de regeneració (Numerador)

Comunitat/ProbabilitatGeneracioExpontanea/Mostra (50) Probabilitat de que es generi un organisme aleatori independentment del tamany de regeneració (Denominador)

Organisme/Cromosoma/LongitudMaxima (10) Nombre màxim de codons d'un cromosoma d'un organismes aleatori

Organisme/Cromosoma/LongitudMinima (1) Nombre mínim de codons d'un cromosoma d'un organismes aleatori

Organisme/Cariotip/LongitudMaxima (10) Nombre màxim de cromosomes d'un organismes aleatori

Organisme/Cariotip/LongitudMinima (3) Nombre mínim de cromosomes d'un organismes aleatori

Organisme/Cariotip/PenalitzacioLlarg/BitsTamanyMaxim (8) Bits que es shifta a la dreta el tamany en codons del cariotip per penalitzar els cariotips massa llargs

Organisme/Cariotip/PenalitzacioLlarg/Factor (6) Factor per el que es multiplica el tamany del cariotip shiftat per obtindre el cost adicional pels cariotips massa llargs

Organisme/Energia/CaducitatCompartiments (20) El nombre d'instruccions que triga en caducar el compartiment d'energia més vell.

Organisme/Energia/Compartiments (8) Nombre de compartiments d'energia que té l'organisme.

Organisme/Fenotip/Longitud (16) Nombre de registres que formen el fenotip.

Organisme/Genotip/Intro/Mascara 0 La màscara que determina si un codó és un intró

Organisme/Genotip/Promotor/Mascara 1048576 La màscara que determina si un codó és un promotor

Organisme/Genotip/Terminador/Mascara 1048576 La màscara que determina si un codó és un terminador

Organisme/Genotip/Traduibilitat/Intents 5 El nombre de gens que es miren si són traduïbles abans de refusar traduir-ne cap.

Organisme/Genotip/ZonaOperadora/Mascara (65536) La màscara que determina si un codó pertany encara a la zona operadora

Organisme/Pap/Capacitat (10) Nombre de nutrients que pot contenir l'organismes (Un valor de zero vol dir que no es limita)

Organisme/ProbabilitatMutacio/Encerts (1) Determina la probabilitat de que es doni una mutació (Numerador).

Organisme/ProbabilitatMutacio/Mostra (15) Determina la probabilitat de que es doni una mutació (Denominador).

Sensor/Intern/Intents (10) Les vegades que un organismes prova de trobar un nutrient en el pap quan vol detectar-ho.

Sensor/Presencia/Intents (30) El nombre de posicions en les que es prova de mirar si hi ha algú que compleix les condicions de presència.

Sensor/Quimic/Intents (10) El nombre de posicions en les que es prova de mirar si hi ha nutrients amb les condicions especificades.

A.2 Arxiu `opcodes.ini`

Aquest arxiu simplement configura la correspondència entre els opcodes i les instruccions que s'executen.

Amb la variable `Biosistema/OpCodes/BitsOperacio` de l'arxiu `bioscena.ini` es pot limitar els bits útils d'aquest byte escollint els n menys significatius. Així no cal generar les 256 entrades.

El següent arxiu serviria per a 5 bits significatius.

```
* 00 NoOperacio
* 01 Mitosi
* 02 Anabol
* 03 Engoleix
* 04 Excreta
* 05 Ataca
* 06 Avanca
* 07 SensorI
```

```
* 08 SensorP
* 09 SensorQ
* 0A Random
* 0B Carrega
* 0C And
* 0D Xor
* 0E Oposa
* 0F ShiftR
```

El números que es fan servir per l'opcode estan codificats en hexadecimal. Cada línia ha d'anar precedida d'un signe `*` i separant els tres elements de cada línia, va un espai o tabulador.

