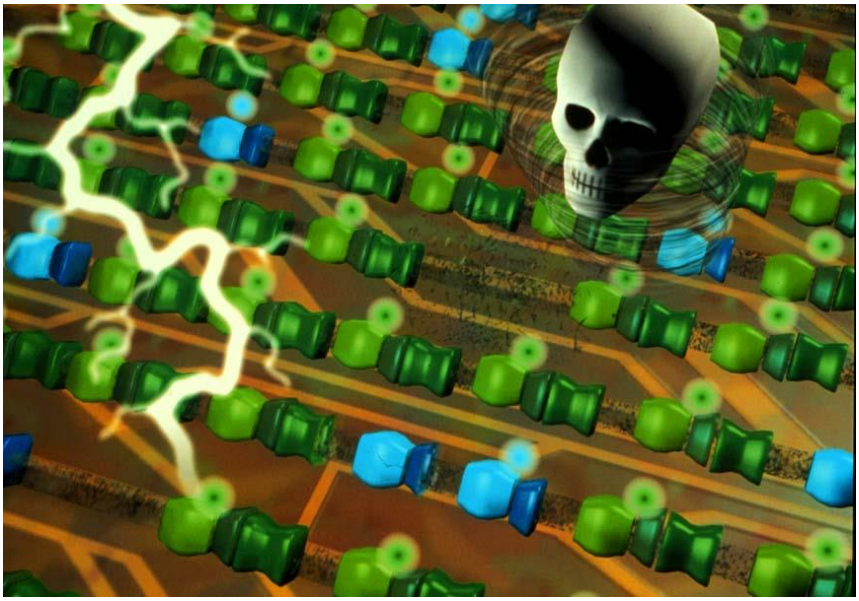
Rapport intermédiaire INF 08

# Evolution naturelle de créatures informatiques




Groupe :

Lucas Clarté

Pierre Descombes

Yueh Lin

Ariane Delrocq

Arthur Migliorini

Tuteur : Nicolas Bredèche

Coordinateur : Marie-Paule Cani

Cadre militaire : Cdt Aubois

# Table des matières

# Motivations

Notre PSC vise à l'élaboration d'un logiciel permettant de simuler informatiquement l'évolution de la vie. Plus précisément, il s'agira de faire évoluer des programmes qui seront l'équivalent informatique d'êtres vivants. L'enjeu du projet est de définir et d'implémenter les fonctionnalités nécessaires pour simuler de manière réaliste l'évolution. En effet, de nombreux projets similaires au notre ont été développés depuis les dernières décennies, mais peu semblent permettre de simuler, même de manière simplifiée, la richesse et la diversité du monde vivant.

Notre projet a deux applications :

→ Tout d'abord, il permettra aux biologistes d'étudier l'évolution de la vie sur Terre sous un autre angle. S'il ne prétend à aucune valeur de preuve, notre système permettra du moins aux chercheurs d'étudier les tendances générales de l'évolution, ses invariants et ses contingences. Par exemple, il est envisageable d'étudier comment la modification des écosystèmes (due à des catastrophes écologiques par exemple) peut entraîner la migration d'espèces et modifier leur évolution. En outre, notre système, en s'affranchissant de toute contrainte physique, pourra permettre de simuler des formes d'espèces vivantes telles qu'il n'y en a pas sur Terre. Ceci pourrait aider les exobiologistes à imaginer l'évolution de la vie sur d'autres planètes que la nôtre.

→ Ensuite, notre système posséderait des applications dans le domaine des algorithmes évolutionnistes. En favorisant l'apparition de programmes de plus en plus complexes, il permettrait l'émergence de solutions originales à des problèmes jusqu'ici mal résolus.

