5860
SIMULACIÓ DEL COMPORTAMENT D'UNS INDIVIDUS EN UN ENTORN I DESENVOLUPAMENT D'UN VIDEOJOC EDUCATIU DE PROGRAMACIÓ

Memòria del projecte final de carrera corresponent als estudis d'Enginyería Superior en Informàtica presentat per Aniol Riu i dirigit per Toni Gurguí.

Bellaterra, setembre de 2014.

El firmant Antoni Gurguí, professor del Departament de Ciències de la Computació de la Universitat de Barcelona

CERTIFIQUEN:

Que la present memòria ha estat realitzada sota la seva direcció per Aniol Riu Torrens

Bellaterra, setembre de 2014

Firmat: Antoni Gurguí

ÍNDEX

ÍNDEX DE FIGURES

1 INTRODUCCIÓ

1.1 PRESENTACIÓ

A les pàgines següents es descriurà el procés d'implementació d'una aplicació capaç de simular i reproduir tridimensionalment el comportament d'uns individus en un entorn en base a un algorisme propi. A més, s'implementarà una interfície d'usuari que permetrà interactuar amb la simulació de forma còmoda i intuïtiva, modificar l'algorisme dels individus i extreure informació de l'entorn, i es dissenyaran uns nivells a mode de videojoc que introduiran l'usuari en el món de la programació.

La feina que s'ha dut a terme és complementària amb la del Nicolau Manubens, company de carrera, que partint de la mateixa base, el simulador del comportament d'uns individus en un entorn, ha desenvolupat les eines necessàries perquè aquests individus es puguin reproduir, mutar i evolucionar, i ha implementat la funcionalitat necessària per monitoritzar el procés i extreure'n resultats. Per tant, si bé aquesta feina no forma part del projecte que ens ocupa, en alguns punts les línies de treball s'entrellacen de tal manera que alguns dels aspectes desenvolupats en aquest projecte no tenen sentit sense l'altra part, sobretot pel que fa a la interfície gràfica. Per tant, en diversos punts de la memòria faré referència a mòduls funcionals no implementats per mi, però que donen sentit al projecte conjunt.

1.2 MOTIVACIÓ

Aquest projecte atén l'interès que se'm va despertar pel món de l'algorísmia des que vaig rebre les primeres nocions al respecte. Quin és l'algorisme que determina el comportament d'una formiga? I el d'un humà? Com s'ha generat? La resposta més versemblant és que per evolució. I si simuléssim un entorn on els algorismes poguessin evolucionar? Seria el cervell humà capaç de dissenyar millors algorismes que els generats per evolució? Per contestar aquesta pregunta, calia implementar un entorn on uns individus la conducta dels quals està determinada per un algorisme poguessin desenvolupar la seva activitat, reproduir-se, morir i, en definitiva, evolucionar, per una banda, i possibilitar que l'usuari pugui dissenyar els seus propis algorismes i assignar-los a aquests individus, per l'altra. Aquest projecte es centra en la segona part.

1.3 OBJECTIUS

1) Dur a terme la implementació d'una aplicació a partir d'eines molt bàsiques partint pràcticament de zero, sense fer servir llibreries externes, plataformes o abstraccions ja implementades.

2) Implementar una aplicació que sigui capaç de simular el comportament de diferents individus que actuen en un entorn en funció dels seus propis algorismes. Aquesta part constitueix el nucli de l'aplicació i es desenvoluparà en cooperació amb el Nicolau Manubens.

3) Aconseguir que la implementació sigui escalable i flexible, i que la majoria de paràmetres d'una simulació els pugui determinar l'usuari: entorn, número d'individus, etc.

4) Implementar una aplicació còmoda, visual i d'interacció intuïtiva. Per això es marquen els subobjectius següents:

4.1) Dissenyar i implementar una interfície amb la qual l'usuari interactuï i controli la simulació.

4.2) Representar l'entorn en 3D. Es tractarà d'una visualització que permeti una navegació intuïtiva per l'entorn i que compleixi finalitats funcionals i no estètiques.

5) Desenvolupar una aplicació que introdueixi l'usuari en el món de la programació i l'algorísmia. És a dir, plantejar una sèrie de reptes a l'usuari que s'hagin de solucionar a través del disseny d'algorismes. Per això es plantegen els subobjectius següents:

5.1) Dissenyar i implementar un editor d'algorismes que permeti a l'usuari crear i modificar algorismes en temps d'execució.

5.2) Dissenyar i implementar nivells de dificultat creixent, i la funcionalitat per poder gestionar els diferents nivells.

1.4 ESTAT DE L'ART

Pel que fa a la representació tridimensional d'un entorn i control d'un individu, l'estat de l'art l'establirien els productes de la indústria del videojoc. Tot i així, l'objectiu de la representació tridimensional d'aquest projecte no és dotar l'aplicació d'un gran realisme visual, sinó proporcionar a l'usuari una eina eficaç i còmoda per navegar per l'entorn, i per tant en cap cas pretenem arribar al nivell tècnic dels videojocs actuals.

Respecte a la interfície de disseny d'un algorisme, l'objectiu no és donar llibertat absoluta a l'usuari perquè escrigui el codi, sinó que es construeixi a partir de menús desplegables. En aquest sentit, una aplicació similar és *Scratch* [1], en què l'usuari construeix el codi a partir de peces predissenyades. Algunes d'aquestes peces tenen paràmetres que poden prendre determinats valors, que es seleccionen amb un desplegable en alguns casos, o que pot escriure directament l'usuari en d'altres.

Figura 1.1: Programació amb *Scratch*.

La sintaxi d'*Scratch* és similar a la de llenguatges com *C* o *Java*, sobretot pel que respecta a instruccions de control de flux (*If-Then, For, While*). La sintaxi del nostre projecte pretenem que sigui més simple per tal que l'usuari s'inicïi en el món de la programació sense sentir-se aclaparat pel gran ventall de possibilitats, seguint l'objectiu 5.

L'edició de codi es du a terme directament en el navegador.

Un altre projecte en què es fa un esforç important per tal que l'edició de codi sigui el més intuïtiva possible i es redueixi la possibilitat de cometre errors és *Alice* [2].

En aquest cas, l'aplicació permet crear clips de vídeo o videojocs en entorns tridimensionals.

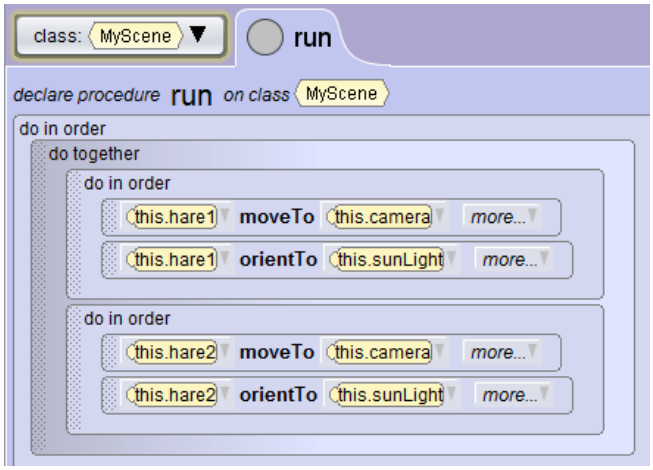

Figura 1.2: Programació amb *Alice*.

De nou, si bé s'acota l'àmbit d'actuació de l'usuari per tal d'orientar-lo en el disseny d'algorismes, aquest és massa ampli i no apte per a una persona que no hagi programat mai.

Microsoft Small Basic [3] és una altra eina orientada a introduir l'usuari en el món de la programació orientada a objectes. La instanciació i declaració d'objectes es fan en un sol pas, i un cop fet, automàticament es despleguen tots els mètodes de l'objecte en qüestió, d'entre els quals l'usuari n'haurà de triar un o esborrar l'objecte.

Figura 1.3: Logotip del programa *Microsoft Small Basic*.

Per últim, el llenguatge de programació *LOGO* ha evolucionat al llarg del temps per convertir-se en una eina pedagògica en què s'inicia a l'usuari en la programació a partir d'escriure ordres que executa una tortuga. Al igual que en el nostre projecte, la sintaxi és molt simple, tot i que *LOGO* no és un llenguatge de programació orientat a objectes, sinó que els mètodes invocats fan referència de forma implícita a una instància o a una altra (bàsicament, la tortuga o la pantalla). A més, a diferència del llenguatge que es vol implementar, els mètodes a Logo accepten arguments. *LOGO* és un llenguatge sobre el qual s'han desenvolupat diversos entorns de programació, actualment la majoria incrustats a pàgines Web [4].

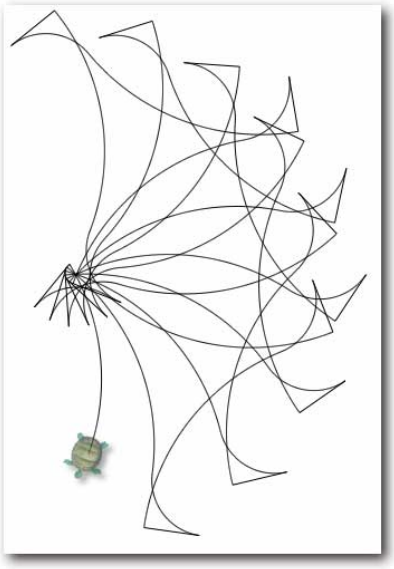

Figura 1.4: Dibuix fet amb el llenguatge *LOGO*.

Fins aquí, hem vist entorns de programació que compilen i executen el codi o l'interpreten per executar-lo en una altra plataforma.

Pel que respecta a l'edició de codi en temps d'execució, hem trobat alguns entorns de programació incrustats en el propi programa que es vol modificar. És el cas de *LEDA* [5], que implementa una classe intèrpret capaç de compilar i executar codi en temps d'execució en programes desenvolupats amb *Unity3D*. *LEDA* està pensat per a usuaris que ja sàpiguen programar.

Per altra banda, en referència a l'execució de diferents algorismes per a diferents individus, existeix una plataforma basada en *Java* per al disseny i execució de models multiagent, anomenada *Repast* [6].

Figura 1.5: Logotip de Repast

2 ANÀLISI DE VIABILITAT

2.1 VIABILITAT TÈCNICA

Un volum de feina important del projecte correspon a la definició i manipulació d'estructures de dades del nucli de l'aplicació. Són necessaris coneixements en estructures de dades i programació que s'han assolit, en gran mesura, durant la carrera.

Per al disseny d'una interfície, s'emprarà l'eina Qt, àmpliament utilitzada i de la qual es pot trobar molta informació, referències, tutorials i ajudes a internet. A més, ja s'ha utilitzat durant el transcurs de la carrera i per tant, no suposarà cap impediment tècnic.

2.2 VIABILITAT ECONÒMICA

El cost econòmic del projecte ve determinat pel cost de les llicències del software utilitzat i pel cost de les hores treballades.

Les eines i biblioteques necessàries per al desenvolupament que es fan servir per a la implementació de l'aplicació són gratuïtes, i en la majoria de casos disposen de llicències de software lliure que permeten la modificació del codi.

Pel que respecta al hardware, tractant-se d'una implementació de software, considerarem que té cost nul, i suposarem que ja disposem de la infraestructura necessària.

Per últim, cal valorar el cost associat a les hores de feina que requereix el desenvolupament del projecte. Aquestes hores es comptabilitzen a partir dels objectius i la planificació del projecte.

2.3 VIABILITAT LEGAL

La viabilitat legal ve determinada per les llicències del software utilitzat, que en aquest cas és lliure. Per tant, no hi ha cap impediment o obstacle legal per a la realització del projecte.

2.4 PLANIFICACIÓ TEMPORAL

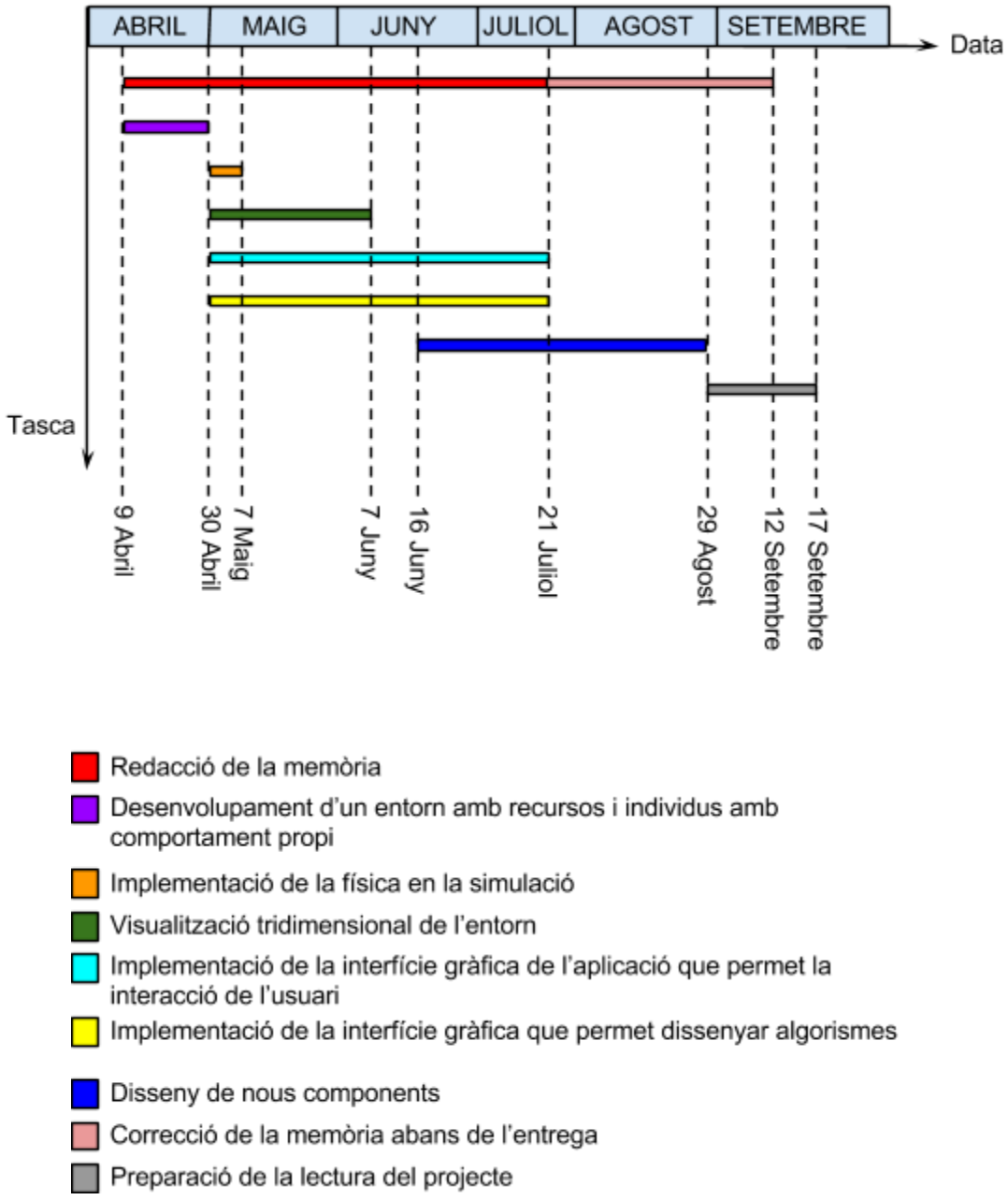

Figura 2.1: Planificació temporal

A continuació es detallen les hores dedicades a cada fase:

Concepte	**Hores**
Redacció de la memòria	80

Desenvolupament d'un entorn amb recursos i individus amb comportament propi	120
Implementació de la física en la simulació	2
Visualització tridimensional de l'entorn	40
Implementació de la interfície gràfica	140
Implementació de l'editor d'algorismes	60
Disseny de components	20
TOTAL	462

2.5 CÀLCUL DEL COST ECONÒMIC

Per al càlcul del pressupost es considerarà el sou d'un programador júnior d'una gran empresa tal com figura al Butlletí Oficial de l'Estat [7] que és de 12,9552 €/hora.

Tenint en compte que el projecte portarà unes 462 hores de feina, resolem que el pressupost ascendeix a 5.985,30 €.

Evidentment, aquest pressupost no s'ajusta a la realitat ja que les hores es dedicaran en concepte d'estudi i, per tant, no es cobraran. El cost real, doncs, serà de 0 €.

3 DISSENY

El nucli del programa, la implementació del qual és un dels objectius del projecte, és un simulador del comportament d'uns individus en un entorn en base a un algorisme propi. Aquesta part, es desenvoluparà en col·laboració amb el Nicolau Manubens. Sobre aquesta base, es vol dissenyar una interfície gràfica que permeti controlar els diferents paràmetres de la simulació i visualitzar-la, si així ho desitja l'usuari. A més, es dissenyaran uns nivells de joc i un editor d'algorismes i s'implementaran les funcionalitats necessàries per tal que es puguin afegir individus a l'entorn, eliminar-los i també modificar-ne el seu comportament. A la figura 3.1 es mostra un diagrama de classes amb els mòduls funcionals que s'han comentat.

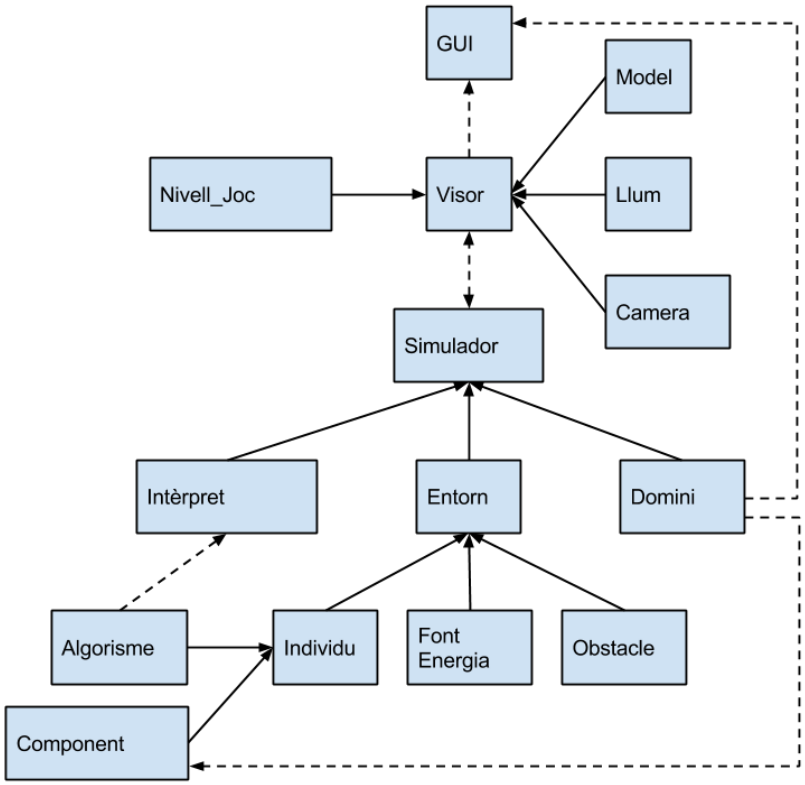

Figura 3.1: Diagrama de classes de l'aplicació.