La taula A.1 es citen tots els mnemònics implementats:

Instruccions Motores

NoOperacio	Mitosi	Avanca	Catabol
Engoleix	Excreta	Ataca	Anabol

Instruccions Fenotip

And	Or	Xor	Carrega
Copia	Not	Oposa	Random
ShiftR	ShiftL		

Instruccions Sensorials

SensorI	SensorP	SensorQ

Taula A.1: Mnemònics implementats

Si cap opcode queda està sense assignar-li un mnemònic, es donarà un missatge d'advertiment i se li asignara `NoOperacio`.

A.3 Arxiu `agents.ini`

A.3.1 Estructura

Els arxius de configuració d'agents ténen una estructura molt simple: Primer van unes linies de text que determinen, per cada agent, el seu nom i el seu tipus. Un cop definits els noms i els tipus, es configuren els paràmetres per a cada agent.

La definició dels noms i els tipus es fa amb una línia per cada agent sent la primera línia la que defineix l'agent que està en l'arrel de la estructura de subordinació. A cada línia es posa, per ordre i *separats per espais* un signe asterisc, el nom i el tipus.

Quan es carrega un arxiu de configuració d'agents, si és possible a cada agent se li dóna el nom amb el que apareix a l'arxiu, però això no és posible si ja existeix un amb el mateix nom. En aquest cas, se li dóna un nom per defecte i s'hi tradueixen totes les posteriors referències al nom antic.

Els noms poden contenir qualsevol caracter que no es consideri un espai a C (espais, tabuladors, retorns...).

El tipus s'especifica amb un identificador propi de cada tipus d'agent. Dins d'un arxiu de configuració d'agents, es reconeixen els següents tipus:

Es fan servir identificadors jerarquics molt semblants als que es fan servir a UNIX per identificar els directoris. La jerarquia de noms el que especifica aquí és una jerarquia de tipus i subtipus que és molt útil de cara a restringir els agents que es poden posar com a dependències. Per exemple, una dependència requereix un *Agent/Posicionador*, aquest lloc

Nom de tipus a la memòria	**Nom del tipus a un fitxer de configuració**
Agent Subordinador Multiple	Agent/Multiple
Agent Subordinador Temporitzador	Agent/Multiple/Temporitzador
Agent Subordinador Iterador	Agent/Multiple/Iterador
Agent Subordinador Aleaturitzador	Agent/Multiple/Aleaturitzador
Posicionador Fixe	Agent/Posicionador
Posicionador Aleatori	Agent/Posicionador/Aleatori
Posicionador Zonal	Agent/Posicionador/Zonal
Posicionador Direccional (Itinerari)	Agent/Posicionador/Direccional
Direccionador Fixe	Agent/Direccionador
Direccionador Aleatori	Agent/Direccionador/Aleatori
Actuador Nutridor	Agent/Actuador/Nutridor
Actuador Desnutridor	Agent/Actuador/Nutridor/Invers

Taula A.2: Tipus d'agents implementats al projecte i identificadors associats

el pot ocupar tant un *Agent/Posicionador/Direccional* com un *Agent/Posicionador/Zonal*.

Una definició de noms i tipus podria quedar com segueix:

```
* Agent_0000 Agent/Multiple
* Agent_0001 Agent/Multiple/Temporitzador
* Agent_0002 Agent/Direccionador/Aleatori
* Agent_0003 Agent/Posicionador/Direccional
* Agent_0004 Agent/Multiple/Iterador
* Agent_0005 Agent/Actuador/Nutridor
```

Aquí, *Agent_0000* seria l'agent arrel. Perqué està declarat en primer lloc.

Un cop definits els noms i els tipus dels agents, cal configurar els seus paràmetres. Per configurar un agent primer cal posar una línia amb el signe + i el nom de l'agent separats per un espai, i, després, tot un seguit de línies de configuració de paràmetres. Les linies de configuració de paràmetres comencen amb un signe menys i el nom del paràmetre i es segueix amb els valors que necessita el paràmetre per configurar-se, tot separat per espais.

Com veurem al següent exemple, és normal que un paràmetre s'especifiqui amb diversos valors separats per espais. La posició dels valors acostuma a ser significativa o sigui que és important mantenir l'ordre.