# État de l'art

Le mathématicien John Von Neumann inventa dans les années 1940 le concept d'automates cellulaires, dont le modèle le plus célèbre est le 'jeu de la vie' de Conway. Un modèle d'automates cellulaires est constitué d'une grille dont chaque case peut se trouver dans divers états. L'état d'une case évolue étape par étape en fonction des états des cases voisines selon des règles de transition prédéfinies. A l'aide d'un jeu de règles bien précis, Von Neumann met au point un automate contenant un ensemble de cases -ou cellules- qui se reproduit à l'identique. Une des caractéristiques essentielles de la vie -la capacité d'auto-réplication- pouvait donc s'observer dans le cadre d'entités symboliques et mathématiques, qui ont pu s'incarner dans le monde informatique dans les années qui suivirent. Cette idée fut à l'origine du mouvement de recherches en vie artificielle, qui entend créer des entités informatiques qui pourraient être considérées comme vivantes, c'est à dire pouvant présenter non pas uniquement une reproduction à l'identique mais une reproduction pouvant permettre un phénomène d'évolution.

## 1. Tierra

Le biologiste Thomas Ray, dans les années 1990, fut un pionnier de ce domaine, présentant une simulation informatique du nom de Tierra, se voulant comme une alternative à l'évolution biologique.

Le système Tierra est basé sur la création d'une machine virtuelle dans laquelle vivent les programmes [1] [2] [3]. Un programme est constitué d'une suite d'instructions. L'ensemble des instructions possibles forme un langage qui a été créé spécialement pour Tierra. Il est assez proche d'un langage machine (constitué d'instructions simples usuelles comme and, xor, jmp, add, ifzero, halt, push…) et restreint à 32 instructions différentes qui peuvent donc être codées sur 5 bits. S'il existait trop d'instructions différentes, une trop grande majorité des altérations du code résulteraient en une suite d'instructions n'ayant aucun sens.

Chaque programme possède un CPU virtuel, qui exécute en boucle son code. Afin de simuler l'exécution simultanée de tous les programmes, une tranche de temps du slicer est associée à chacun d'entre eux. Le slicer est une file circulaire, et la taille de la tranche détermine le nombre d'instructions exécutées

par le CPU virtuel du programme correspondant. Cette taille peut être déterminée aléatoirement, ou en fonction de la taille du programme.

La Faucheuse est une file d'attente. Lorsqu'un programme naît, il prend sa place à la fin de la file. Lorsque la mémoire est trop remplie, la Faucheuse efface les programmes en tête de la file. On peut modifier les paramètres pour qu'elle efface des programmes choisis au hasard parmi les x premiers de la file, ou pour que les programmes avancent dans la file lorsque leur exécution génère une erreur.

Un programme, pour se répliquer, doit calculer sa taille, trouver un espace mémoire suffisamment grand, y recopier son propre code, puis allouer un CPU virtuel à son fils, qui se voit alors attribuer une place dans la file de la Faucheuse et dans le slicer. Cependant, les conditions assez libres ont permis l'apparition de programmes parasites qui ne contiennent pas d'exécution de réplication et lisent à la place le code d'un autre programme pour se répliquer.

Au début de la simulation, la mémoire virtuelle est peuplée par plusieurs instances d'un même programme auto-réplicateur écrit par l'expérimentateur et appelé ancêtre. Puis, au fur et à mesure que de nouveaux génomes (suffisamment fonctionnels pour exister chez au moins 3 individus) apparaissent, ils sont enregistrés dans une banque de génomes, et le nombre d'individus possédant chaque génome est lui-aussi mis à jour à chaque nouvelle naissance ou mort. L'expérimentateur a ainsi accès à tous les génomes ayant effectivement vécu dans le monde de Tierra, et peut les analyser pour comprendre leur fonctionnement.

L'ancêtre est constitué de 82 instructions. De nouvelles espèces, notamment de 81, 79, 90 instructions et accomplissant la même fonction sont apparues. Une espèce de programmes parasites comme expliqué ci-dessus sont apparus. En réaction, des espèces résistantes aux parasites est apparue, ainsi que des hyperparasites qui s'accaparent le CPU du parasite qui tentait de les lire.
Une « faune » diversifiée, et dont les espèces s'influencent entre elles, sont apparues spontanément, dans un monde informatique ou les seules contraintes sont celles du CPU, de l'espace mémoire, et la présence de la Faucheuse.