Els següents apartats descriuen amb més detall les característiques de disseny de cadascun d'aquests mòduls.

3.1 ENTORN

L'entorn engloba l'espai on els individus poden dur a terme la seva activitat. Els individus es desplaçaran en un pla, i només en dues direccions ortogonals. És a dir, endavant, endarrere, esquerra i dreta. Els moviments dels individus seran discrets.

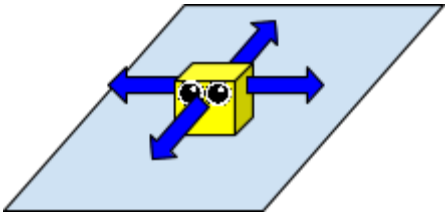

Figura 3.2: Moviment d'un individu.

A més, s'hi ubicaran obstacles que impediran el pas dels individus i també fonts d'energia, que seran esgotables; se'n poden esgotar i crear de noves.

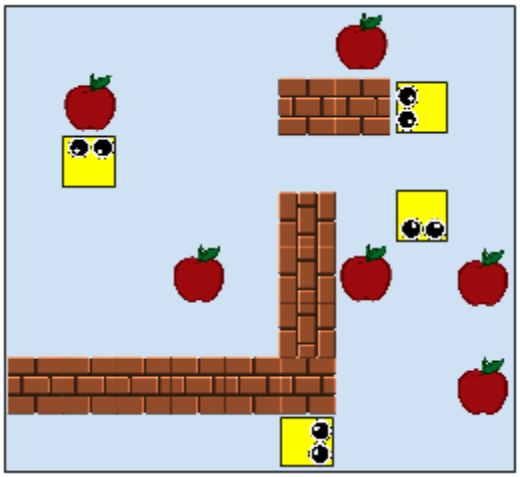

Figura 3.3: Exemple d'entorn.

Es vol que els entorns es puguin guardar en un fitxer extern codificat amb ASCII amb l'extensió *.ent*. La informació de l'entorn es codifica de la manera següent: dues barres baixes "__" significa una cel·la buida, les lletres "OB" significaran obstacle i les lletres "FO" font d'energia. Per exemple, les línies següents codifiquen l'escenari de la Figura 3.3, de mida 9 x 8 cel·les.

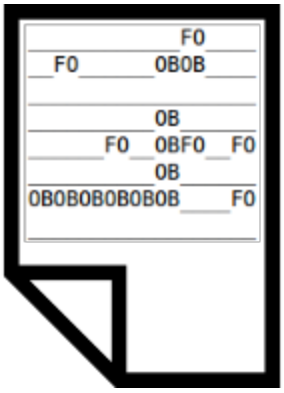

Figura 3.4: Fitxer d'entorn *.ent*.

D'aquesta manera l'usuari pot dissenyar entorns amb un editor de text i decidir en temps d'execució sobre quin entorn vol llançar la simulació.

3.2 INDIVIDU

L'individu és l'element bàsic de l'aplicació, i es caracteritza a dos nivells: aspecte (hardware) i comportament (software), és a dir, els components de què està construït l'individu i el seu algorisme.

3.2.1 COMPONENTS

Els components són les peces que formen els individus. Un component pot contenir altres components, i també mètodes. El conjunt de components dels quals pot estar format un individu s'anomena el domini de components. Tal i com s'indica a l'objectiu 3, es vol que l'aplicació sigui escalable i flexible, i aquí el disseny del domini juga un paper important. Es vol que el domini permeti crear individus amb estructures molt diverses però lògiques i jerarquitzades. En aquest sentit, no es vol que els components que configuren un individu siguin simplement una llista, sinó que tinguin una estructura de llista d'arbres. Per altra banda, es vol que aquesta estructura tingui una certa lògica. Vegem un exemple: suposem que tenim els components de la Figura 3.5.

Figura 3.5: Exemple de components.

Si no establim cap tipus de jerarquia entre ells, no hi ha cap diferència entre dos individus que tenen uns ulls, dues articulacions i tres cames. Si en canvi establim una jerarquia, podem obtenir individus que tot i tenir les mateixes peces, no tinguin el mateix aspecte, tal i com es mostra a les figures 3.6 i 3.7.

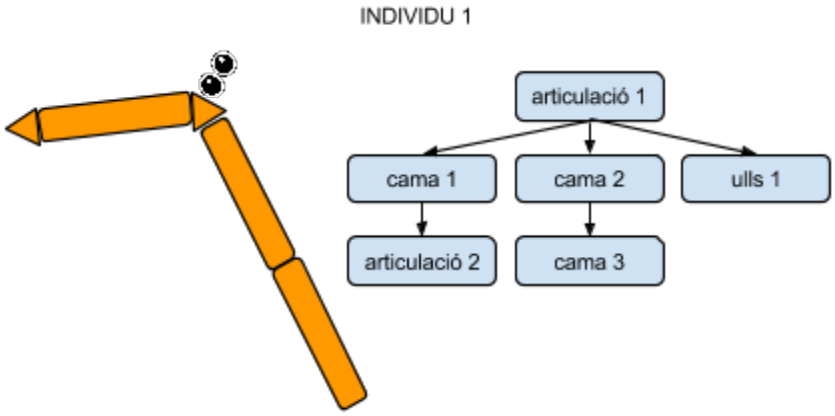

Figura 3.6: Organització dels components de l'Individu 1.

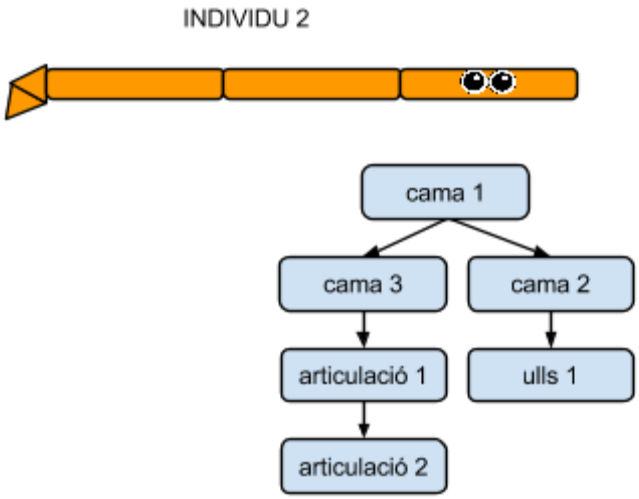

Figura 3.7: Organització dels comopnents de l'Indiviu 2.

Tot i així, es vol limitar quins components poden ser fills de quin altres components. En el cas anterior, per exemple, podríem no voler que un ulls puguin sortir directament d'una articulació, o que una articulació pugui estar annexada a una altra. El domini de components, doncs, ens indica de quins components pot estar format un individu, quins mètodes tenen aquests components, i quins poden estar annexionats a quins altres.

Aquesta estructuració dels components respon a l'objectiu 3. Tot i que l'estructura dels components no tindrà cap rellevància en aquest projecte, es vol que l'aplicació sigui ampliable en el futur per tal que es puguin desenvolupar els indivius no només a nivell de comportament sinó també d'aspecte. És a dir, que dos individus amb el mateix comportament puguin estar més o menys ben adaptats al medi en funció de la seva estructura.

Tal com s'ha comentat, els components poden contenir mètodes, que són les accions amb les quals un individu interactua amb l'entorn i també modifica alguns paràmetres propis. Aquests components poden ser de dos tipus:

- **Get**: Els mètodes *Get* configuren el conjunt de sensors de l'individu i són els mètodes que proporcionen inputs als sistema. Permeten a l'individu interpretar certs paràmetres de l'entorn o del propi individu. Quan s'invoquen, retornen un valor, que pot ser Cert o Fals.

- **Set**: Els mètodes *Set* permeten a l'individu alterar l'entorn o paràmetres del propi individu. Configuren el conjunt d'actuadors del sistema i són el mitjà amb el qual l'individu expressa els seus outputs. No retornen cap valor.

Per fer referència a un mètode d'un component, farem servir la nomenclatura següent:

nom_component_NX1.nom_component_NX2...nom_metode

Com hem vist, els components estan distribuïts en forma de llista d'arbres, i un component de tipus *CapsaDeSensors* pot tenir diversos fills anomenats *SensorDeProximitat*. Per distingir un fill d'un altre del mateix tipus, s'utilitza l'índex X1, X2, etc. Cal observar que els mètodes no tenen índex.

Per tal de simplificar la sintaxi, els mètodes no tindran arguments. Així, un component de memòria que gestiona un registre de 1 bit, no tindrà un mètode *SetValor(bool)*, sinó que implementarà els mètodes de tipus *Set* següents: *SetCert*, *SetFals*, *Negar*, a més dels mètodes de tipus *Get* següents: *EsCert*, *EsFals*, que retornarien el valor del registre.

Per exemple, suposem que tenim un component *CapsaDeSensors*, amb dos sensors de proximitat a dins anomenats *SensorProx_N0* i *SensorProx_N1*. Per tal d'accedir al valor que té un dels sensors en un moment donat, disposem del mètode *GET_DetectarPosicioOcupada*. Per tal de saber si hi ha un objecte davant de l'individu en la direcció del *SensorDeProximitat_N1* de la *CapsaDeSensors_N0*, utilitzarem la sintaxi següent:

```
CapsaDeSensors_N0.SensorProx_N1.DetectarPosicioOcupada
```

Com veiem, cada component d'un individu és un cas particular d'un component abstracte, amb els seus propis atributs (identificador, fills, etc). Veiem que s'estableix un paral·lelisme entre els components de l'aplicació i les classes en la programació orientada a objectes. En ambdós casos, els objectes són instanciacions de classes abstractes, amb atributs propis i mètodes. Això respon a la voluntat plasmada en l'objectiu 5 de dissenyar una aplicació que introdueixi l'usuari en el món de l'algorísmia i la programació.

També en aquesta línia, és important observar que s'introdueixen dos altres conceptes importants: l'abstracció i la modularització, ja que l'estructura dels components en un individu està construïda com una llista d'arbres de mòduls. Suposem l'estructura de components de la figura 3.8.

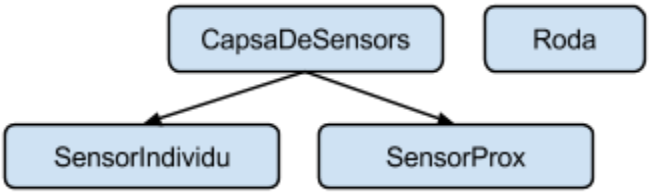

Figura 3.8: Exemple d'estructura de components.

Dins del mòdul *CapsaDeSensors*, s'hi pot trobar un sensor de proximitat *SensorProx* i un sensor d'individus *SensorIndividu*. Així, el component *CapsaDeSensors* s'erigeix com una

abstracció dels components que conté, i al mateix temps és un mòdul que conté tots els sensors de forma ordenada.

En resum, el cos d'un individu està compost per components que formen part d'un domini de components. Aquests components poden contenir altres components, i també poden tenir mètodes i atributs. Els mètodes poden ser de tipus *Set* o *Get*.

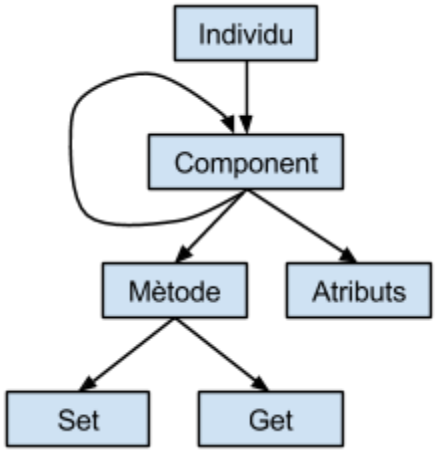

Figura 3.9: Estructura d'un individu.

A continuació es mostren els components que es volen implementar, amb els respectius mètodes.

- Roda:

Descripció	Permet el desplaçament dels individus.
Possibles fills	
Mètodes Get	
Mètodes Set	SET_MoureEndavant: Desplaça l'individu una posició endavant, és a dir, e el sentit en què està orientat. SET_GirarDreta: Gira l'individu 90º en sentit horari. SET_GirarEsquerra: Gira l'individu 90º en sentit antihorari.

- CapsaDeSensors:

Descripció	Contenidor de sensors.
Possibles fills	SensorProx, SensorEnergia4dDireccional, SensorIndividu
Mètodes Get	
Mètodes Set	

- CapsaDeSensors.SensorProx:

Descripció	És un sensor capaç de detectar obstacles davant l'individu.
Possibles fills	
Mètodes Get	GET_DetectarPosicioLliure: Si la posició de davant l'individu està ocupada per un altre individu, font d'energia o obstacle retorna Fals; si no, retorna Cert. GET_DetectarPosicioOcupada: Actua de forma contrària al mètode anterior, és a dir, retorna Cert quan la cel·la de davant l'individu està ocupada per un obstacle, font d'energia o individu. En cas contrari, retorna Fals. Encara que pot semblar redundant implementar les dues funcions, ja que una és la negada de l'altra, facilita l'edició de codi a l'usuari, ja que les condicions després de la instrucció IF no es poden negar.
Mètodes Set	

- RecolectorEnergia:

Descripció	Permet a l'individu proveir-se de l'energia que hi hagi a les fonts adjacents.
Possibles fills	
Mètodes Get	
Mètodes Set	SET_Recolectar: Absorbeix 20 unitats d'energia de cada font d'energia que hi hagi a les cel·les adjacents a l'individu. Si una font té menys de 20 unitats d'energia, un individu només absorbirà de la font en qüestió la quantitat d'energia que tingui, i la font desapareixerà.

- CapsaDeSensors.SensorEnergia4dDireccional:

Descripció	Permet detectar fonts d'energia en totes les direccions i a llarga distància.
Possibles fills	
Mètodes Get	GET_SensorDavant: Detecta si el primer obstacle que es troba davant de l'individu és una font d'energia. En aquest cas retorna Cert, i en cas contrari retorna Fals. GET_SensorDreta: Detecta si el primer obstacle que es troba a la dreta de l'individu és una font d'energia. En aquest cas retorna Cert, i en cas contrari retorna Fals. GET_SensorDarrere: Detecta si el primer obstacle que es troba darrere de l'individu és una font d'energia. En aquest cas retorna Cert, i en cas contrari retorna Fals. GET_SensorEsquerra: Detecta si el primer obstacle que es troba a l'esquerra de l'individu és una font d'energia. En aquest cas retorna Cert, en cas contrari retorna Fals.

Mètodes Set	

- CapsaDeSensors.SensorIndividu:

Descripció	Aquest sensor permet detectar la presència d'individus.
Possibles fills	
Mètodes Get	GET_DetectarIndividuADavant: Retorna Cert si a la cel·la de davant d l'individu hi ha un altre individu, retorna Fals en cas contrari.
Mètodes Set	

- MemoriaSimple:

Descripció	És una memòria d'un bit.
Possibles fills	
Mètodes Get	GET_LlegirValor: Retorna el valor de la memòria, que pot ser Cert o fals.
Mètodes Set	SET_GuardarCert: Fixa el valor de la memòria a Cert. SET_GuardarFals: Fixa el valor de la memòria a Fals.

- MemoriaComplexa

Descripció	Simula dos registres, A i B, d'1 byte i una senzilla unitat de còmput interna de l'individu. El bit 0 de cada registre és el de menor pes i el bit 7 és el d més pes. Per a fer les translacions de binari a decimal, es considerarà que Cert equival a 1 i Fals a 0.
Possibles fills	
Mètodes Get	GET_AIgualQueB: Retorna Cert si el valor dels bits del registre A coincideixen amb el valor dels del B, un a un. En cas contrari retorna Fals. GET_AMesGranQueB: Retorna Cert si el valor en decimal dels bits de registre A és estrictament superior al valor en decimal dels del registre B. En cas contrari retorna Fals. GET_AMesPetitQueB: Retorna Cert si el valor en decimal dels bits de registre A és estrictament inferior al valor en decimal dels del registre B. En cas contrari retorna Fals.
Mètodes Set	SET_RegistreASetBit0: Fixa a Cert el valor del bit 0 del registre A. SET_RegistreASetBit0: Fixa a Cert el valor del bit 1 del registre A. … (El mètode anterior existeix per cada un dels bits del registre A) SET_RegistreAUnsetBit0: Fixa a Fals el valor del bit 0 del registre A. SET_RegistreAUnsetBit1: Fixa a Fals el valor del bit 1 del registre A.

	… (El mètode anterior existeix per cada un dels bits del registre A) SET_RegistreAIncrementar: Incrementa el valor decimal del registre A en 1. SET_RegistreADecrementar: Decrementa el valor decimal del registre A en 1. SET_RegistreBSetBit0: Fixa a Cert el valor del bit 0 del registre B. SET_RegistreBSetBit1: Fixa a Cert el valor del bit 1 del registre B. … (El mètode anterior existeix per cada un dels bits del registre A) SET_RegistreBUnsetBit0: Fixa a Fals el valor del bit 0 del registre B. SET_RegistreBUnsetBit1: Fixa a Fals el valor del bit 1 del registre B. … (El mètode anterior existeix per cada un dels bits del registre B) SET_RegistreBIncrementar: Incrementa el valor decimal del registre B en 1. SET_RegistreBDecrementar: Decrementa el valor decimal del registre B en 1.

- Replicador

Descripció	Aquest component permet generar un individu amb el mateix algorisme.
Possibles fills	
Mètodes Get	
Mètodes Set	SET_FerFill: Crea un individu amb un algorisme còpia de l'individu que crida el mètode, sempre i quan alguna de les quatre posicions adjacents a l'individu estigui lliure.

El graf següent mostra la llista d'arbres de components que tindria un domini amb tots els components anteriorment vistos.

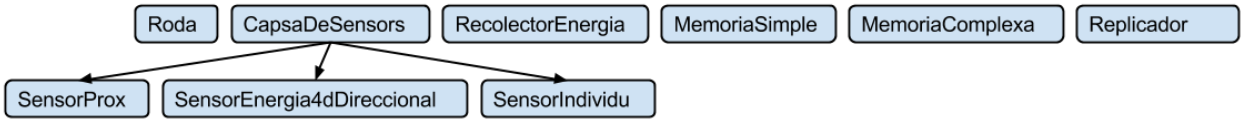

Figura 3.10: Estructura del domini amb tots els components implementats.