Per configurar un Nutridor es faria de la següent forma

```
+ Agent_0004
- Posicionador Agent_0003
- Composicio 31 0
```

Quan un paràmetre necessita com a valor un altre agent, fa servir els seu nom com a referència.

Al següent apartat, es detalla els paràmetres que controlen cada tipus d'agent.

A.3.2 Paràmetres configurables per a cada tipus d'agent

El que segueix és una especificació de com s'especifiquen al fitxer de configuració els paràmetres dels tipus d'agent implementats. Per fer-ho fem servir la següent estructura:

Per a cada paràmetre de cada tipus d'agent es fa una petita explicació i es detallen, ordenats tal qual han d'aparèixer, els valors que el defineixen.

Els valors dels paràmetres es detallen posant un tipus de dada, dos punts, una petita explicació del valor i, entre parèntesis, les restriccions que s'hi apliquen.

Als agents implementats, els tipus de dada possibles pels valors són:

- **agent:** Nom d'un agent especificat a la definició de noms i tipus

- **uint32:** Senser sense signe codificable en 32 bits i expressat en base decimal
- **id(Alternativa1/Alternativa2...):** Un dels identificadors posats com a alternativa

Després dels paràmetres de cada tipus hi ha un exemple de com quedarien les línies de configuració.

A la secció 8 hi ha una explicació menys esquemàtica de cada tipus d'agent.

MultiAgent (Agent/Multiple)

Accio: Determina un agent subordinat. Es repeteix tantes vegades com subordinats tingui.

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

Exemple de configuració

```
+ AgentMultiple1:
- Accio Posicionador1
- Accio Posicionador2
- Accio Direccionador1
```

Temporitzador (Agent/Multiple/Temporitzador)

Accio: Determina un agent subordinat. Es repeteix tantes vegades com subordinats tingui.

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

CicleActiu: Determina quan triguen els períodes de temps actius

- uint32: període mínim
- uint32: número de daus
- uint32: magnitud dels daus (Van de zero a la magnitud)

CicleInactiu: Determina quan triguen els períodes de temps inactius

- uint32: període mínim
- uint32: número de daus
- uint32: magnitud dels daus (Van de zero a la magnitud)

AntiAccio: Agent subordinat especial que s'acciona en el cicle inactiu (Només un per temporitzador i es opcional)

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

CicleActual: Valors del temporitzador quan es reemprengui la marxa

- id(Actiu/Inactiu): cicle actiu o inactiu
- uint32: període restant del cicle actual

Exemple de configuració

```
+ Temporitzador1
- Accio Posicionador3
- CicleActiu 34 2 5
- CicleInactiu 2 4 4
- CicleActual 3 Inactiu
```

Iterador (Agent/Multiple/Iterador)

Accio: Determina un agent subordinat. Es repeteix tantes vegades com subordinats tingui.

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

Iteracions: Determina quantes vegades es repiteixen els subordinats

- uint32: iteracions mínimes
- uint32: número de daus
- uint32: magnitud dels daus (Van de zero a la magnitud)

PreAccio: Agent subordinat especial que s'executa un sol cop abans de tot (Només un per iterador i és opcional)

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

PostAccio: Agent subordinat especial que s'executa un sol cop despres de tot (Només un per iterador i és opcional)

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

Exemple de configuració

```
+ Iterador3
- Accio Posicionador4
- Accio Actuador2
- Iteracions 20 3 6
```

Aleaturitzador (Agent/Multiple/Aleaturitzador)

Accio: Determina un agent subordinat. Es repeteix tantes vegades com subordinats tingui.