# 2. L'Open Ended Evolution

L'exemple de Tierra provoqua un engouement pour la vie artificielle chez les biologistes comme chez les informaticiens. De nombreux systèmes inspirés de celui de Thomas Ray virent le jour, comme Avida, conçu plutôt pour des applications en biologie, permettant d'évaluer l'influence de conditions précises sur l'évolution, comme les phénomènes de pénurie de ressources, ou les dynamiques proies-prédateurs. D'autres systèmes plus abstraits comme Amoeba ont visé à observer l'étape primordiale de l'évolution de la vie, l'émergence à partir de matière organique d'unités réplicatives.

Cependant, la recherche dans ce domaine ne s'est pas arrêtée là. Partant du constat que de tels systèmes d'évolution artificielle arrivaient toujours à un état stationnaire dans lequel la complexité et la diversité des génotypes n'évoluaient plus (état qui correspond par exemple à un optimum local dans le cas d'algorithmes génétiques), les chercheurs ont cherché les causes de cette stagnation, qui s'oppose à une diversité et une complexité comme on les observe dans la vie sur Terre. C'est le problème de l'open-ended evolution, c'est-à-dire de l'évolution ouverte, sans barrière ni stagnation. Ce problème se pose encore aujourd'hui, et bien que le système Geb 11 12 (qui simule l'évolution de créatures possédant un réseau de neurones et interagissant dans un monde en 2D) réponde à certains critères particuliers de l'open-ended evolution, aucun système ne possède aujourd'hui cette propriété. Sa définition précise demeure d'ailleurs floue. Soros et Stanley 13 ont défini en 2014 quatre conditions nécessaires à l'open-ended evolution :

1. Impossibilité de reproduction pour les individus qui ne vérifient pas un critère minimal, qui doit être non-trivial. En ce qui concerne la vie sur Terre, ce critère est naturellement la possession d'un système reproductif (mitose chez les bactéries, méiose et fécondation chez les espèces sexuées…). En l'absence d'un tel critère, l'évolution s'arrêterait à des organismes triviaux.
2. L'évolution des nouveaux individus doit créer de nouvelles possibilités pour vérifier le critère minimum. En effet, l'évolution s'arrête s'il n'est plus vérifié, et les nouveautés doivent donc le vérifier aussi afin d'être conservées.
3. Les décisions déterminant où et comment les individus interagissent avec l'environnement doivent être prises par les individu eux-mêmes.
4. La limite sur la taille et la complexité des phénotypes des individus doit être *a priori* infinie, c'est-à-dire, plus concrètement, la représentation du génotype ne doit pas limiter l'explosion évolutive attendue de l'*open-ended evolution*.

Les chercheurs continuent de tenter d'obtenir l'open-ended evolution, en la définissant et en cherchant à découvrir les facteurs qui la limitent.

# 3. Algorithmes génétiques

Les algorithmes évolutionnistes s'inspirent eux aussi des principes de l'évolution pour trouver des solutions efficaces à un problème donné. Des programmes répondant de manière inefficace au problème sont écrits, puis modifiés aléatoirement pour passer à la génération suivante. Les programmes obtenus sont alors évalués sur leur capacité à résoudre le problème, et les plus efficaces sont conservés pour la création de la génération suivante. Cette méthode est très employée car elle permet effectivement de trouver des solutions plus efficaces.