Per altra banda, volem que el domini de components d'una simulació el pugui especificar l'usuari a través d'un fitxer extern guardat al sistema de fitxers amb l'extensió *.dom*. El fitxer té el format següent:

(component <nom_component> fidelinia [fills fidelinia (<nom_component_fill> fidelinia)* [metodes fidelinia (<nom_metode>)*])*

El fitxer de la Figura 3.11 exemplifica el format del fitxer de domini.

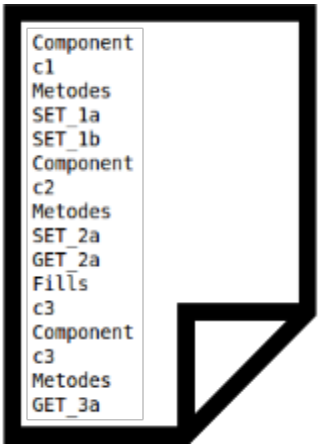

Figura 3.11: Exemple de fitxer *.dom*.

3.2.2 COMPORTAMENT

La conducta de l'individu, és a dir, les resposta que presenta als inputs que rep, està determinada per un algorisme. Cada individu té un algorisme que està format per línies de codi i un valor que indica quina és la línia de codi actual, el comptador del programa (PC). A l'inici de la simulació, el PC de cada individu serà la primera línia del seu codi, però amb cada línia interpretada, aquest PC s'anirà actualitzant.

Cada línia de codi segueix un dels patrons següents:

1) num_línia IF condició THEN EXE instrucció: si la condició retorna Cert, s'executa la instrucció, si retorna Fals no. En qualsevol cas s'incrementa en 1 el PC.

2) num_línia IF condició THEN GOTO num_línia_destí: si la condició retorna Cert, el PC pren el valor de num_línia_destí. Si retorna Fals, s'incrementa en 1.

3) num_línia EXE instrucció: s'executa la instrucció i s'incrementa el PC en 1.

4) num_línia GOTO num_línia_destí: el PC pren el valor de num_línia_destí.

De forma compacta, podem expressar la sintaxi del codi de la forma següent:

(nLinia [IF condicio THEN] (EXE instruccio | GOTO nLinia))*

Apuntem que les condicions fan referència a mètodes de tipus Get i les instruccions a mètodes de tipus *Set*.

3.3 INTÈRPRET

Per tal de llegir l'algorisme de cada individu i extreure'n les accions que ha de dur a terme necessitarem un intèrpret de codi.

Aquest intèrpret ha de recórrer el flux d'execució d'un algorisme a partir del valor del PC, fins que es troba una instrucció de tipus *Set*. Es retornarà la instrucció i s'incrementarà el PC

Pot ocórrer que aquesta instrucció de tipus *Set* no es trobi mai, per exemple en el cas que el flux d'execució entrés en un bucle d'aquest estil:

```
…
12 GOTO 12
...
```

Per solucionar aquest cas, quan el PC s'ha instanciat 10 cops i no s'ha trobat cap instrucció de tipus *Set*, s'atura la interpretació del codi i l'individu no fa res, NOOP.

L'intèrpret ha de ser capaç de llegir codi malgrat es doni alguna d'aquestes condicions:

- Que dues línies de codi tinguin el mateix número de línia.

```
…
12 EXE Roda_N0.SET_GirarEsquerra
12 GOTO 21
...
```

En aquest cas, la segona línia no serà mai interpretada.

- Que els números de línia no siguin consecutius.

```
3 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDavant THEN
        EXE Roda_N0.SET_MoureEndavant
5 GOTO 1
```

En aquest cas, el PC s'incrementarà fins que tingui el valor d'un número de línia existent. Si el PC té un valor superior al número de línia més alt, passarà a apuntar la línia amb el número de línia més baix. En el codi del requadre, es farien les traduccions següents:

- Si el PC és 1, 2, 3 o més gran que 5, passa a ser 3.
- Si el PC és 4 o 5, passa a ser 5.

- Que els números de línia estiguin desordenats.

```
...
14 EXE RecolectorEnergia_N0.SET_Recolectar
3 IF CapsaDeSensors_N1.SensorProx_N3.GET_DetectarPosicioLliure THEN GOTO 6
...
```

En aquest cas, s'interpretarà el codi com si estigués ordenat. Pel cas del requadre, quan s'hagi interpretat la línia amb número 14, la següent línia a interpretar no serà la que té el número 3, sinó que es buscarà la que té el número 15. Si no es troba, s'aplicaran les traduccions anteriorment mencionades.

En l'annex 1 es mostren diversos exemples de codi corresponents a possibles solucions per cadascun dels nivells.

3.4 SIMULADOR

Abans d'introduir la funcionalitat del simulador, cal diferenciar entre els tres modes de simulació que es volen implementar:

1) **Simulador lliure**: En aquest cas, l'usuari llança una simulació indeterminista, en la qual l'usuari pot determinar tots els paràmetres de la simulació a l'inici, modificar-ne alguns durant la simulació i interactuar amb la mateixa, afegint i eliminant individus i modificant-ne el comportament.

2) **Reproducció**: Només es reproduirà un procés prèviament simulat, de manera que l'execució és determinista. La informació per a la reproducció s'extreu de *checkpoints* que s'han generat en una simulació anterior en el mode 1. No ens extendrem en aquesta funcionalitat perquè ha estat desenvolupada pel Nicolau Manubens i no entra dins l'àmbit d'aquest projecte.

3) **Videojoc**: En aquest cas, la simulació comença a partir d'entorns o nivells predissenyats, i l'execució es duu a terme fins que es dóna una condició de victòria o fins que s'arriba al límit d'iteracions.

La funcionalitat del simulador es pot dividir en diferents fases:

1) **Inicialització**: En la inicialització, es carrega un mapa i s'hi reparteix energia en forma de fonts d'energia de forma aleatòria. Un cop repartida l'energia per l'escenari, es creen els individus fins que s'arriba a un valor de mínim d'individus. Quan es crea un individu, se li atorga energia que s'extreu de l'escenari seguint la fórmula següent:

energia_a_individu = min(energia_restant_a_escenari, energia_inicial_individu)
on:

- energia_restant_a_escenari: és la suma d'energia de les fonts d'energia que hi ha a l'escenari.
- energia_max_individu: és un valor fixat que indica el màxim d'energia que s'ha d'atorgar a un individu en el moment de ser creat.

Es pot donar el cas que energia_a_individu sigui 0. En aquest cas, l'individu serà creat igualment, però no tindrà energia i presumiblement morirà tot just arranqui la simulació.

A mesura que es va eliminant energia de les fonts, aquestes van desapareixent de l'escenari.

El mapa a carregar, el domini de components, la quantitat d'energia a repartir per l'escenari, el valor de mínim d'individus i la posició, orientació i algorisme de cada individu són aleatoris en el mode 1, determinats per la informació del *checkpoint* en el mode 2 i fixats en funció del nivell de joc en el mode 3.

Per tant, el fitxer de *checkpoints* a reproduir en el mode 2 els determinarà l'usuari, igual que el nivell de joc en el mode 3.

2) **Execució**: Per tal de gestionar l'activitat de cada individu en el temps, dividim l'execució en iteracions, de manera que a cada iteració del programa i per a cada individu es demana a l'intèrpret que ens doni la següent instrucció de tipus *Set* del seu algorisme, que serà executada.

A més, el simulador evita les col·lisions entre els diferents objectes de l'escenari. Si un individu pretén desplaçar-se a una cel·la que ja està ocupada, simplement es quedarà on és. Si dos individus intenten ocupar una cel·la desocupada, només un dels dos l'ocuparà, concretament el que s'hagi creat abans.

Cada individu consumeix energia a cada iteració pel fet d'existir, i a més consumeix energia per cada instrucció que duu a terme. Si es queda sense energia, l'individu desapareix. L'energia que consumeixen els individus es guarda en forma de residu. Quan la quantitat d'energia en forma de residu arriba a un valor màxim, es crea un font d'energia a l'escenari i el valor d'energia residual de l'individu en qüestió es fixa a 0. Quan l'individu mor, es crea una font d'energia amb tanta energia com energia residual tenia l'individu en el moment de morir. Així, l'energia de l'entorn es manté sempre constant, considerant com energia de l'entorn la suma de

les energies del individus, la de les seves energies residuals i també la suma d'energies de les fonts d'energia.

Una altra funció del simulador, doncs, és eliminar els individus que s'han quedat sense energia i crear nous individus quan el número d'individus és inferior a un mínim establert.

En el mode 1, el simulador també s'encarrega de comprovar si s'ha de generar un *checkpoint*, i generar-lo si és el cas. La generació de *checkpoints* i el nombre d'iteracions entre *checkpoints* es deixen a decisió de l'usuari abans de començar la simulació. De nou, no aprofundiré més en aquest apartat ja que el desenvoluparà el Nicolau Manubens i no entra dins l'àmbit d'aquest projecte.

A més, a cada iteració es comprova si es compleix alguna de les condicions de finalització de la simulació. Aquestes condicions poden ser:

- S'ha arribat al límit d'iteracions. Aquest límit està fixat per l'usuari en el mode 1, pel fitxer de *checkpoint* en el mode 2 o determinat en funció del nivell de joc en el mode 3. Si el màxim d'iteracions és 0, es considerarà infinit.
- Es compleix alguna de les condicions de victòria. Això només es produeix en el mode 3, quan es donen les condicions fixades en un nivell per considerar que l'usuari ha programat un bon algorisme, per exemple, que l'individu estigui en una posició de l'escenari o que hi hagi un determinat nombre d'individus a l'escenari.
- L'usuari decideix finalitzar l'execució. Això es pot donar en qualsevol dels tres modes.

3) **Finalització**: En els modes 1 i 2 es dóna la possibilitat a l'usuari de guardar els cromosomes dels individus que estiguin vius en el moment de finalitzar la simulació. Si aquesta opció està activa, quan s'acaba la simulació s'han de guardar en un fitxer extern els algorismes dels individus que hi hagi a l'escenari.

3.5 INTERFíCIE GRÀFICA D'USUARI (GUI)

Per tal que l'usuari pugui fixar el valor dels diferents paràmetres d'una simulació, visualitzar-la i interactuar-hi, hem dissenyat i implementat una GUI. Aquesta permet especificar els paràmetres i controlar la funcionalitat de l'aplicació completa, tant de les parts que entren dins l'àmbit d'aquest projecte com de la generació i reproducció de *checkpoints*. Per tant, encara que ja s'ha mencionat aquesta funcionalitat en l'apartat anterior, en farem una breu aproximació abans de continuar amb la descripció de la GUI.

Com que el procés evolutiu requereix de moltes iteracions, és convenient que es pugui dur a terme sense que es representi l'entorn tridimensional per pantalla a cada iteració. El que sí hem cregut necessari és poder reproduir un procés evolutiu a partir de qualsevol punt. Per això es vol possibilitar la generació de *checkpoints* que guarden l'estat de l'entorn en un moment determinat i la informació necessària per tal de reproduir el procés fins al pròxim *checkpoint*.

Això també permet reprendre un procés evolutiu que s'ha començat en una altra màquina o en un altre procés a partir del seu últim *checkpoint*. Mentre que la implementació d'aquesta funcionalitat no és un dels objectius d'aquest treball, sí que ho és el disseny i implementació de la interfície que permet gestionar aquests *checkpoints* i el propi procés.

A la taula següent es mostren els diferents paràmetres que determinen una simulació, i s'indica si l'usuari els pot fixar o controlar en funció del mode.

	Mode 1: SIMULACIÓ LLIURE	**Mode 2: REPRODUCCIÓ**	**Mode 3: VIDEOJOC**
Control progrés (inicialitza, para, accelera, desaccelera, etc.)	Actiu	Actiu	Actiu
Triar entorn	Actiu quan la simulació no està en fase d'execució	No actiu. Especificat al fitxer de *checkpoints*	No actiu. Determinat pel nivell.
Triar domini	Actiu quan la simulació no està en fase d'execució	No actiu. Especificat al fitxer de *checkpoints*	No actiu. Determinat pel nivell.
Triar mínim d'individus	Actiu quan la simulació està en pausa	No actiu. Especificat al fitxer de *checkpoints*	No actiu. Determinat pel nivell.
Triar màxim d'iteracions	Actiu	No actiu. Especificat al fitxer de *checkpoints*	No actiu. Determinat pel nivell.
Guardar cromosomes + ruta	Actiu	Actiu	No actiu
Visualització on/off	Actiu	Actiu	Actiu
Guardar *checkpoints* + ruta	Actiu quan la simulació no està en fase d'execució	No actiu	No actiu
Estadístiques	Actiu	Actiu	Actiu
Dades dels individus	Actiu	Actiu	Actiu
Edició algorisme	Actiu	No actiu	Actiu només en la

			fase d'inicialització
Indicador *checkpoint* en progrés	No actiu	Actiu	No actiu
Número d'iteracions per *checkpoint*	Actiu	No actiu	No actiu
Seleccionar ruta de *checkpoint* a carregar	No actiu	Actiu només abans de la reproducció	No actiu
Seleccionar nivell de joc	No actiu	No actiu	Actiu
Afegir individu	Actiu	No actiu	No actiu

Vegem les diferents parts de què es compon la GUI:

Figura 3.12: Disseny de la interfície.

3.5.1 PARÀMETRES DE LA SIMULACIÓ

Consta de tres pestanyes que permeten establir els paràmetres dels tres modes de simulació:

- **Simulació lliure**: En aquesta pestanya es mostren les opcions que permeten especificar el fitxer d'entorn i de domini. També ha de permetre especificar el mínim d'individus, el màxim d'iteracions (o duració de la simulació), l'energia total de l'entorn (que recordem es mantindrà constant al llarg de la simulació), si es volen guardar els cromosomes dels individus vius en el moment de finalitzar la simulació i a on, i si es volen generar *checkpoints* i a on s'han de guardar. També es vol donar la possibilitat a l'usuari que afegeixi individus de forma manual, especificant la posició i orientació.

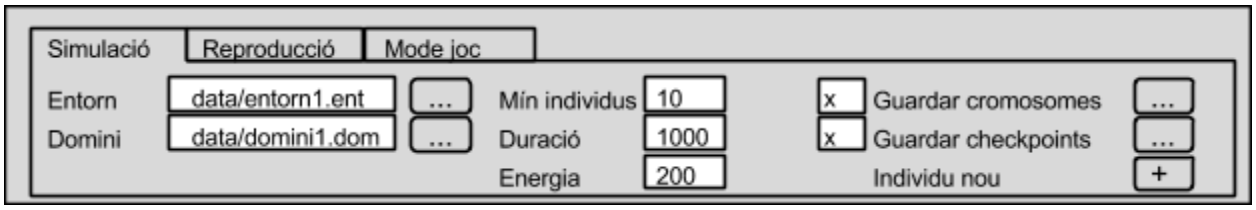

Figura 3.13: Pestanya del mode *Simulació lliure*.

- **Reproducció**: En aquesta pestanya s'ha de permetre seleccionar el fitxer de *checkpoints* a reproduir, i si es volen guardar o no els cromosomes al finalitzar la reproducció.

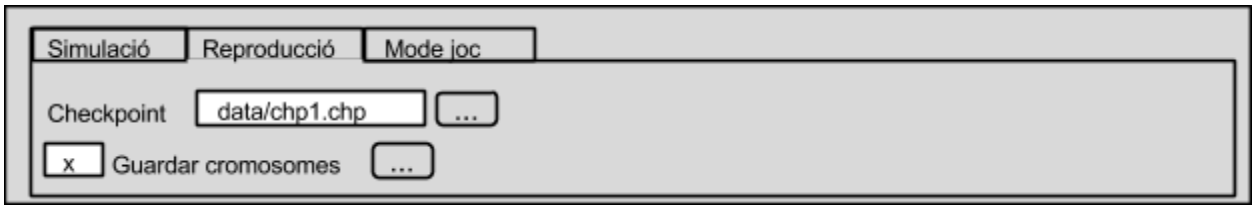

Figura 3.14: Pestanya del mode *Reproducció*.

- **Videojoc**: En aquesta pestanya només s'ha de permetre seleccionar el nivell de joc, que per defecte ha de ser el 0, recarregar el nivell i mostrar informació referent al nivell.

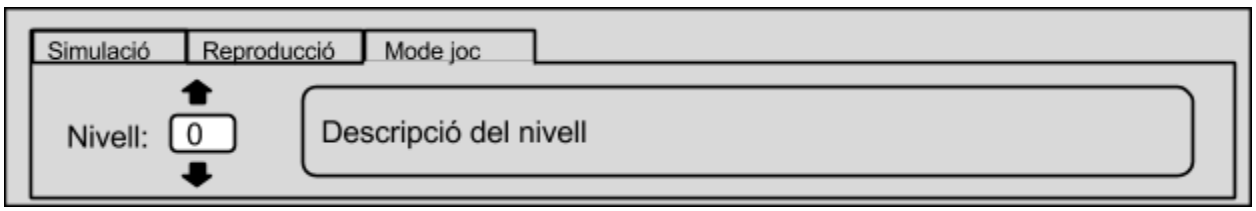

Figura 3.15: Pestanya del mode *Videojoc*.

3.5.2 VISUALITZACIÓ 3D

La representació tridimensional de l'entorn ha de mostrar per pantalla l'entorn, és a dir, el pla, els obstacles, les fonts d'energia i els individus a les seves respectives posicions, i pemetre a l'usuari navegar per aquest entorn i seleccionar individus. Això atén l'objectiu 4.

La representació dels objectes de l'entorn es representaran amb cossos geomètrics simples, cubs i esferes.

La càmera està orientada inicialment al centre de l'entorn, tal i com es mostra a la Figura 3.16.

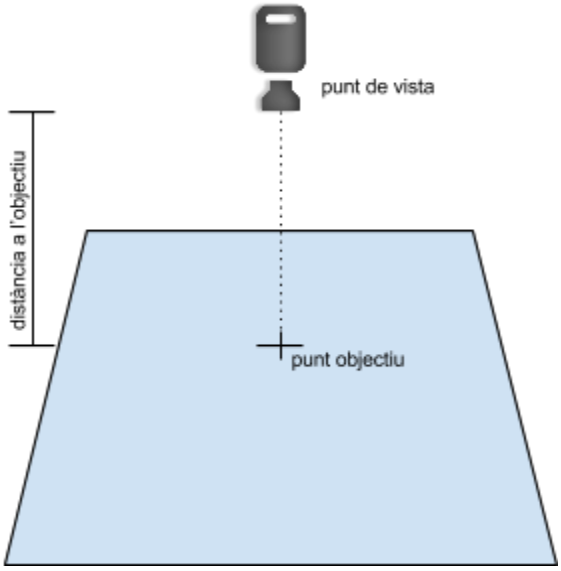

Figura 3.16: Inciialització de la posició de la càmera.