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

Probabilitat: La que hi ha d'accionar els subordinats

- uint32: número d'encerts que segons la probabilitat tenderien a donar-se en la mostra

- uint32: número de intents o mostra

ReAccio: Agent subordinat especial que s'acciona si no es dóna la probabilitat (Només un per temporitzador i és opcional)

- agent: agent que es subordina (No ha de ser subordinat de cap altre)

Exemple de configuració

```
+ Aleaturitzador1
- Accio Posicionador3
- Probabilitat 20 100
```

Posicionador Fixe (Agent/Posicionador)

Posicio: Posició inicial

- uint32: valor de la posició (Ha d'existir a la topologia)

Exemple de configuració

```
+ Posicionador1
- Posicio 12
```

Posicionador Aleatori (Agent/Posicionador/Aleatori)

Posicio: Posició inicial

- uint32: valor de la posició (Ha d'existir a la topologia)

Exemple de configuració

```
+ Posicionador2
- Posicio 23
```

PosicionadorSequencial (Agent/Posicionador/Sequencial)

Posicio: Posició inicial

- uint32: valor de la posició (Ha d'existir a la topologia)

Sequencia : Determina una posició de la seqüència. Es repeteix tantes vegades com calgui.

- uint32: valor de la posició (Ha d'existir a la topologia)

SequenciaActual :

- uint32: el número de seqüència de la següent posició (Si es passa es pren l'ultim)

Exemple de configuració

```
+ Posicionador3
- Posicio 23
- Sequencia 27
- Sequencia 50
- Sequencia 402
- SequenciaActual 2
```

PosicionadorZonal (Agent/Posicionador/Zonal)

Posicio: Posicio inicial

- uint32: valor de la posició (Ha d'existir a la topologia)

Posicionador: Dona la posició central de la zona

- agent: agent posicionador (dependència)

Radi: Nombre de desplacaments que pot fer la posició entorn al centre

- uint32: valor del radi

Exemple de configuració

```
+ Posicionador4
- Posicio 23
- Posicionador Posicionador1
- Radi 3
```

Itinerari (Agent/Posicionador/Direccional)

Posicio: Posició inicial

- uint32: valor de la posició (Ha d'existir a la topologia)

Direccionador: Dona la direcció del desplacament

- agent: agent direccionador (dependència)

Radi: Nombre de desplacaments que pot fer la posició respecte a la posició anterior

- uint32: valor del radi

Exemple de configuració

```
+ Posicionador5
- Posicio 23
- Direccionador Direccionador3
- Radi 1
```

Direccionador (Agent/Direccionador)

Direccio: Direcció inicial

- uint32: valor de la direcció

Exemple de configuració

```
+ Direccionador1
- Direccio 876342
```

DireccionadorAleatori (Agent/Direccionador/Aleatori)

Direccio: Direcció inicial

- uint32: valor de la direcció

Exemple de configuració

```
+ Direccionador2
- Direccio 23442684
```

DireccionadorSequencial (Agent/Direccionador/Sequencial)

Direccio: Direcció inicial

- uint32: valor de la direcció

Sequencia: Determina una direcció de la seqüència. Es repeteix tantes vegades com calgui.

- uint32: valor de la direcció

SequenciaActual: Determina el punt actual de la seqüència

- uint32: el número de seqüència de la següent direcció (Si es passa es pren l'ultim)

Exemple de configuració

```
+ Direccionador3
- Direccio 23
- Sequencia 27
- Sequencia 50
- Sequencia 402
- SequenciaActual 2
```

Nutridor (Agent/Actuador/Nutridor)

Posicionador: Dona la posició on s'actua

- agent: Agent posicionador (dependència)

Composicio: Determina els elements que es depositen

- uint32: element basic

- uint32: variabilitat, a 1 els bits que poden variar

Exemple de configuració

```
+ Actuador1
- Posicionador Posicionador4
- Composicio 13152450903 0
```

Desnutridor (Agent/Actuador/Nutridor/Invers)

Posicionador: Dona la posició on s'actua

- agent: Agent posicionador (dependència)

Composicio: Determina els elements que s'eliminen

- uint32: element basic

- uint32: tolerancia, a 1 els bits que no importa que coincideixin

Exemple de configuració

```
+ Actuador2
- Posicionador Posicionador3
- Composicio 8943742645 768764258
```

Bibliografia

[AML86] Francisco J. Segovia Pèrez Angel Morales Lozano. *Programacion orientada a objetos, aplicaciones con Smalltalk*. Paraninfo, Madrid, 1986.