Toutefois, le système évolutionnaire des algorithmes génétiques, s'inspirant souvent des recombinaisons chromosomiques, est en général extrêmement rigide, et ne permet pas la puissance expressive de l'évolution biologique, qui permet l'avènement de systèmes et de niveaux d'organisation radicalement différents de ceux du 'programme ancêtre'. Ainsi, le modèle évolutionnaire d'un système 'open-ended' pourrait inspirer l'implémentation de procédure d'algorithmes génétiques plus souples et plus aptes à faire émerger des programmes complexes.

# Notre projet

Nous désirons ainsi créer un système d'évolution virtuelle qui, tout en s'inspirant de Tierra, permettrait de répondre aux critères de l'open-ended evolution.

## 1. Critique de Tierra

Comme nous l'avons vu, Tierra, à l'instar de la plupart des modèles de vie artificielle, ne présente pas une réelle évolution en boucle ouverte. Thomas Ray lui même remarqua que, même en variant le set d'instructions utilisées, l'évolution atteignait toujours un plateau. En effet, l'évolution dans Tierra présentait une course à la simplicité plutôt qu'une course à la complexité : comme tous les programmes disposaient du même temps de lecture par le slicer, leur reproduction était d'autant plus efficace que leur programme était court. La pression de sélection favorisa donc les programmes optimaux au niveau de leur nombre d'instructions. Cette même pression de sélection favorisa les parasites, qui pouvaient se reproduire en utilisant le temps imparti aux autres programmes. Toutefois, contrairement à la course à la complexité, la course à la simplicité est bornée : une fois le programme le plus court pouvant se reproduire atteint (qui peut différer selon le set d'instruction), l'évolution est condamnée à stagner. Le système de Thomas Ray ne peut donc pas concourir au titre de 'système open-ended'.

Nous avons ainsi analysé ce qui selon nous présentait les problèmes majeurs de Tierra à l'aune des quatre critères de Soros et Stanley :

- La présence d'un critère minimal non trivial pour la reproduction est bien garantie, comme nous le prouve l'existence d'un programme de longueur minimale apte à se reproduire. Ce critère est bien plus simple que dans la reproduction terrestre, où la reproduction de l'ADN demande une machinerie complexe, mais la nécessité de pouvoir interpréter le résultat de notre simulation exige de partir d'une base assez simple.

- La possibilité de vérifier ce critère par de nouvelles manières est en partie respectée dans Tierra, en témoigne l'apparition de parasites, qui se reproduisent d'une manière que Thomas Ray n'avait pas pu prévoir. Toutefois, le mécanisme reproductif n'en demeure pas moins très rigide, chaque programme possédant un CPU et une case mémoire bien déterminée.
- L'interaction avec l'environnement est bien permise en partie aux organismes, bien que ceux-ci soient par leur structure même confinés dans une case mémoire précises. La lecture est permise mais pas l'écriture : les programmes doivent faire une demande d'allocation de mémoire.
- Toutefois, comme nous l'avons vu, le dernier critère n'est clairement pas respecté. La complexité des individus est limitée dès le départ par la pression de sélection favorisant les programmes plus courts. La limitation dans la longueur des programmes induit directement une limitation dans la complexité de ceux-ci. En effet, le set d'instruction étant fini, on ne peut réaliser qu'un nombre fini de programmes de longueur bornée.

Nous avons donc décidé de nous attaquer à ce blocage limitant Tierra. A priori, dans le cadre de l'évolution biologique, la réplication de l'ADN se fait environ à la même vitesse au niveau des unités de réplication dans toutes les cellules. Pourtant, les cellules contenant un génome gigantesque ne semblent pas défavorisées par rapport à celles contenant un génome plus court de plusieurs ordres de grandeurs, comme les bactéries primitives. En effet, la réplication de l'ADN est un processus massivement parallèle, le génome d'une cellule est répliqué par de nombreuses unités à la fois. Ainsi, la coordination de multiples unités réplicatives indépendantes garantit une durée de réplication environ constante dans tout le vivant. Nous avons donc envisagé d'introduire du parallélisme dans notre 'Tierra 2.0'.