La parella punt de vista i punt objectiu es pot desplaçar endavant (W), endarrere (S), a l'esquerra (A) i a la dreta (D) sobre el pla de l'entorn, i també amb la combinació de tecles Ctrl + Botó esquerre del ratolí. Només amb el botó esquerre del ratolí podem col·locar la càmera en qualsevol punt de la semiesfera que formen el punts col·locats per sobre el pla i a una distància determinada del punt objectiu.

Amb la rodeta del ratolí, es pot modificar la distància del punt de vista al punt objectiu.

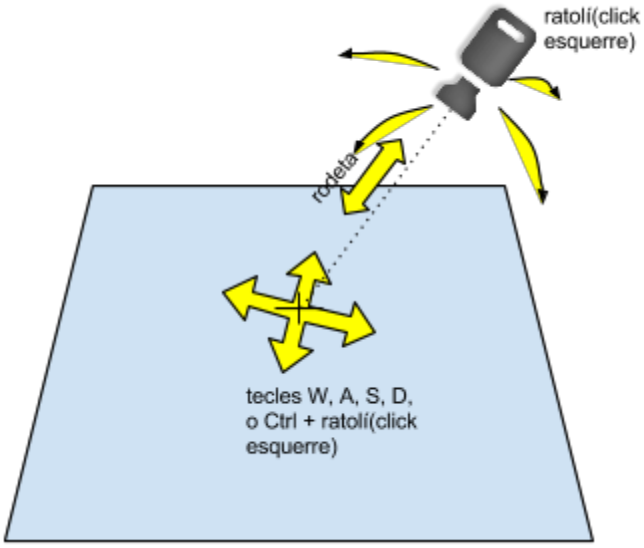

Figura 3.17: Control de la càmera.

3.5.3 CONTROL DE LA SIMULACIÓ

En aquest espai hi ha els botons que permeten començar, pausar i finalitzar una execució. També permeten accelerar i desaccelerar la simulació, hi ha una barra d'estat que mostra el percentatge d'iteracions que s'ha reproduït i s'ha de permetre activar i desactivar la visualització. Això últim té especial rellevància si es pretén realitzar una simulació molt llarga i no ens interessa veure el procés, ja que desactivar la visualització reduirà els càlculs a realitzar i accelerarà simulació.

3.5.4 INFORMACIÓ SOBRE L'ENTORN

En aquest espai es mostren dades de l'entorn com el número d'individus, la mitja d'energia dels individus, etc.

3.5.5 INFORMACIÓ I OPCIONS DE L'INDIVIDU

Es compon de dues pestanyes:

- **Informació**: Es mostrarà en aquesta pestanya informació sobre l'individu seleccionat, si és que n'hi ha cap: identificador, energia, edat, PC, algorisme, etc.

- **Editor**: En aquesta pestanya es pot modificar l'algorisme, editant, afegint i eliminant les línies del mateix. Es detalla la funcionalitat en l'apartat **3.6**.

Figura 3.18: Disseny de l'editor d'algorismes.

3.6 EDITOR D'ALGORISMES

Tot i que l'editor d'algorismes forma part de la GUI, en presentem la funcionalitat en un apartat diferent ja que en si representa un bloc funcional autònom i ben definit. A partir d'un algorisme i un domini de components, ha de poder generar un altre algorisme.

L'editor d'algorismes és l'eina a través del qual l'usuari pot generar algorismes dins l'entorn del programa de forma segura a nivell sintàctic i semàntic. A més, l'usuari pot guardar aquests algorismes en un fitxer extern o obrir-ne de ja creats. Els algorismes es guarden en el sistema de fitxers com a arxius amb l'extensió *.cro*. A més, l'algorisme es pot assignar a l'individu que estigui seleccionat en el moment d'editar l'algorisme. Això, evidentment, modificarà la conducta de l'individu.

Per tal d'editar un algorisme, l'usuari pot afegir línies a qualsevol punt del mateix, i també seleccionar qualsevol línia per tal de modificar-la o esborrar-la.

Durant l'edició de línies de codi, l'usuari només ha de tenir disponibles totes les opcions correctes a nivell lèxic i semàntic per completar la línia. Així, es restringeix el marge d'actuació de l'usuari, que se sent més orientat, i s'anulen les possibilitats que generi un algorisme incorrecte. És evident, doncs, que l'editor d'algorismes és un mòdul que necessita conèixer el domini de components per tal de mostrar quins són els components als quals pot fer referència una condició o instrucció.

3.7 MODE VIDEOJOC

El mode Videojoc el constitueixen una sèrie de nivells cadascun dels quals amb un entorn inicial, una duració màxima (en iteracions), un mínim d'individus i un objectiu determinat.

A cada nivell, es permet editar l'algorisme d'un individu només una vegada, abans de simular la conducta de l'individu un número màxim d'iteracions. Si en algun moment es compleix la condició de final de nivell, apareix un missatge indicant-ho i l'usuari sap que ha programat un bon algorisme. Si s'arriba al màxim d'iteracions, l'algorisme de l'usuari no és prou bo.

Si l'objectiu del nivell és situar l'individu en una posició, s'indicarà amb un requadre vermell.

A continuació es descriuen els 4 nivell dissenyats, i a l'**Annex 1** es mostren possibles solucions per cada nivell.

Nivell_0: L'objectiu d'aquest nivell és que l'usuari prengui contacte amb l'entorn i dissenyi un algorisme molt bàsic que condueixi l'individu en línia recta fins a una casella objectiu, marcada amb vermell, com es mostra a la Figura 3.18.

Figura 3.19: Nivell 0.

- Nivell 1: Aquest nivell està pensat perquè l'usuari utilitzi les instruccions de control de flux, ja que l'individu ha de comprovar que topa amb la paret i, en conseqüència, prendre la decisió de girar.

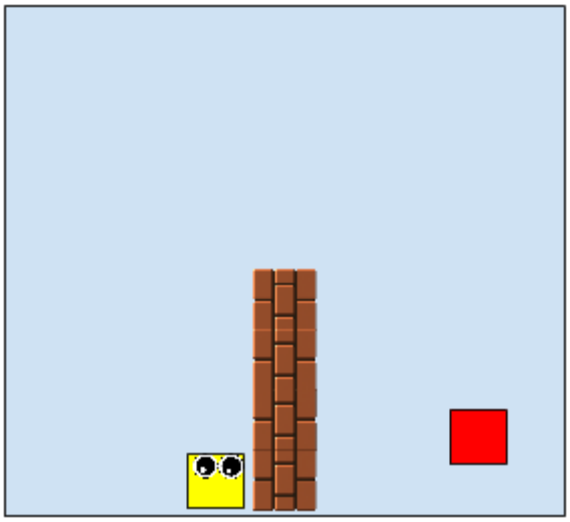

Figura 3.20: Nivell 1.

- Nivell 2: En aquest nivell l'individu ha de fer un recorregut fins a arribar al destí, com en el cas anterior. La diferència és que en qualsevol punt del recorregut hi pot trobar fonts d'energia que impediran el pas. Com que no es pot passar a través de les fonts, aquestes hauran de ser absorbides. Dissenyar un algorisme per resoldre aquest cas és més complex que l'anterior.

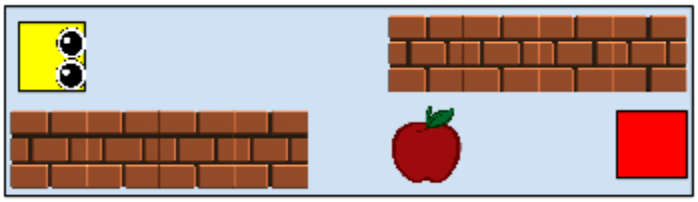

Figura 3.21: Nivell 2.

- Nivell 3: L'últim nivell planteja a l'usuari dissenyar un algorisme capaç d'expandir-se i perpetuar-se. L'objectiu del nivell és, a partir d'un individu, generar un mínim de 20 individus. Per fer això serà necessari que cada individu sigui capaç d'abastir-se de menjar i reproduir-se.

El disseny del nivell, en aquest cas, no té massa importància. Cal que sigui força gran i que hi hagi prou fonts d'energia.

4 IMPLEMENTACIÓ

4.1 EINES

Per a la realització d'aquest projecte hem intentat equilibrar la voluntat de dur a terme una implementació des de zero, expressada a l'objectiu 1, amb les possibilitats reals de desenvolupar nosaltres mateixos cada part del projecte, tenint en compte el volum de feina i el perllongament en el temps que això suposaria. Per tant, al servir-nos de qualsevol eina, hem valorat si s'adequava a les nostres necessitats, si realment valia la pena el temps que ens estalviàvem i si permetria que l'aplicació seguís sent flexible i escalable. Finalment he decidit utilitzar el llenguatge C++, per la seva potència i versatilitat, a més de comptar amb una gran comunitat que ha ajudat a depurar i millorar el llenguatge i desenvolupar i testejar llibreries molt interessants. A més, és un llenguatge que ja dominava abans de començar la implementació, la qual cosa m'ha estalviat temps d'aprenentatge.

Pel que fa a l'entorn de programació, hem optat per Qt, una eina de desenvolupament de codi que a més integra extensions per al disseny d'interfícies.

Hem utilitzat la llibreria externa *OpenGL*, que a més de ser una eina gratuïta i molt potent, ja dominava a priori.

Tractant-se d'un projecte col·laboratiu, per al control de versions ens ha semblat oportú treballar amb Git, amb el qual ja havíem treballat els dos.

Figura 4.1: Logotips de les eines utilitzades.

4.2 ENTORN

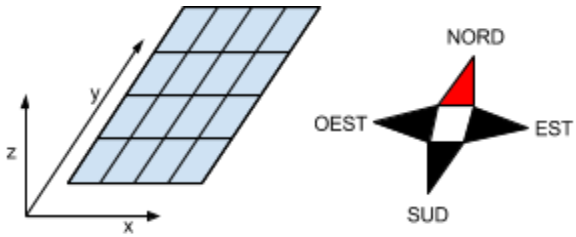

Figura 4.2: Pla de l'entorn.

L'entorn és una matriu de dues dimensions, X i Y. L'eix Z representarà l'alçada dels objectes. Cada objecte ocuparà una i només una cel·la del pla, i els individus, si es desplacen,

ho faran de cel·la a cel·la, incrementant i decrementant els valors X i Y de la seva posició. En el sentit positiu de l'eix Y es situarà el Nord, i la resta de punts cardinals s'ubicaran en conseqüència, tal i com es mostra a la Figura 4.2.

4.3 INDIVIDU

Cada instància de la classe individu tindrà el seu propi algorisme i estructura de components, a més d'un identificador que serà únic en cada simulació. També es guardarà la seva posició i orientació, energia, energia residual, iteracions viscudes, identificador del primer ancestre (si el té), suma d'iteracions passades a partir de la creació del seu primer ancestre i el nombre d'ancestres.

La variable que guarda l'identificador únic de cada individu és un enter sense signe codificat amb 4 bytes, i amb rang de valors de 0 a 4.294.967.295. Això suposa una limitació del número d'individus diferents que es poden instanciar en una simulació. Aquesta limitació, però, només afecta al mode 1 (Simulació lliure), ja que en el mode 2 (Reproducció) només es reprodueix el que ja s'ha simulat al mode 1, i en el mode 3 (Videojoc) en cap cas es creen tants individus. Suposant que en el mode 1 s'arribés al límit d'individus possibles, es podrien guardar els cromosomes i començar una altra simulació amb els mateixos paràmetres i assignant a cada individu un dels cromosomes guardats.

4.3.1 COMPONENTS

La llista d'arbres de components de cada individu es construeix a partir del seu algorisme. Tant si l'algorisme es crea de forma aleatòria (per exemple en la creació d'individus en la inicialització o quan hi ha menys individus que el llindar especificat, en el mode 1), com si l'edita l'usuari, com si es genera per mutació d'un algorisme ja existent, pot contenir qualsevol component del domini i també qualsevol mètode. Un cop creat l'algorisme, es genera l'arbre de components.

Per exemple, suposem l'algorisme següent:

```
0 EXE Roda_N0.SET_GirarDreta
3 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioLliure THEN GOTO 9
4 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N1.GET_SensorDreta
        THEN GOTO 1
5 EXE RecolectorEnergia_N1.SET_Recolectar
6 EXE RecolectorEnergia_N3.SET_Recolectar
```

L'estructura de components que es construiria seria el de la Figura 4.3.

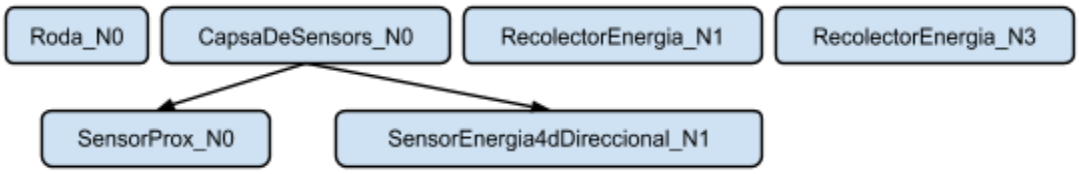

Figura 4.3: Estructura de components.

També s'ha implementat una funció per obrir i interpretar dominis guardats en un fitxer extern.

4.3.2 CONDUCTA

L'algorisme es guarda en una estructura de dades com la representada a la Figura 4.4.

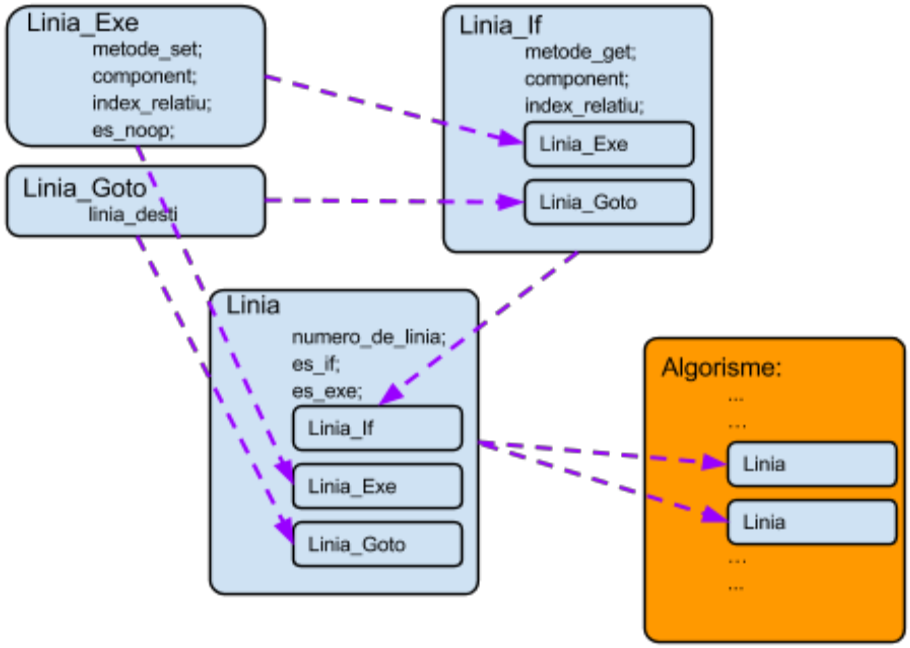

Figura 4.4: Estructures de dades de l'Algorisme.

Tal i com s'explica a l'apartat de disseny, les línies de codi poden tenir 4 formats diferents. Les variables **es_if** i **es_exe** són *booleans* que codifiquen el tipus de línia de la forma següent.

es_if	**es_exe**	**tipus de línia**
Cert	Cert	<num_linia> IF <condicio> THEN EXE <instruccio>
Cert	Fals	<num_linia> IF <condicio> THEN GOTO <num_linia_desti>
Fals	Cert	<num_linia> EXE <instruccio>
Fals	Fals	<num_linia> GOTO <num_linia_desti>

4.4 INTÈRPRET

Cada línia de codi s'avalua seguint l'algorisme de la Figura 4.5.

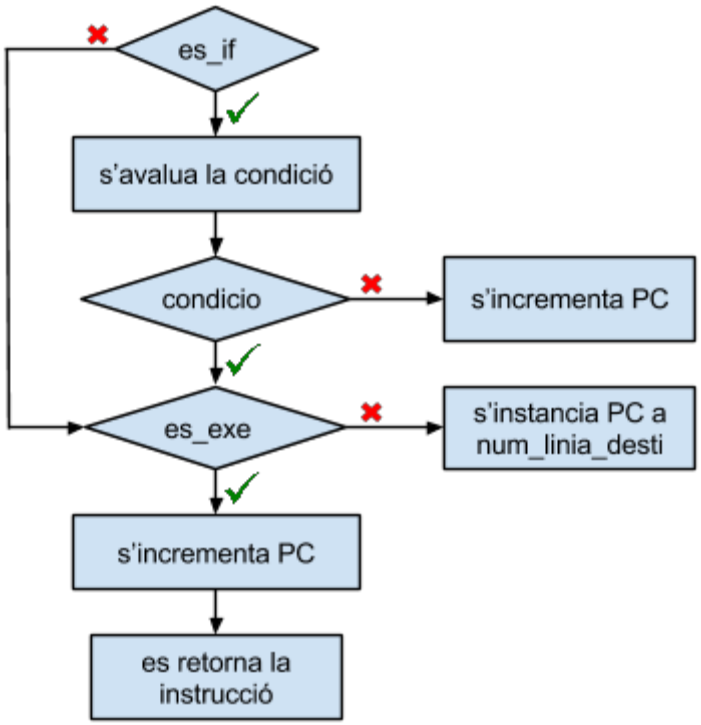

Figura 4.5: Diagrama de flux de la interpretació de línies de codi.

Recordem que l'intèrpret ha de traduir el PC al següent valor de número de línia existent i superior. En cas que el PC sigui superior al número de línia més elevat del codi, es tradueix al número de línia més petit.

Quan es retorna una instrucció, ja no s'avaluen més línies de codi. Si PC s'ha incrementat o instanciat 10 cops, s'ha de retornar una instrucció tipus NOOP.

4.5 SIMULADOR

S'ha implementat el Simulador amb les funcionalitats especificades a l'apartat **3.4**. A cada iteració es comprova que la mida del vector d'individu de l'entorn és superior o igual al mínim d'individus. Si no és així, es generen individus amb algorismes aleatoris en els modes 1 i 3, i amb un algorisme predeterminat en el mode 2. La generació d'algorismes aleatoris no entra dins l'àmbit d'aquest projecte.