[AMR97] Jamshid Ghaboussi Anne M. Raich. Autogenesis and redundancy in ga representation. In *Proceedings of the 1997 International Conference on Genetics Algorithms ICGA97*. Michigan State University, 1997.

[Dar59] Charles Darwin. *The Origin of the Especies*. Guttemberg Project, 1859.

[FH91] Thomas Back Frank Hoffmeister. Genetic algortihms and evolution strategies: Similarities and differences technical report sys-91/2. Technical report, Systems Analysis Research Group, LSXI, Dept. of Computer Science, 1991.

[Fra97] Wolfgang Banzhaf; Peter Nordin; Frank D. Francone. Why introns in genetic programming grow exponentially. In *Proceedings of the 1997 International Conference on Genetics Algorithms ICGA97*. Michigan State University, 1997.

[Gar99] David García Garzón. *Bioscena: Simulació d'un entorn biològic evolutiu amb interacció entre els organismes*. Enginyeria la Salle, 1999.

[ISO] *Working Paper for Draft Proposed International Standard for Information Systems - Programming Language C++*. ISO/ANSI.

[JCGC98] Grupo G9 Jose Carlos González Cristóbal. *Vida artificial: análisis y simulación*. PhD thesis, 1998.

[Kar97] Hillol Kargupta. Relation learning in gene expression: Introns, variable length representation, and all that. In *Proceedings of the 1997 International Conference on Genetics Algorithms ICGA97*. Michigan State University, 1997.

[Lam86] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley, 1986.

[Man] G. Mangiarotti. *Del gen al organismo. Biologia General*. Picin, Padova, Italia.

[May97] Helmut A. Mayer. ptga's, genetic algorithms using promoter/terminator sequences evolution of number, size and location of parameters and parts of representation. In *Proceedings of the 1997 International Conference on Genetics Algorithms ICGA97*. Michigan State University, 1997.

[MB93] Filippo Menezer and R. K. Beleg. Latent energy enviroments: A tool for artificial live simulations. Technical Report 93-301, Computer Science and Engineering Department, La Jolla, CA 92093-0114, 1993.

[MB94] Filippo Menezer and R. K. Beleg. Evolving sensors in environments of controled complexity. In R. Brooks and P. Maes, editors, *Artificial Live IV*, La Jolla, CA 92093-0114, 1994. MIT Press.

[MB95] Filippo Menezer and R. K. Beleg. Latent energy enviroments. Technical report, Computer Science and Engineering Department, La Jolla, CA 92093-0114, 1995.

[MB96] Filippo Menezer and R. K. Beleg. Complex environments to complex behaviors. *Adaptive Behavior*, 4(3/4):317–363, 1996.

[MBC95] Filippo Menezer, R. K. Beleg, and Federico Cecconi. Maturation and evolution of imitative learning in artificial organisms. Technical Report 506, Computer Science and Engineering Department, La Jolla, CA 92093-0114, 1995.

[Men94] Filippo Menezer. Changing latent energy enviroments: A case for evolution of plasticity. Technical Report 94-336, Computer Science and Engineering Department, La Jolla, CA 92093-0114, 1994.

[Ray93] Thomas S. Ray. Jugué a ser dios y creé vida en mi computadora. *Epistemología e Informática*, pages 257–267, 1993.

[Ser] Anselmo Pérez Serrada. *Una Introducción a la Computación Evolutiva*. PhD thesis.

[Ste] S C Stearns. *The evolution of Life Histories*. Oxford University Press, New York.

[Str] M. Strickberger. *Genética*. Ed. Omega, Barcelona, 3 edition.

[Str86] Bjarne Stroustrup. *El C++, lenguaje de programacion*. Addison-Wesley, 2 edition, 1986.

[Tod93] PM Todd. Artificial death. In *Second European Conference on Artificial Life*, Brussels, Belgium, May 1993.

[TS97] James A. Foster Terence Soule. Comments on the intron/exon distinction as it relates to the genetic programming and biology. In *Proceedings of the 1997*

International Conference on Genetics Algorithms ICGA97. Michigan State University, 1997.

Índex alfabètic