De plus, lorsque l'on analyse l'émergence de la complexité dans l'évolution biologique, on remarque que le passage à un niveau supérieur de complexité s'accompagne d'une redéfinition de la notion d'individu : on passe ainsi du niveau moléculaire au niveau cellulaire, puis au niveau de l'organisme multicellulaire, et enfin dans certaines espèces sociales comme les fourmis à une société. A chaque évolution, ce qui pouvait être considéré comme un individu indépendant devient un sous-système d'un 'individu' plus vaste. Nous avons donc pensé que Thomas Ray, en prédéfinissant la notion d'individu comme 'un programme contenu dans une case mémoire', bloquait de fait la possibilité d'une telle évolution fractale. Nous avons donc envisagé de brouiller les frontières entre individu, laissant l'évolution elle-même définir quel niveau d'organisation sera pertinent pour analyser le phénomène vivant.

# 2. Description du système Turing-Tierra

Nous avons donc conçu un système qui pourrait se présenter comme la fusion de Tierra et d'une machine de Turing. Nous avons conservé autant que possible les instructions de Tierra, se rapprochant de celles du langage X86 des compilateurs, afin de rester dans l'esprit de Thomas Ray et de conserver une bonne lisibilité. Toutefois, notre univers est constitué d'un bandeau sur lequel sont écrits des instructions, infini en théorie, comme pour une machine de Turing, mais circulaire en pratique. Les CPUs sont similaires aux têtes de lectures des machines de Turing : ils se déplacent sur la bande, et, en fonction des instructions qu'ils lisent, peuvent se déplacer, lire une instruction et la conserver en mémoire, et écrire une instruction (avec une certaine probabilité d'erreur, introduisant par là des possibilités de mutation). Ils possèdent en outre une pile et des registres, pouvant leur servir à stocker des adresses, des instructions ou des variants de boucles.

Les déplacements se font à l'aide d'adresses relatives ou de 'templates', une fonctionnalité déjà présente dans Tierra. Les templates sont des suites des 0 et de 1 présents sur la bande. Lorsqu'un CPU se trouve sur un template, il peut sauter sur le template le plus proche en avant ou en arrière présentant une séquence complémentaire de 0 et de 1. Ce système s'inspire des systèmes de reconnaissance de protéines : par exemple, certaines protéines se fixent sur des séquences complémentaires en amont ou en aval de certains gènes en vue de réguler leur expression. L'utilisation des templates permet une gestion de l'espace plus intrinsèque que les adresses relatives, où les sous modules d'une même unité sont accessibles directement et non pas seulement à partir de leur position dans l'espace physique. La répartition des sous-modules d'une unité peut donc s'adapter à chaque lieu, et n'est pas obligée de suivre une répartition figée.

En outre, ces CPUs peuvent, en fonction des instructions qu'ils lisent, se diviser en deux ou mourir. Toutefois, il serait réducteur de voir en eux les 'structures réplicatives' qui forment la vie. On peut plutôt les interpréter comme des outils, en pensant au parallélisme discuté plus haut lors de la réplication de l'ADN : ils ne sont que des outils, permettant à un même 'organisme' d'effectuer plusieurs tâches à la fois, des tâches qui peuvent être identiques (penser aux multiples unités réplicatives) ou différentiées (penser à la différentiation des cellules d'un organisme pluricellulaire, qui accomplissent

chacune des tâches précises). La véritable unité qui se réplique à l'identique (modulo les mutations) est constituée par une suite d'instruction de la bande, qui doit être a priori recopiée par un ou divers CPUs en divers endroits de la bande. Cette suite d'instructions peut être vue comme le support de l'hérédité, au même titre que le génome dans le monde biologique, tandis que l' 'organisme' est constitué de la suite d'instructions et de l'ensemble des CPUs qui 'travaillent' pour elle. On se rapproche de la vision du Gène égoïste du biologiste Richard Dawkins, selon lequel l'organisme n'est qu'une entité complexe instrumentalisée par le gène en vue de se répliquer efficacement. Ici les CPUs (sur lesquels le code a d'ailleurs droit de vie et de mort) sont des outils temporaires utilisés par les instructions pour se réécrire.