Per cada individu es comprova que el seu nivell d'energia és superiora 0. De no ser així, l'individu s'elimina i la seva energia residual es retorna a l'escenari en forma de font d'energia. També es comprova si el seu nivell d'energia residual és superior o igual a 100. En aquest cas,

també es generarà una font de nivell 100 d'energia i es posarà a 0 el comptador d'energia residual de l'individu.

4.6 INTERFÍCIE GRÀFICA D'USUARI (GUI)

Figura 4.6: Interfície de l'aplicació.

4.6.1 PARÀMETRES DE LA SIMULACIÓ

S'han implementat les tres pestanyes corresponents als 3 modes d'execució:

- **Simulació lliure**:

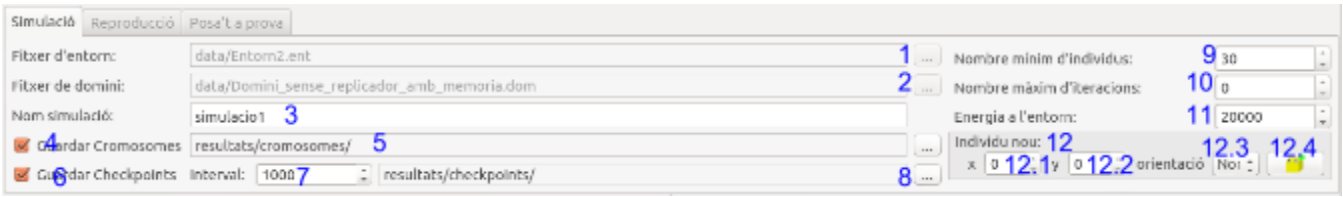

Figura 4.7: Pestanya del mode *Simulació lliure*.

1) Permet especificar el fitxer que codifica l'entorn. Obre una finestra per explorar el sistema d'arxius.

2) Permet especificar el fitxer que codifica el domini. Obre una finestra per explorar el sistema d'arxius.

3) Permet fixar el nom de la simulació. Això és necessari per crear les carpetes on es guardaran els *checkpoints* i els cromosomes. Si s'inicialitza una simulació en la que es volen guardar checkpoints i/o cromosomes i el nom no està especificat, es ressaltarà el requadre de vermell i no s'inicialitzarà la simulació.

4) Permet especificar si es volen guardar o no cromosomes dels individus vius a l'última iteració.

5) Permet especificar a quina carpeta s'ha de crear la carpeta on es guardaran els cromosomes.

6) Permet especificar si es volen generar o no *checkpoints* durant el procés.

7) Permet especificar el número d'iteracions que separa els *checkpoints*.Permet especificar a quina carpeta s'ha de crear la carpeta on es guardaran els *checkpoints*.

8) Permet especificar a quina carpeta s'ha de crear la carpeta on es guardaran els checkpoints.

9) Permet especificar el número mínim d'individus de la simulació. Aquest paràmetre es pot modificar mentre la simulació està en marxa.

10) Permet especificar el número màxim d'iteracions. Si es fixa a 0, s'interpretarà que el número màxim d'iteracions és infinit. Aquest paràmetre es pot modificar mentre la simulació està en marxa.

11) Permet especificar l'energia total de l'entorn. Aquest paràmetre es pot modificar mentre la simulació està en marxa.

12) Aquest requadre permet afegir un individu a l'escenari. Conté les opcions següents:

12.1) Permet fixar la posició X on apareixerà l'individu nou.

12.2) Permet fixar la posició Y on apareixerà l'individu nou.

Quan es modifiquen els valors de X o Y on ha d'aparèixer el nou individu, apareix un requadre vermell que desapareix lentament sobre la posició corresponent en el visualitzador.

12.3) Permet fixar la orientació de l'individu nou.

12.4) Amb aquest botó s'afageix un individu nou a l'escenari a la posició especificada (si no hi ha cap obstacle) i amb la orientació indicada.

- **Reproducció**:

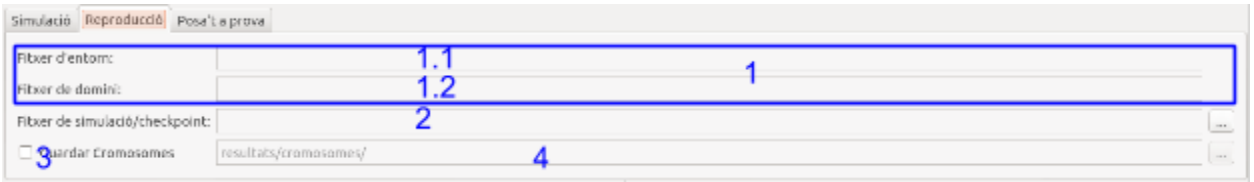

Figura 4.8: Pestanya del mode *Reproducció*.

1) Informació no editable:

1.1) Informació de qui és el fitxer d'entorn.

1.2) Informa de quin és el fitxer de domini.

2) Permet seleccionar un fitxer de *checkpoint*.

3) Permet especificar si es volen guardar el cromosomes al final de la reproducció.

4) Si s'habilita l'opció 3, permet especificar a quina carpeta guardar els cromosomes.

- **Videojoc**: Les opcions que dóna la pestanya del videojoc són les següents:

Figura 4.9: Pestanya del mode *Videojoc*.

1) Permet seleccionar el nivell:

2) Recarrega el nivell, és a dir, posa l'individu a la seva posició inicial i torna a mostrar l'objectiu.

3) En aquesta part es mostra informació no editable referent al nivell, ja que formen part de la dificultat pròpia del nivell.

3.1) Es mostra el fitxer d'entorn.

3.2) Es mostra el fitxer de domini.

3.3) Es mostra el mínim d'individus del nivell.

3.4) Es mostra la duració màxima del nivell en iteracions. Si s'arriba al màxim d'iteracions no es dóna l'algorisme per bo.

3.5) Es mostra el total d'energia que hi ha a l'entorn.

3.6) Descripció del nivell en què s'explica l'objectiu i es dóna alguna pista a l'usuari.

3.7) Imatge representativa del nivell.

4.6.2 VISUALITZACIÓ 3D

Es vol implementar una visualització de l'entorn funcional i senzilla i que a més permeti controlar la càmera amb les tecles i el ratolí tal i com s'ha definit en l'apartat de disseny, i seleccionar individus. Com que treballem amb *OpenGL*, hem de tenir en compte amb quin ordre es processa la pila de matrius que composaran l'escena final.

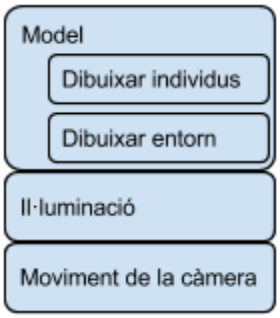

Figura 4.10: Pila de matrius en l'ordre en què les calcula *OpenGL*.

Vegem els diferents blocs tractats:

- **Model**: El terra el composarà un pla de color gris, els obstacles es representaràn amb un cub blau, les fonts d'energia amb una esfera verda i els individus amb un cub groc amb ulls per saber la seva orientació. En alguns casos es poden pintar els individus amb altres colors per tal de diferenciar els més antics o els que tenen més antecessors. Es comentarà aquesta funcionalitat en l'apartat **4.6.4.**

- **Il·luminació**: Hem activat la il·luminació tipus *Gouraud* [8], a través de la funció *glShadeModel()* [9], que calcula el sombreig o matís de color dels vèrtex de cada polígon i la interpola per la resta de punts dels polígon. És el tipus d'il·luminació que ofereix millors resultats a nivell de realisme mantenint un compromís amb el cost computacional. No hem escollit una il·luminació més senzilla perquè hi ha l'opció de desactivar la visualització si es desitja una simulació ràpida.

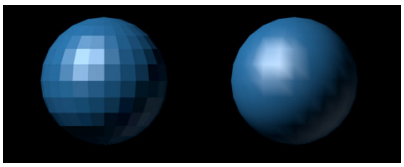

Figura 4.11. A l'esquerra il·luminació simple, en què s'assigna un sol color a cada polígon. A la dreta il·luminació calculada amb *Gouraud*.

La il·luminació és direccional, en el sentit negatiu de l'eix Z, tal i com es mostra a la figura següent.

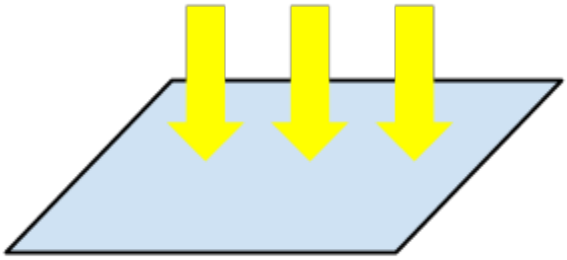

Figura 4.12: Direcció de la llum en l'entorn, de dalt a baix.

S'han configurat els diferents components RGBA dels diferents tipus d'il·luminació de la forma següent:

	Red	**Green**	**Blue**	**Alpha**
Ambient	0.1	0.1	0.1	1.0
Difusa	0.5	0.5	0.5	1.0
Especular	0.0	0.0	0.0	1.0

No es permet variar els paràmetres de la il·luminació a l'usuari en temps d'execució perquè en cap cas aporta una funcionalitat substancial a l'aplicació en la direcció establerta als objectius.

- **Càmera**: En OpenGL, podem especificar el volum davant la càmara dins el qual queden els objectes que es mostraran per pantalla a través de la funció *gluPerspective()* [10], a partir dels paràmetres següents: angle vertical de visió, relació d'aspecte horitzontal/vertical, distància al pla proper i distància al pla llunyà.

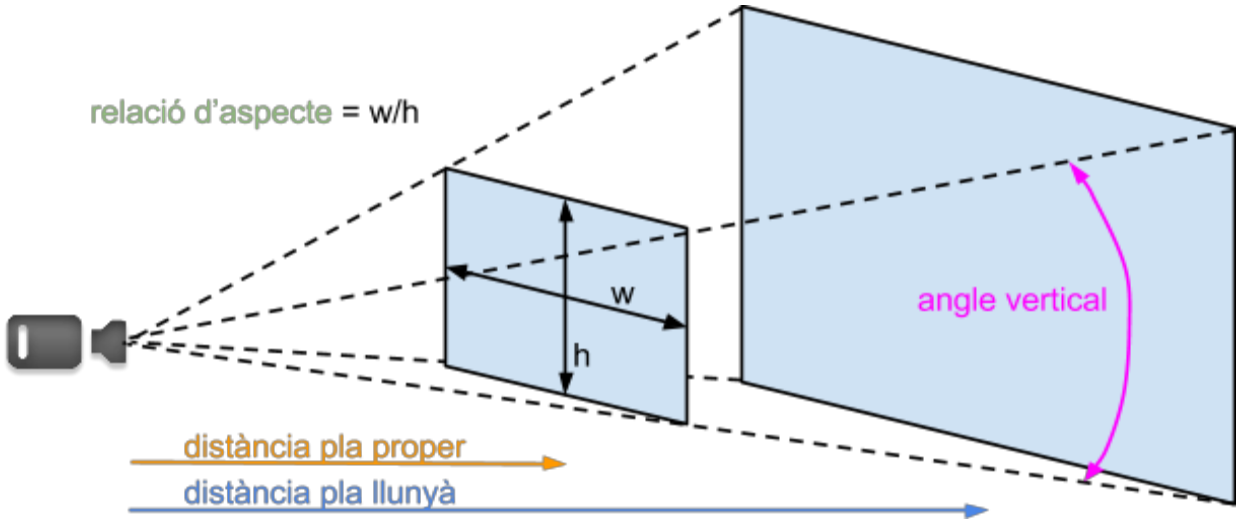

Figura 4.13: Volum de visualització.

L'angle de visió s'ha fixat a 30º, les distàncies als plans proper i llunya a 0.1 i 500 respectivament i la relació d'aspecte es calcula a partir de l'amplada i l'alçada de la finestra de visualització. D'aquesta manera els objectes apareixen amb les proporcions adequades i no es deformen quan ajustem la mida de la finestra en temps d'execució.

La posició i orientació del punt de vista en un espai tridimensional, en *OpenGL*, else podem especificar amb la funció *gluLookAt()* [11], a partir dels vectors que defineixen els paràmetres següents: posició, punt de mira i vector amunt.

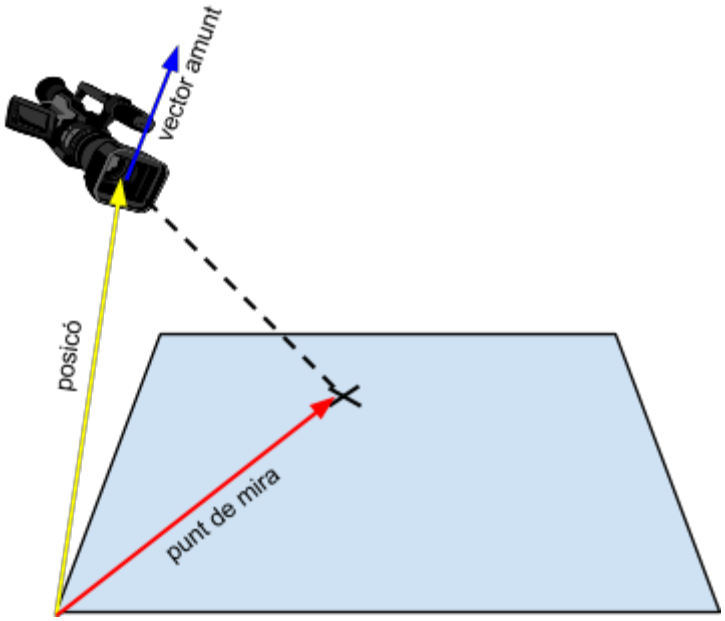

Figura 4.14: Posició i orientació de la càmera determinats pels vectors de posició, punt de vista i amunt.

Com veiem, són tres vectors en un espai tridimensional i que, per tant, tenen tres coordenades.

Per defecte, volem que el punt de vista es situï al centre del mapa, mirant en el sentit negatiu de l'eix Z i a una distància que permeti observar tot l'escenari i proporcioni un marge als contorns.

A partir de les mides X i Y del mapa, calculem el centre, que serà X/2 i Y/2.

Per tant, el punt de mira serà (X/2, Y/2, 0). La posició del punt de vista comparteix les mateixes components X i Y, però per calcular la Z s'ha d'aplicar el teorema del sinus [12].

Sigui r el radi de la circumferència que conté l'entorn, i α l'angle de visió, l'alçada a la que s'ha d'ubicar el punt de vista, és a dir, el component Z de la càmera, és:

alçada = radi * (sin(180º - 90º - α/2) / sin(α/2))

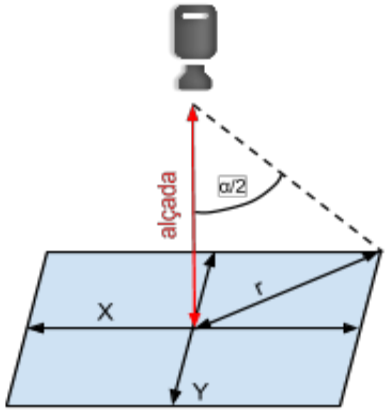

Figura 4.15: Posició inicial de la càmera al carregar un mapa.

Per tal de controlar el punt de vista des del qual es visualitza l'entorn tal i com s'ha comentat a l'apartat de disseny, es tenen les estructures de dades següents (columna de l'esquerra), inicialitzades com s'explicita (columna de la dreta):

estructura Posicio{
enter x,
enter y,
enter z,
}
Posicio **punt_de_mira**;
float **angle_vertical**;
float **angle_horitzontal**;
float **radi**;
float **alçada**;
	punt_de_mira, **radi** i **alçada** es calculen quan es carrega un entorn tal i com s'ha comentat
	angle_horitzontal = 270º
	angle_vertical = 90º

Quan es detecta algun dels inputs per controlar la càmera, es procedeix com continua:

- **Tecla W**:

punt_de_mira.x = punt_de_mira.x - cos(**angle_horitzontal**)
punt_de_mira.y = **punt_de_mira**.y - sin(**angle_horitzontal**)

- **Tecla A**:

punt_de_mira.x = **punt_de_mira**.x - cos(**angle_horitzontal** + 90º)
punt_de_mira.y = **punt_de_mira**.y - sin(**angle_horitzontal** + 90º)

- **Tecla S**:

punt_de_mira.x = **punt_de_mira**.x + cos(**angle_horitzontal**)
punt_de_mira.y = **punt_de_mira**.y + sin(**angle_horitzontal**)

- **Tecla D**:

punt_de_mira.x = **punt_de_mira**.x + cos(**angle_horitzontal** + 90º)
punt_de_mira.y = **punt_de_mira**.y + sin(**angle_horitzontal** + 90º)

Les fórmules anteriors també s'apliquen quan es desplaça el ratolí per l'escenari amb la **tecla Ctrl i el botó esquerre del ratolí premut**.

- **Botó esquerre ratolí + moviment ratolí**: Per a que el moviment de la càmera segueixi el ratolí, dividim l'ample i alt de la pantalla en 180º, de manera que desplaçar el ratolí amb el botó esquerre premut per tot el llarg de la pantalla suposi una rotació de 180º. Les

components horitzontal i vertical del desplaçament del cursor es guarden a **desplaçament_horitzontal** i **desplaçament_vertical**

angle_horitzontal = **angle_horitzontal** - **desplaçament_horitzontal** / (180*ample_pantalla)
angle_horitzontal = **angle_horitzontal** - **desplaçament_vertical** / (180 * alt_pantalla)

Es limita la posició de la càmera a l'espai per sobre el pla de l'entorn

- **Rodeta del ratolí**: Guardem els graus de rotació de la rodeta del ratolí a **rotacio**.

zoom = **zoom** + **rotacio**

Es limita el zoom als plans proper i llunyà del volum de visualització, de manera que l'entorn no quedi fora d'aquest.

Per últim, es calculen els paràmetres que determinen la posició, orientació i vector amunt de la càmera de la manera següent:

punt_de_mira ja està calculat.
posicio.x = **punt_de_mira**.x + (**radi** - **zoom**) * cos(**angle_horitzontal**)*cos (**angle_vertical**)
posicio.y = **punt_de_mira**.y + (**radi** - **zoom**) * sin(**angle_horitzontal**) * cos (**angle_vertical**)
posicio.z = **punt_de_mira**.y + (**radi** - **zoom**) * sin(**angle_vertical**)
vector_amunt.x = -cos(**angle_horitzontal**)*sin(**angle_vertical**)
vector_amunt.x = -sin(**angle_horitzontal**)*sin(**angle_vertical**)
vector_amunt.x = cos(**angle_vertical**)

Recordem que la variable **radi** guarda la longitud de mitja diagonal del pla de l'entorn.

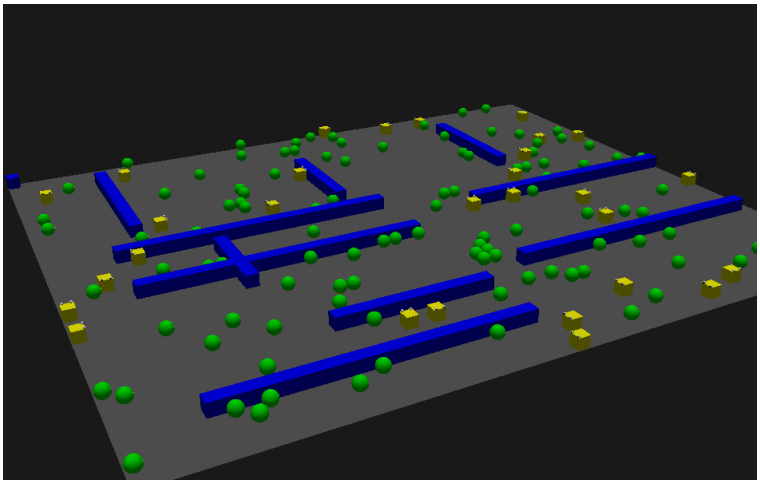

Figura 4.16: Visualitzció de l'entorn.

Per tal de seleccionar individus, quan fem click amb el ratolí sobre algun punt de la pantalla de visualització, es tradueix la posició del cursor a les coordenades tridimensionals corresponents al punt de la superfície de l'objecte que hi havia a sota el cursor. Per fer això, obtenim les matrius de projecció, de model i de finestra i guardem la informació de profunditat del píxel que hi havia sota el ratolí amb la funció *glReadPixel()* [13] de *OpenGL*. Un cop tenim les coordenades de finestra del punt en qüestió, calculem les coordenades del punt en el model amb la funció *glUnProject()* [14] de *OpenGL*. Les coordenades del punt tindran tres components, X, Y i Z. Per a seleccionar un individu només cal buscar quin individu es troba a la posició (X, Y).

Apareixerà una bombolla rosada al voltant de l'individu indicant que està seleccionat.

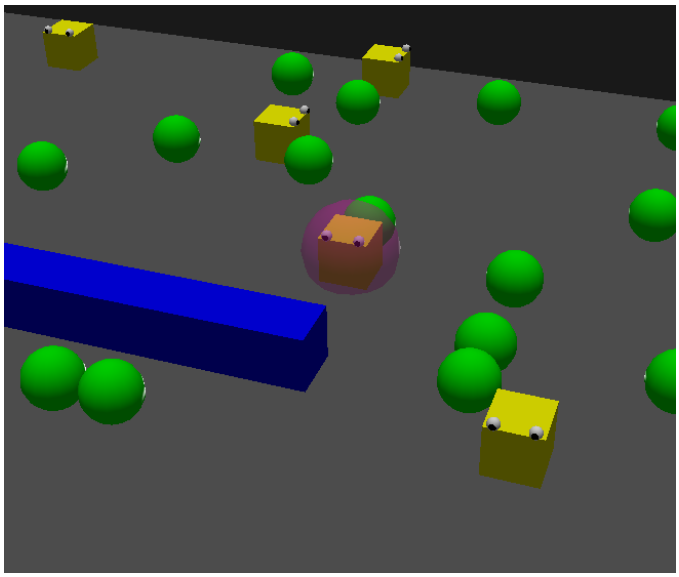

Figura 4.17: Individu seleccionat. Observem una esfera rosada per indicar-ho.

4.6.3 CONTROL DE LA SIMULACIÓ

El control de la simulació apareix sota el visualitzador i ofereix les funcionalitats següents:

Figura 4.18: Opcions per al control de la simulació.

1) Permet activar i desactivar la visualització.

2) Desaccelera la simulació multiplicant per dos el període entre iteracions. Només té efecte si la visualització està activada,

3) Inicialitza una simulació o la reemprèn si ja havia començat però s'havia parat / Pausa una simulació en marxa.

4) Finalitza una simulació en marxa.

5) Accelera la simulació dividint per dos el període entre iteracions. Només té efecte si la visualització està activada.

6) Mostra el percentatge de simulació que s'ha dut a terme. Si es deixa el ratolí quiet sobre la barra de progrés, es mostrarà exactament el número d'iteracions que s'han simulat. Si el màxim d'iteracions està fixat a zero (infinit), la barra de progrés no avança.

7) Activa les etiquetes d'individu. Això significa que al visualitzador es mostrarà al costat de cada individu el seu identificador, la seva energia, la seva edat i el número d'ancestres. Sobre les fonts d'energia s'hi mostrarà la quantitat d'energia que els queda.

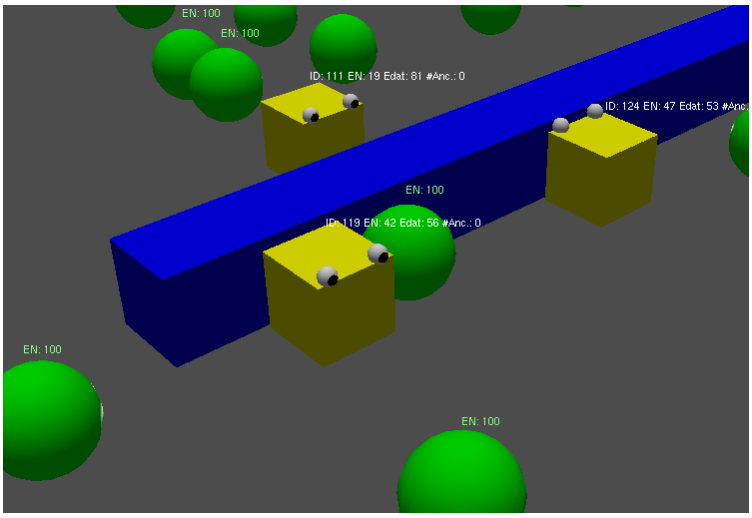

Figura 4.19: Es pot activar la informació sobre individus i fonts d'energia en la finestra de visualització tridimensional.

8) Mostra / amaga el mòdul informació sobre l'entorn, que es descriu en l'apartat següent.

4.6.4 INFORMACIÓ SOBRE L'ENTORN

En aquesta part es mostra la informació i opcions següents:

1) Mostra informació de tot l'entorn i el individus en temps real, és a dir, s'actualitza a cad iteració.

1.1) Nombre d'individus vius a l iteració actual.

1.2) Mitja d'edat dels individus vius la iteració actual.

1.3) Mitja d'energia dels individu vius a la iteració actual.

1.4) Energia total de l'entorn, és dir, suma de les energies de tots els individus també de les seves energies residuals i de la de le fonts d'energia. Encara que aquest paràmetre é immutable i està fixat per l'usuari, es calcula a cad iteració i es mostra per detectar possibles errors.

2) Activa i desactiva la coloració del individus en funció de diferents paràmetres. E detall dels criteris de coloració que s'han seguit n s'explica ja que aquesta funcionalitat ha est desenvolupada pel Nicolau en el seu projecte.

2.1) Coloreja els individus en funci del número d'iteracions des que van ser creats.

2.2) Coloreja els individus en funci del seu nivell d'energia.

2.3) Coloreja els individus en funci del nombre d'ancestres. Quan el simulador gener un individu perquè el número d'individus és inferi al llindar fixat per l'usuari, es considera que té ancestres.

2.4) Coloreja els individus en funci del número d'iteracions des de la creació del se primer ancestre. Evidentment, si no té ancestres aquest paràmetre coincideix amb el temps de vida

2.5) Coloreja els individus en funci de l'identificador del seu últim ancestre i del núme d'ancestres.

Figura 4.21: Part d'interfície corresponent a la informació de l'entorn i els individus.

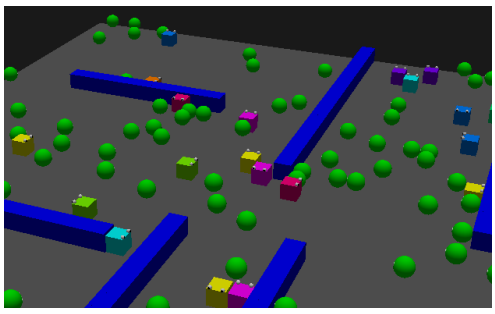

Figura 4.22: Individus pintats amb diferents colors en funció del seu últim ancestre i del número d'ancestres.

4.6.5 INFORMACIÓ I OPCIONS DE L'INDIVIDU

Aquesta finestra només es mostra quan hi ha un individu seleccionat, i tota la informació i opcions que s'hi presenten estan relacionades amb el mateix. Consta de dues pestanyes:

- **Informació**: Si mostra de dalt a baix, l'identificador de l'individu seleccionat, e número d'teracions des de la seva creació, el número d'iteracions des de la creació del seu últim ancestre, el nombre d'ancestres i l'identificador de l'últim ancestre. A continuació es mostra l'algorisme de l'individu, i es ressalta en vermell la instrucció que s'ha executat en aquesta iteració. En els casos en què no s'hagi trobat cap instrucció de tipus Set en el flux d'iteració és mostrarà la línia de codi que té el número de línia igual al PC.

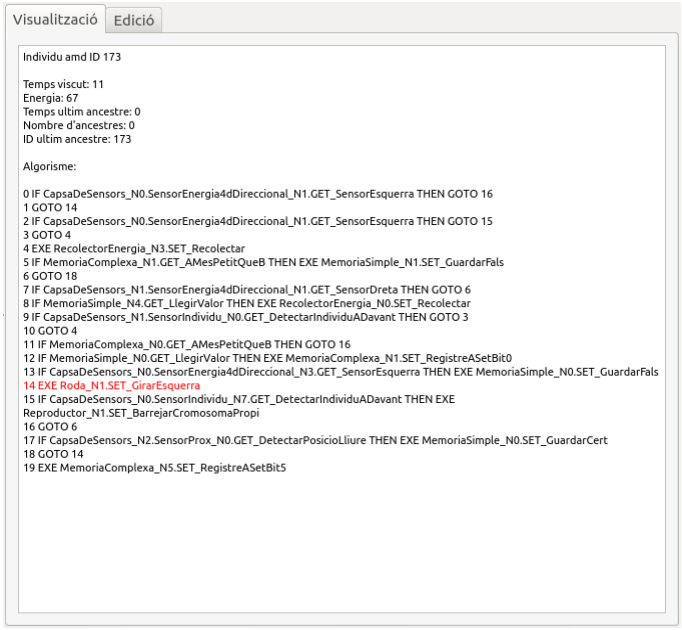

Figura 4.23: Pestanya corresponent a la Informació de l'individu seleccionat.

- Editor: aquest menú mostra l'editor d'algorismes. Aquest editor permet seleccionar i modificar una línia al mateix temps. Es divideix en tres parts:

1) Porció de codi anterior a la línia que s'està editant.

2) Línia que s'està editant opcions d'edició. Es detalla la funcionalitat d'aquest menú a l'apartat següent.

3) Porció de codi posterior a la línia en edició.

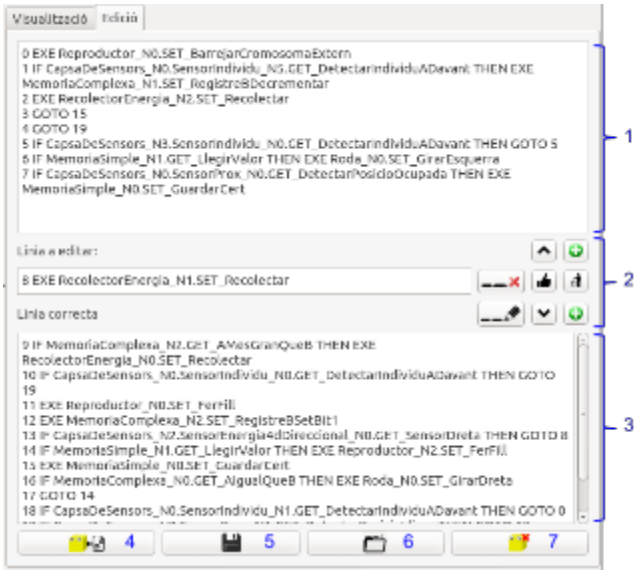

Figura 4.24: Editor d'algorismes.

A més permet assignar l'algorisme a l'individu seleccionat (4) o guardar-lo e un fitxer extern (5). També es pot obrir u algorisme guardat en el sistema d'arxius (6) per tal d'editar-lo o assignar-lo l'individu actual, i eliminar l'individ seleccionat (7).

4.7 EDITOR D'ALGORISMES

L'editor d'algorismes permet seleccionar i editar una línia. Per editar una línia, es divideix en unitats d'informació que de dividir-se no tenen cap sentit. A continuació es mostra una línia dividida segons aquest criteri. Cada un d'aquests elements l'anomenarem **unitat d'informació**.

8	EXE	RecolectorEnergia_N1	SET_Recolectar

L'editor es basa en treure i posar unitats d'informació fins que la línia sigui correcta o no es puguin treure més elements. Per a gestionar l'edició de línies tenim els següents elements:

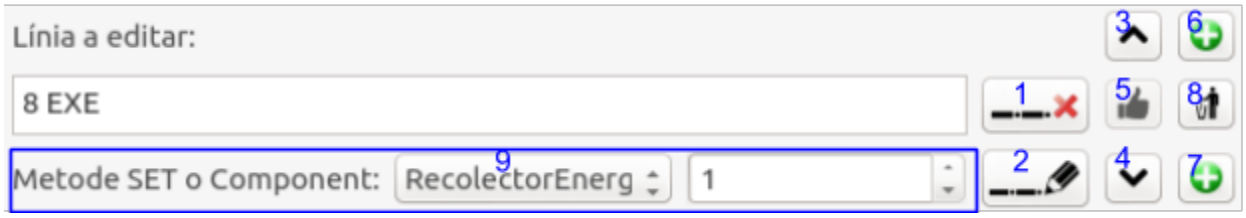

Figura 4.25: Editor de línies de codi.

1) Elimina una unitat d'informació. Es mostren totes les opcions correctes per tal de completar la línia.

2) Si la línia no està completa, es mostren totes les opcions possibles per completar-la. Amb aquesta opció afegim una unitat d'informació.

3) Selecciona la línia anterior, sense guardar els canvis fets a la línia actual.

4) Selecciona la línia posterior, sense guardar els canvis fets a la línia actual.

5) Valida la línia actual. És a dir, la substitueix en el codi. Només està habilitat quan la línia està completa.

6) Afegeix una línia al davant de la línia actual, a la qual no se li guardaran les modificacions.

7) Afegeix una línia darrere la línia actual, a la qual no se li guardaran les modificacions.

8) Elimina la línia actual i selecciona la línia posterior per editar. Si no hi ha línia posterior, selecciona l'anterior. Si és l'única línia del codi no es podrà eliminar.

9) Aquesta part varia en funció de la informació que pugui completar la línia. Per decidir si la línia està completa o quina informació s'ha de mostrar a cada moment es segueix el següent algorisme:

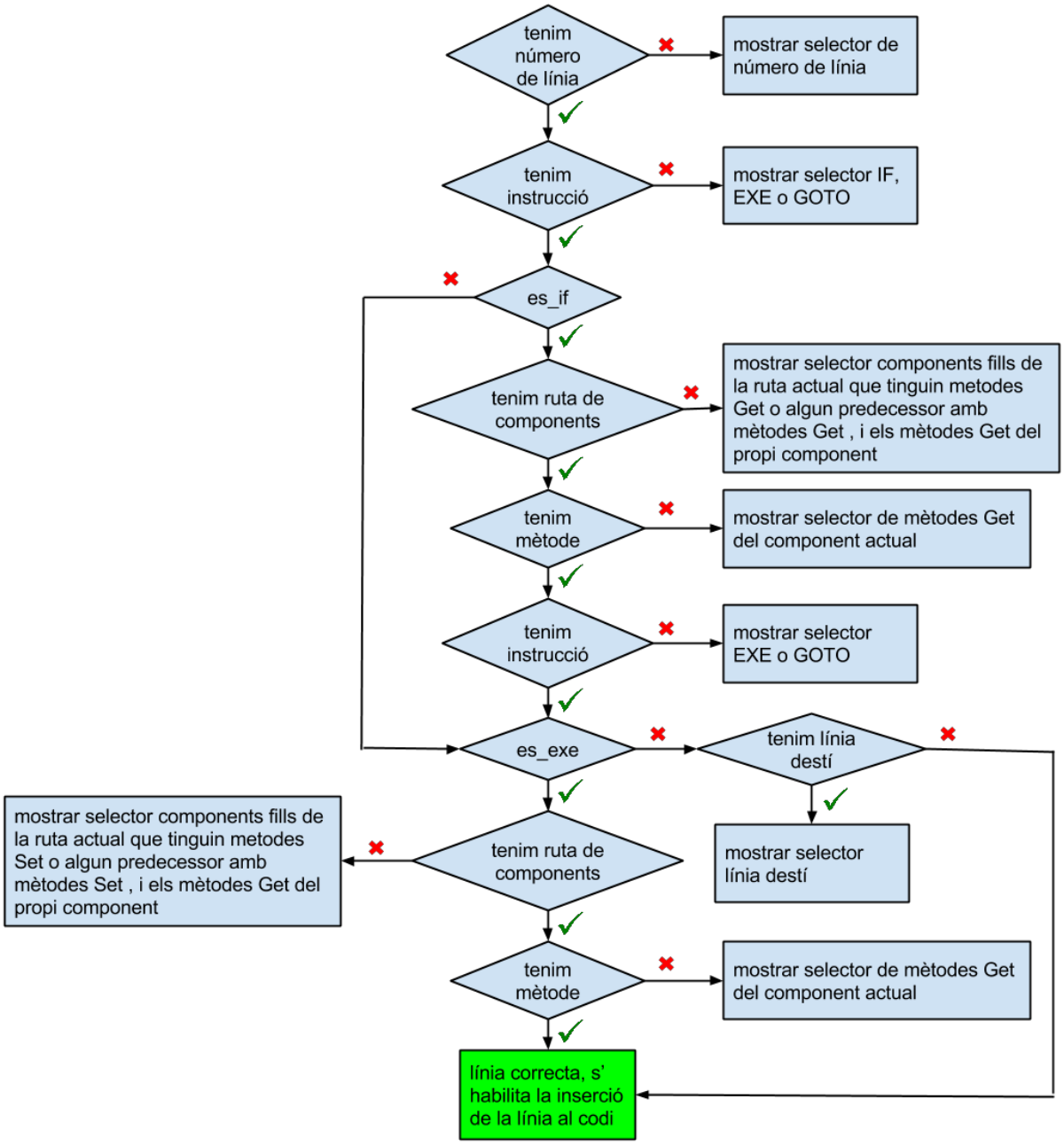

Figura 4.26: Diagrama de flux de l'algorisme que determina les opcions que s'han de mostrar a l'usuari per continuar la línia.

Com veiem, quan la línia no està completa s'han de mostrar les opcions correctes que poden anar a continuació. Aquestes opcions són unitats d'informació i es poden dividir en tres categories:

- **Número de línia**: Tant si la línia està buida com si trobem un GOTO, necessitem un número de línia a continuació. En aquest cas deixarem escollir a l'usuari qualsevol número entre 0 i 65535, que és el valor màxim que pot prendre un *unsigned short*, el tipus de la variable on es

guarda el número de línia. En ambdós casos no s'ha de restringir el número a cap interval a part del mencionat.

- **Instrucció**: Si la línia només té el número de línia, qualsevol de les tres instruccions, IF, EXE i GOTO és una opció vàlida. Si en canvi ja hi trobem un IF i una condició, només les instruccions EXE i GOTO poden completar la línia.

- **Condició**: Aquest és el cas més complex. Quan no hi ha una ruta de components o aquesta està incompleta, s'han de mostrar a l'usuari els mètodes tipus *Get* del component que figura a la línia i també els fills d'aquest component que tenen mètodes *Get* o poden tenir predecessors amb mètodes *Get*. És a dir, suposem el domini següent:

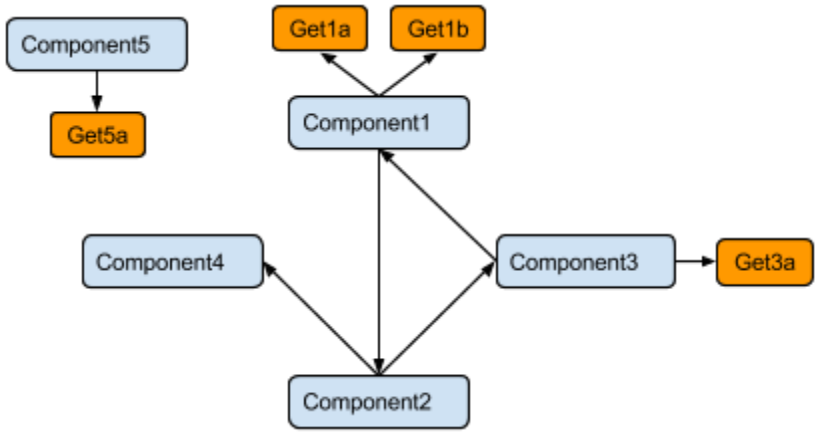

Figura 4.27: Exemple de domini de components.

Les fletxes entre components indiquen possibles relacions pare-fill.

Si a la línia actualment hi ha el *Component1*., la següent unitat d'informació pot ser *Get1a*, *Get1b*, i també *Component2*, ja que pot tenir un predecessor amb mètodes Get, en aquest cas el *Component3*. Per tant, el desplegable hauria de mostrar les opcions *Get1a*, *Get1b* i *Component2*. El graf següent mostra en vermell els camins que porten a mètodes Get o components amb mètodes Get desde *Component1*.

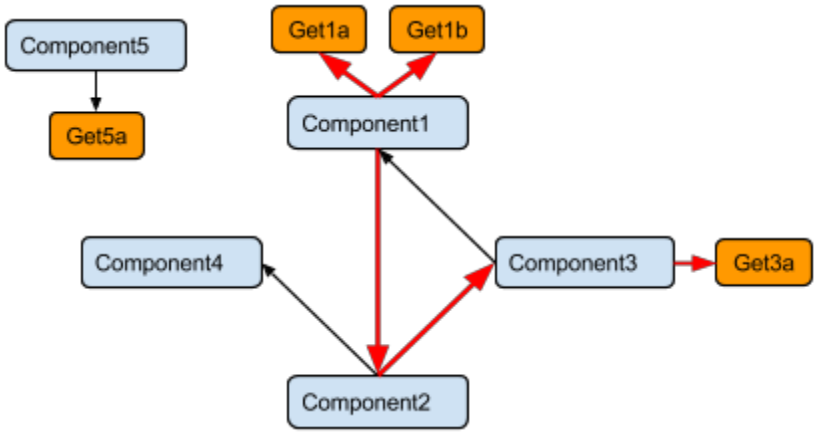

Figura 4.28: Camins des de *Component1* fins a mètodes *Get*.

Les dues primeres opcions donarien la part de la línia corresponent a la condició per conclosa, però si l'usuari selecciona la tercera opció, l'única opció que es donarà per continuar la línia serà *Component3*.

L'algorisme que, donat un node, ens permet trobar els fills d'aquest node amb mètodes *Get* o que poden tenir un predecessor amb un mètode *Get* és el següent:

Suposem que **node_actual** és el node del qual volem trobar fills amb mètodes *Get* o amb predecessors amb mètodes *Get*. Utilitzarem les llistes següents:

- **fills**: on guardarem els identificadors dels fills directes de node actual.
- **flags_fills**: que és una llista de *booleans* de la mateixa longitud que fills. Cada posició d'aquesta llista serà un *flag* del node de **fills** que es troba a la mateixa posició.
- **nodes_amb_metode**: on guardarem els identificadors dels nodes amb mètodes *Get* o que poden tenir nodes predecessors amb mètodes *Get*.
- **nodes**: on guardarem els identificadors de tots els nodes del graf.

També farem servir un valor, **mida_nodes**, que guardarà la mida de la llista nodes.

D'entrada, les llistes estan buides. Els passos a seguir són els següents.

1) Guardem els identificadors dels fills de **node_actual** a **fills**.

2) Creem la llista **flags_fills** de la mateixa mida que **fills** i inicialitzem tots els seus valors a Fals.

3) Posem els identificadors de tots els nodes a **nodes**, excepte el de **node_actual**.

4) Guardem la mida de la llista **nodes** a **mida_nodes**.

5) Posem tots els nodes de la llista **nodes** que tenen mètodes *Get* o que són pares d'algun node de la llista **nodes_amb_metode** a la llista **nodes_amb_metode**.

6) Comprovem quins nodes de **fills** que no tinguin el seu *flag* corresponent a Cert es troben a **nodes_amb_metode**, i posem el seu *flag* corresponent a Cert.

7) Si la mida de la llista **nodes** és igual a **mida_nodes** o tots els *flags* de la llista **flags_fills** estan a Cert, parem l'execució de l'algorisme i retornem el vector **flags_fills**. Si no continuem l'execució en el punt 4.

La llista **flags_fills**, doncs, ens indica quins fills de node actual han de ser seleccionables per l'usuari per tal de continuar la línia.

S'ha de considerar un cas particular, que és quan a la línia no s'ha començat a escriure la ruta del component que té el mètode de tipus *Get*. En aquest cas, doncs, mostrarem com a possibles opcions tots els components amb mètodes tipus *Get* o que poden ser predecessors de components amb aquest tipus de mètode. Per trobar-los, s'afegeix un node arrel, pare de tots els components, i s'aplica l'algorisme ja mencionat sobre aquest node arrel.

Quan la instrucció que falta a la línia és de tipus *Set*, és a dir, després d'un EXE, la metodologia per buscar les possibles opcions per continuar la línia és la mateixa, però en

aquest cas es busquen els fills del component actual que tenen mètodes *Set* o que poden ser predecessors de components amb mètodes *Set*.

Vegem el funcionament global de l'editor de línies amb el domini descrit a l'apartat **4.2.1**. Partim de la següent situació:

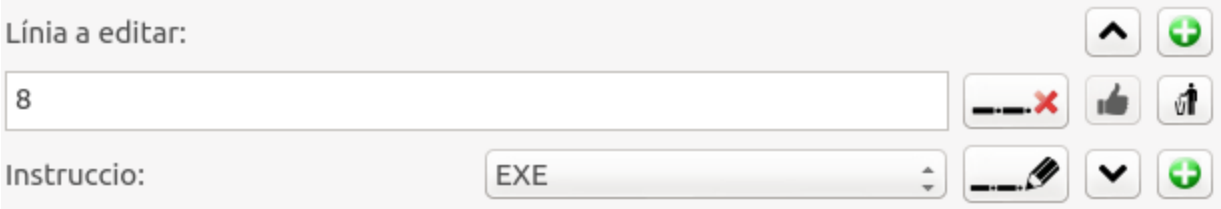

Figura 4.29: Estat de l'editor de línies quan la línia només té el número de línia.

En aquest punt, l'única unitat d'informació que hi ha a la línia a editar és el número de línia. Per tant, es mostra un desplegable amb les opcions IF, EXE i GOTO. Si ara afegim la unitat d'informació EXE, ens trobarem a la següent situació:

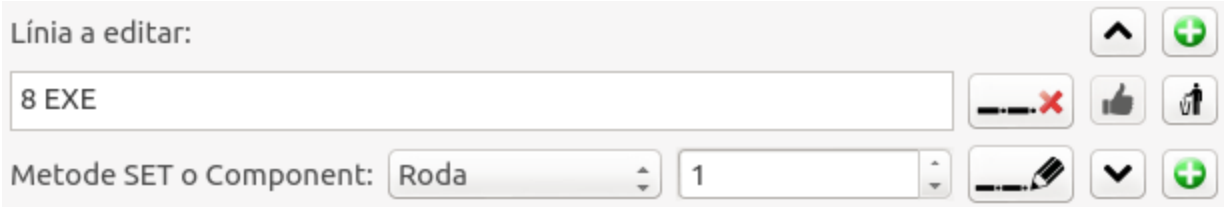

Figura 4.30: Estat de l'editor de línies quan la línia té el número de línia i una instrucció.

En aquest cas, tenim el número de línia i una instrucció de tipus EXE. Per tant, ara es mostraran els components amb instruccions de tipus *Set* o que poden tenir predecessors amb mètodes de tipus *Set*. En aquest cas, el desplegable mostra les opcions següents: *Roda*, *Reproductor*, *RecolectorEnergia*, *MemoriaSimple* i *MemoriaComplexa*. Observem que no s'hii mostra el component CapsaDeSensors, ja que no té mètodes de tipus *Set* ni cap dels seus possibles predecessors en tenen. El número que acompanya el component serà el seu índex.

Si afegim la unitat d'informació *Roda*, ens trobarem a la següent situació:

Figura 4.31: Estat de l'editor de línies quan la línia en edició té el número de línia, una instrucció i el primer element de la ruta del component amb el mètode.

L'últim element de la línia és un component. Per tant es mostren els components fills d'aquest component i els seus mètodes tipus *Set*. Com que Roda no té components fills, en el desplegable només apareixen els mètodes tipus *Set*, que són: *SET_MoureEndavant*, *SET_GirarDreta* i *SET_GirarEsquerra*.

Finalment afegim la unitat d'informació corresponent al mètode *Set SET_MoureEndavant*, i la línia ja està completa. Apareix un missatge que així ho indica i s'habilita la inserció de la línia al codi.

Figura 4.32: Editor de línies amb una línia completa. El botó per insertar la línia al codi està habilitat.

4.8 MODE VIDEOJOC

La informació de cada nivell de joc s'emmagatzema en una estructura de dades: nivell, fitxer entorn, fitxer domini, descripció, posició inicial, objectiu i ruta de la imatge. Quan es carrega un nivell, es llegeix el fitxer que codifica l'entorn i es mostra per pantalla, amb l'individu a la seva posició inicial i un requadre sobre la posició objectiu, si n'hi ha. També es llegeix el fitxer de domini que es carrega en memòria, i es pausa el joc a l'espera que l'usuari editi l'algorisme d'un o varis individus. A partir que es prem el botó de *play*, s'utilitza un *flag* per especificar que l'usuari no pot tornar a editar cap algorisme, fins que la simulació del nivell no hagi finalitzat.

A cada iteració es comprova si es dóna la condició de finalització del nivell. En cas afirmatiu es para la simulació i es mostra un missatge felicitant l'usuari.

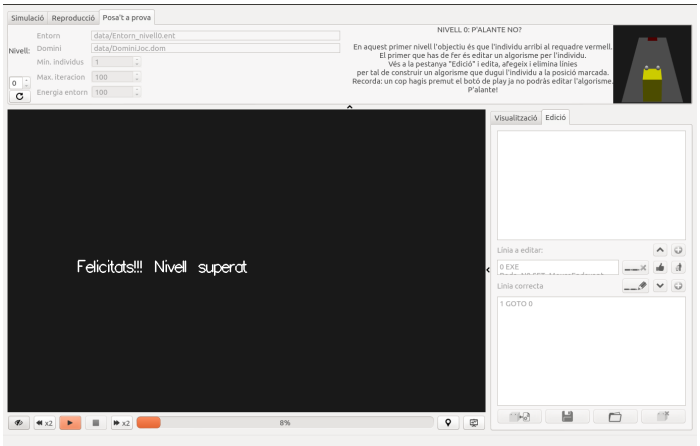

Figura 4.33: Interfície amb el missatge que es mostra quan es supera un nivell del videojoc.

Si s'arriba a l'última iteració i no s'ha donat la condició, simplement es para la simulació.

5 RESULTATS

Hem implementat una aplicació que permet simular el comportament d'un individus en un entorn. L'usuari pot dur a terme aquesta simulació especificant la majoria de paràmetres, tals com el disseny de l'entorn, el domini de components que poden formar els individus, el nombre mínim d'individus a l'entorn, la durada de la simulació i l'energia total de l'entorn. Per tal de fixar el valor d'aquests paràmetres i visualitzar l'entorn, s'ha implementat una interfície, tal i com es pot veure a la figura 5.1.

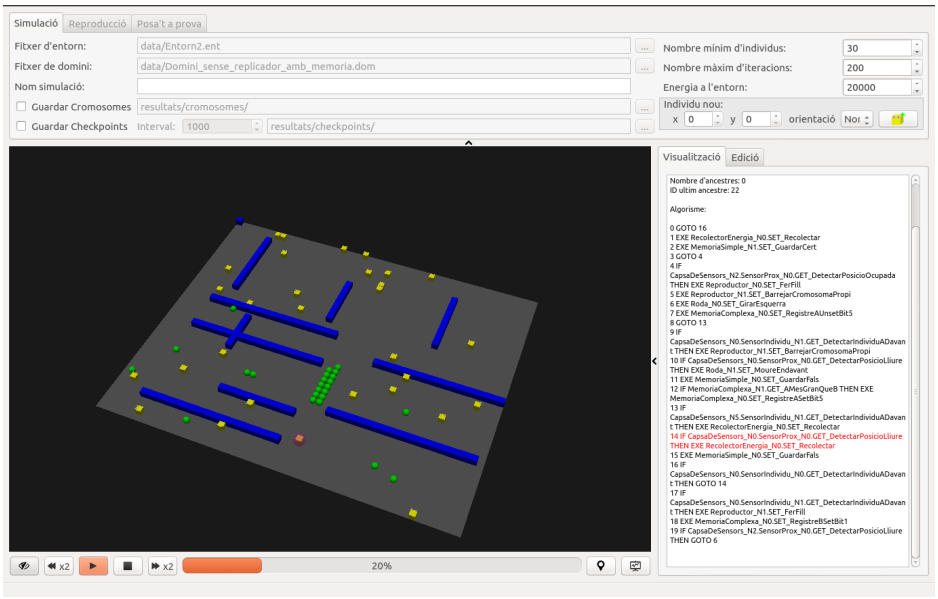

Figura 5.1: Aplicació implementada corrent una simulació.

Es poden seleccionar els individus, i s'ha implementat un mòdul que mostra a l'usuari certs atributs de l'individu seleccionat, així com el seu algorisme i la línia que s'està executant actualment, tal i com es pot veure a la figura 5.2.

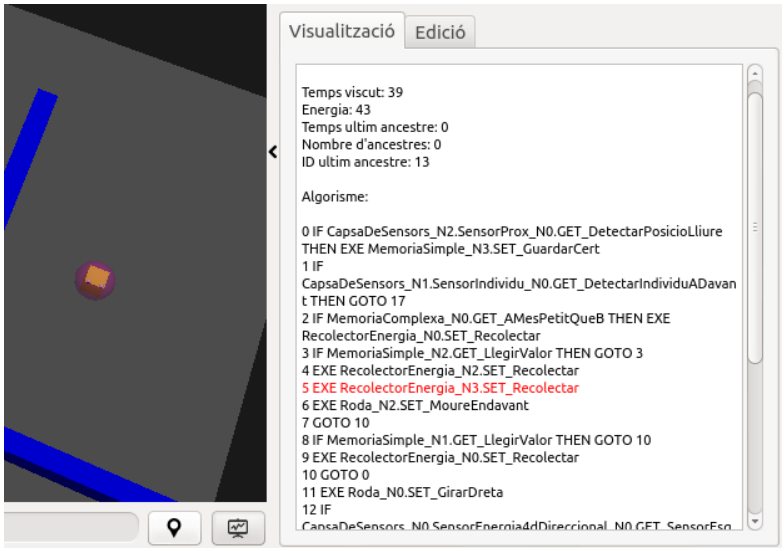

Figura 5.2: Individu seleccionat amb els seus atributs i algorisme.

Es poden afegir i eliminar inidividus, i també editar-ne el comportament. A la figura 5.3 es mostra l'editor de l'algorisme de l'individu seleccionat.

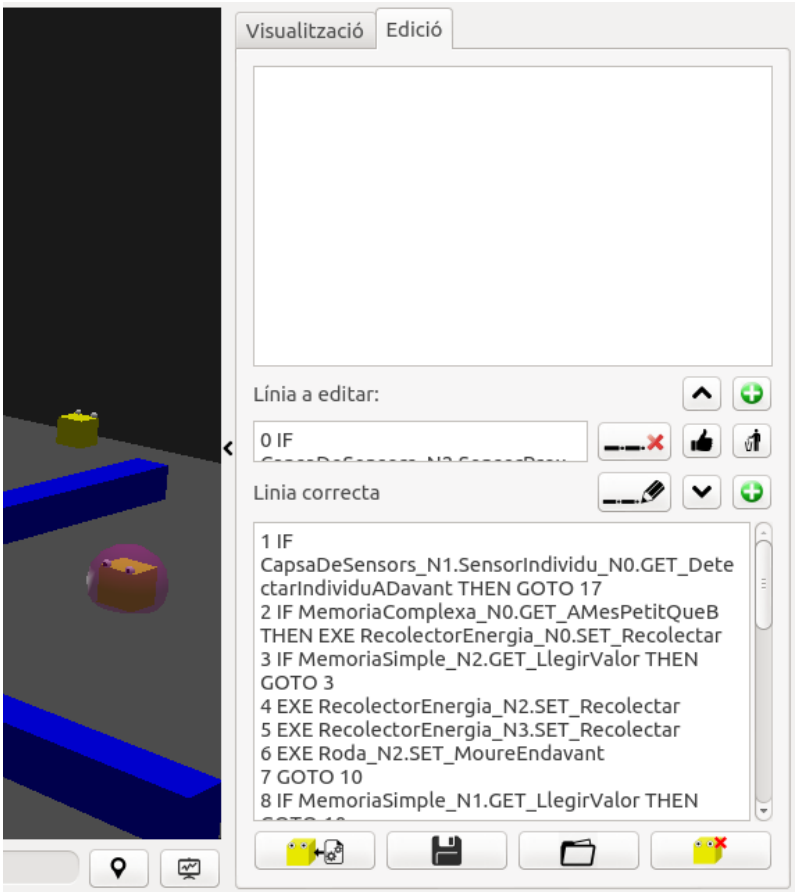

Figura 5.3: Editor de l'algorisme de l'individu seleccionat.

La velocitat de la simulació es pot controlar amb els botons de la figura 5.4, així com també la inicialització, pausa i parada de la simulació, i alguns paràmetres de visualització.

Figura 5.4: Barra de control de la simulació.

A més, s'ha implementat un mode videojoc en què s'han d'assolir uns objectius a través de la programació de l'algorisme d'un individu. A la figura 5.5 es mostra el nivell 0 i, amb vermell, la posició a on ha d'arribar l'individu.

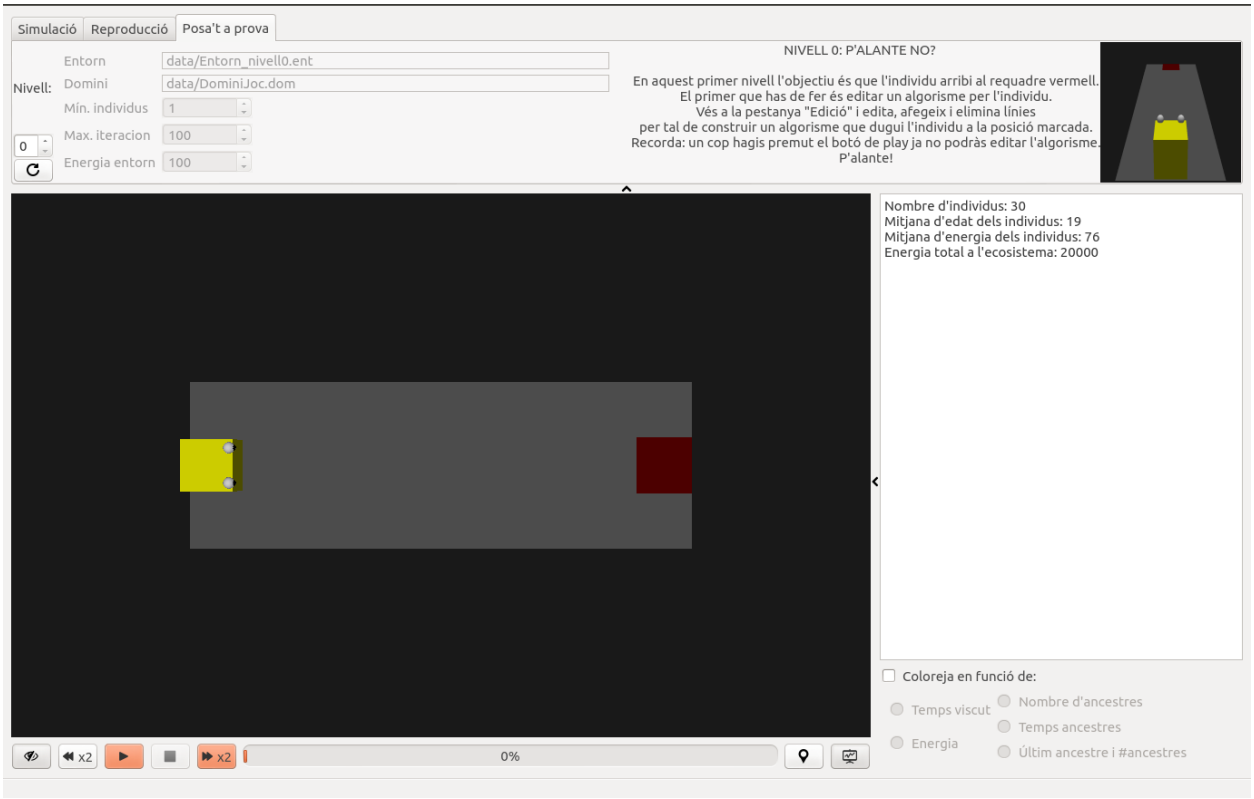

Figura 5.5: Interfície en mode *Videojoc*. L'individu ha d'arribar a col·locar-se sobre el quadrat vermell.

6 CONCLUSIONS

6.1 ACOMPLIMENT DELS OBJECTIUS

A continuació es repassen els objectius i se'n valora l'acompliment un a un.

1) ***Dur a terme la implementació d'una aplicació a partir d'eines molt bàsiques partint pràcticament de zero, sense fer servir llibreries externes, plataformes o abstraccions ja implementades.*** Si bé és cert que s'han implementat la majoria de mòduls, en alguns casos hem utilitzat abstraccions i llibreries externes per tal d'accelerar el procés d'implementació. És el cas de la llibreria gràfica utilitzada, *OpenGL*, i el dissenyador d'interfícies, *Qtdesigner*.

2) ***Es vol implementar una aplicació que sigui capaç de simular el comportament de diferents individus que actuen en un entorn en funció dels seus propis algorismes. Aquesta part constitueix el nucli de l'aplicació i es desenvoluparà en cooperació amb el Nicolau Manubens.*** Aquest apartat s'ha acomplert de forma exitosa

3) ***Es vol que la implementació sigui escalable i flexible, i que la majoria de paràmetres d'una simulació els pugui determinar l'usuari: entorn, número d'individus, etc.*** S'ha tingut en compte aquest objectiu en pràcticament totes les fases de la implementació, la qual cosa ha resultat en una aplicació que permet a l'usuari personalitzar i dissenyar cada simulació. A més, el disseny del domini de components permet dissenyar nous components de forma versàtil, escalant així el programa i, si es plantejés la física necessària, introduïnt components com articulacions o pistons que portarien la diferenciació dels individus no només al terreny algorísmic sinó també a l'estructural. Es tracta aquest tema a l'apartat **6.3**.

4) ***A més, es pretén que l'aplicació sigui còmode, visual i d'interacció intuïtiva. Per això es marquen els subobjectius següents:***

4.1) ***Es dissenyarà i implementarà una interfície amb la qual l'usuari interactui i controli la simulació.*** La interfície s'ha estructurat modularitzant els components de la manera més lògica pel que fa a ús. És a dir, s'ha intentat compactar les opcions per àmbits d'ús similar. Tot i així, falta algún tipus d'orientació a l'usuari per tal de familiaritzar-lo amb la interfície, ja que d'entrada la gran quantitat d'opcions que se li dónen pot resultar confusa.

4.2) ***Es representarà l'entorn en 3D. Es tractarà d'una visualització que permeti una navegació intuïtiva i que compleixi finalitats funcionals i no estètiques.*** En aquest sentit penso que la implementació feta compleix amb els requisits, ja que la visualització és simple però permet diferenciar els diferents elements que s'hi mostren de forma clara i desplaçar el punt de vista de forma intuïtiva.

5) ***Desenvolupar una aplicació que introdueixi l'usuari en el món de la programació i l'algorismia. És a dir, plantejar una sèrie de reptes a l'usuari que s'hagin de solucionar a través del disseny d'algorismes. Per això es plantejen els subobjectius següents:***

5.1) ***El disseny i implementació d'un editor d'algorismes que permeti a l'usuari crear i modificar algorismes en temps d'execució.*** L'editor és una eina que, tal i com es pretenia, resulta útil per al disseny d'algorismes ja que evita crear algorismes no vàlids i orienta l'usuari mostrant-li a cada moment les opcions vàlides amb les que continuar cada línia. Per altra banda, la seva usabilitat no és prou intuïtiva i a més l'edició d'algorismes pot resultar lenta.

5.2) ***El disseny i implementació de nivells de dificultat creixent, i la funcionalitat per poder gestionar els diferents nivells.*** En aquest sentit, la conclusió és que la quantitat de nivells és molt escassa i el salt de dificultat entre ells és elevada. La implementació és un primer enfoc, però per tal que l'eina resulti educativa cal acompanyar més l'usuari durant el procés d'aprenentatge, orientar-lo i que aquest procés sigui més progressiu.

Per últim, voldria fer una valoració d'acompliment d'un objectiu no especificat per redundant, que és desenvolupar un projecte amb el qual es consolidin i concretin un gran ventall de coneixements assolits durant la carrera, i que requerís d'habilitats referents a diversos àmbits relacionats amb la informàtica. En aquest sentit, tinc la sensació d'haver completat un projecte que m'ha ajudat a reafirmar i assentar les habilitats i coneixements adquirits durant el procés formatiu.

6.2 REVISIÓ DE LA PLANIFICACIÓ

La planificació temporal prevista al principi del projecte s'ha seguit en la seva majoria, si bé és cert que s'ha vist lleugerament modificada pel replantejament d'alguns aspectes de la implementació. És el cas de la física, que ha dut molta menys feina del previst, i per tant hem pogut escurçar el temps reservat per a la seva implementació, i el desenvolupament de la interfície gràfica, que s'ha anat modificant i ajustant als replantejaments fins a l'últim moment. En aquest últim cas, doncs, ha requerit de més temps que l'inicialment estipulat.

6.3 FUTURES LÍNIES DE TREBALL

Al tractar-se d'un projecte amb tantes ramificacions, es podrien plantejar línies de treball que suposarien millores substancials en l'experiència de l'usuari en molts sentits. A continuació n'exposarem alguns que ens semblen d'especial rellevància:

- **Millora de l'entorn i el simulador**: Per tal de crear un entorn més realista i reptador és podrien implementar diferents tipus de fonts d'energia, cel·les a diferents alçades per tal de crear relleu, zones de més i menys temperatura, etc. Evidentment, s'hauria de programar un efecte d'aquests factors en els individus per tal que poguessin sortir perjudicats o beneficiats en

funció del seu algorisme i components. A més, el moviment dels individus podria ser continu en lloc de discret.

- **Extensió del codi**: Es podria estudiar incloure noves instruccions al codi per tal de facilitar el disseny d'algorismes, tals com bucles. Altres opcions interessants serien possibilitar el pas de paràmetres als mètodes, modularització del codi (és a dir, donar opció a crear blocs de codi amb una funció específica, com buscar menjar, i que es pogués fer referència a línies d'altres mòduls per tal de continuar amb el flux d'execució), opció a inserir-hi comentaris, etc.

- **Millora de la física i components amb cos**: En la implementació feta, els components dels individus són virtuals. Seria interessant implementar les modificacions necessàries per tal que els components tinguéssin una representació física en l'entorn tridimensional. Així, un individu podria tenir diferents membres units a articulacions i pistons, i moure'ls en funció de l'algorisme. Això requeriria implementar una física molt més realista capaç de calcular col·lisions i forces entre cossos complexes. El disseny d'un individu significaria implementar el seu algorisme i estructurar els diferents components.

- **Millora de la interfície gràfica**: L'objectiu d'aquesta línia de treball seria convertir la GUI en una eina més intuïtiva, còmoda i agradable, redistribuint els botons i opcions i orientant a l'usuari amb missatges o indicacions.

- **Millora de l'experiència de joc**: En aquest sentit es podrien desenvolupar vàries sublínies de treball. D'entrada, s'haurien de crear més nivells de dificultat més progressiva, amb diversos individus a programar i objectius més variats. També es podria implementar un servidor on pujar un individu dissenyat i programat per l'usuari on competís amb altres individus d'altres usuaris.

REFERÈNCIES

[1] http://scratch.mit.edu - Projecte Scratch. Última visita 8 de setembre de 2014.
[2] http://www.alice.org - Projecte Alice. Última visita 8 de setembre de 2014.
[3] http://smallbasic.com - Projecte SmallBasic. Última visita 8 de setembre de 2014.
[4] http://www.turtleacademy.com/ - Turtle Academy. Última visita 8 de setembre de 2014.
[5] http://u3d.as/content/nick-aliferopoulos/leda-runtime-scripting-language/2ZP - Projecte LEDA. Última visita 8 de setembre de 2014.
[6] http://repast.sourceforge.net/repast_simphony.php - Projecte Repast. Última visita 8 de setembre de 2014.
[7] "Resolución de 8 de febrero de 2013, de la Dirección General de Empleo, por la que se registra y publica el IX Convenio colectivo de Telefónica Telecomunicaciones Públicas, SA.", Boletín oficial del estado, Núm 43, pàgs 15724-15760, Febrer 2013.
[8] http://en.wikipedia.org/wiki/Gouraud_shading - Wiki sobre la il·luminació tipus Gouraud. Última visita 8 de setembre de 2014.
[9] https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml - Referència de glShadeModel(). Última visita 8 de setembre 2014.
[10] https://www.opengl.org/sdk/docs/man2/xhtml/gluPerspective.xml - Referència de gluLookAt(). Última visita 13 de setembre de 2014.
[11] https://www.opengl.org/sdk/docs/man2/xhtml/gluLookAt.xml - Referència de la funció gluLookAt(). Última visita 13 de setembre de 2014.
[12] http://ca.wikipedia.org/wiki/Trigonometria#Teorema_del_sinus - Teorema del sinus. Última visita 13 de setembre de 2014.
[13] https://www.khronos.org/opengles/sdk/docs/man/xhtml/glReadPixels.xml - Referència de glReadPixel(). Última visita 13 de setembre de 2014.
[14] https://www.opengl.org/sdk/docs/man2/xhtml/gluUnProject.xml - Referència de gluUnProject(). Última visita 13 de setmbre de 2014.

ANNEX 1 SOLUCIONS DEL VIDEOJOC

A continuació s'exposen unes possibles solucions per a cadascun dels nivells del videojoc.

- **Nivell 0**:

```
0 EXE Roda_N0.SET_MoureEndavant
1 GOTO 0
```

- **Nivell 1**:

```
0 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN GOTO 3
1 EXE Roda_N0.SET_MoureEndavant
2 GOTO 0
3 EXE Roda_N0.SET_GirarDreta
4 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN GOTO 7
5 EXE Roda_N0.SET_MoureEndavant
6 GOTO 4
7 EXE Roda_N0.SET_GirarDreta
8 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN GOTO 11
9 EXE Roda_N0.SET_MoureEndavant
10 GOTO 8
11 EXE Roda_N0.SET_GirarDreta
12 EXE Roda_N0.SET_MoureEndavant
11 EXE Roda_N0.SET_GirarDreta
12 EXE Roda_N0.SET_MoureEndavant
```

- **Nivell 2**:

```
0 EXE Roda_N0.SET_GirarDreta
1 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN GOTO 4
2 EXE Roda_N0.SET_MoureEndavant
3 GOTO 1
4 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDavant THE
GOTO 6
5 GOTO 8
6 EXE RecolectorEnergia_N0.SET_Recolectar
7 GOTO 4
8 EXE Roda_N0.SET_GirarDreta
9 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDavant THE
GOTO 11
10 GOTO 13
11 EXE RecolectorEnergia_N0.SET_Recolectar
12 GOTO 9
```

```
13 EXE Roda_N0.SET_MoureEndavant
14 EXE Roda_N0.SET_GirarEsquerra
15 GOTO 1
```

- **Nivell 3**: En el cas del quart nivell (és a dir, el 3), i donat que les posicions en les que apareixen les fonts d'energia són aleatòries, no es pot garantir que l'algorisme que a continuació es proporcioni condueixi a completar el nivell, tot i que en la majoria de casos serà així.

```
0 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDavant THEN
GOTO 9
1 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorEsquerra THEN
GOTO 5
2 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDreta THEN GOTO
8
3 IF CapsaDeSensors_N0.SensorEnergia4dDireccional_N0.GET_SensorDarrere THEN
GOTO 7
4 GOTO 15
5 EXE Roda_N0.SET_GirarEsquerre
6 GOTO 9
7 EXE Roda_N0.SET_GirarDreta
8 EXE Roda_N0.SET_GirarDreta
9 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN GOTO 12
10 EXE Roda_N0.SET_MoureEndavant
11 GOTO 9
12 EXE RecolectorEnergia_N0.SET_Recolectar
13 EXE Replicador_N0.SET_FerFill
14 GOTO 0
15 IF CapsaDeSensors_N0.SensorProx_N0.GET_DetectarPosicioOcupada THEN EXE
Roda_N0.SET_GirarDreta
16 EXE Roda_N0.SET_MoureEndavant
```

RESUM

Aquest projecte tracta la implementació d'una aplicació capaç de simular el comportament d'uns individus que segueixen el seu propi algorisme en un entorn que planteja certes dificultat per a la supervivència: hi ha obstacles i fonts que proveeixen energia que els individus necessiten per existir. S'han desenvolupat les eines necessàries perquè l'usuari pugui visualitzar, recollir dades i interactuar amb l'entorn d'una forma còmoda i intuïtiva.

Sobre aquesta base, s'ha implementat un videojoc que planteja reptes a l'usuari que es poden resoldre a través de l'edició dels algorismes d'un o més individus, és a dir, a través de la programació. D'aquesta manera s'introdueix a l'usuari en els principis de l'algorismia d'una forma lúdica i molt visual.

RESUMEN

Este proyecto trata la implementación de una aplicación capaz de simular el comportamiento de unos individuos que siguen su porpio algoritmo en un entorno que plantea ciertas dificultades para la supervivencia: hay barreras y fuentes que proveen energia que los individuos necesitan para existir. Se han implementado las herramientas necesarias para que el usuario pueda visualizar, recoger datos y interactuar con el entorno de una manera cómoda e intuitiva.

Sobre esta base, se ha implementado un videojuego que plantea retos al usuario que pueden ser resueltos a través de la edición del algoritmo de uno o más individuos, es decir, a través de la programación. De este modo se inicia al usuario en los principios de la algorismia de un modo lúdico y muy visual.

ABSTRACT

This project deals with the implementation of an application capable of simulating the behavior of some individuals who follow their own algorithm in an environment that poses certain difficulties for survival: barriers and there are sources that supply energy that individuals need to exist. Tools necessary for the user to view, collect data and interact with the environment in a comfortable and intuitive way have been developed.

On this basis, we have implemented a viedogame that poses challenges to the user that can be solved by editing the algorithms of one or more individuals, that is, through programming. This introduces the user on the algorithms principles in a very visual and entertaining way.