En théorie, l'exécution des multiples CPUs est parallèle. Pour des raisons pratiques, nous faisons effectuer une instruction à chaque CPU dans l'ordre d'une file circulaire de CPUs, si bien qu'au niveau de la longueur de l'organisme primitif (une cinquantaine d'instructions), le fonctionnement semble parallèle. De plus, pour éviter une accumulation de CPUs inutiles créés massivement par une boucle infinie contenant une instruction de division, ce qui ralentirait énormément, voire pourrait bloquer, notre simulation, nous imposons une limite sur la densité de CPUs, nous autorisant à tuer les CPUs s'ils sont trop nombreux en un endroit précis. Un tel phénomène peut s'apparenter à une limitation dans les ressources d'énergie, qui se rencontre aussi dans l'évolution biologique, et ne devrait donc pas empêcher le caractère éventuellement open-ended de notre système.

# 3. Implémentation

Nous avons décidé, sur conseil de notre tuteur, d'implémenter notre système Turing-Tierra en langage python. En effet, ce langage est de niveau assez élevé, donc proche des raisonnements abstraits. Même si sa relative lenteur par rapport au C++ le rend moins efficace pour un développement industriel, son caractère ergonomique le rend préférable pour une activité de recherche comme la nôtre. A la manière de Thomas Ray, nous avons implémenté une machine virtuelle qui simule le comportement de notre 'machine de Turing-Tierra'.

Notre programme comporte une classe principale : Univers, contenant la mémoire, c'est à dire la 'bande' d'instructions, sous forme d'un tableau d'instruction (utilisé comme une file circulaire à l'aide d'effets de bords), ainsi que la liste des CPUs sous la forme d'un tableau d'identifiants, l'ordre dans le tableau définissant l'ordre d'exécution, la liste de leur localisation sous la forme d'un tableau d'adresses, un pointeur indiquant le CPU en cours d'exécution et le taux de mutation courant lors du recopiage des instructions par les CPUs. Cette classe comporte en outre des méthodes permettant d'effectuer les CPUs un à un, de 'tuer' ou de 'faire naître' à une position donnée dans la mémoire un CPU et modifiant la liste de CPUs, en réponse à une instruction 'dead' ou 'new', et une méthode permettant de tuer des CPUs en cas de trop forte densité, comme évoqué plus haut.

La classe CPU comporte ainsi le pointeur indiquant la position du CPU dans la mémoire, un pointeur vers la classe univers (utile pour avoir accès à l'instruction que 'lit' le CPU), et les diverses places où il peut conserver des informations : ses quatre registres et sa pile bornée. La pile fonctionne en outre avec un pointeur indiquant l'élément en cours de traitement dans celle-ci, susceptible de se déplacer pour donner accès aux autres informations stockées dans celle-ci. La classe CPU comporte en outre des méthodes permettant la création d'un nouvel élément de la classe, son déplacement dans la mémoire ou encore la gestion de ses registres et de sa pile.

Les instructions sont stockées dans la mémoire virtuelle (et dans les registres et la pile des CPUs) sous forme de nombres, pour pouvoir être interprétées comme instructions ou utilisées comme valeurs. Cela permet de plus de faciliter l'évolution puisque les programmes peuvent muter les instructions par opérations arithmétiques ou en écrire de nouvelles en assignant une valeur quelconque à une case mémoire. Le nom de l'instruction et le nombre correspondants sont reliés par une instance de la classe InstructionsDict, qui est une sorte de dictionnaire à double sens et contient les méthodes nécessaires à la création, modification et suppression d'associations nom-nombre et à la traduction d'une forme vers l'autre. (instruction n'est pas une classe!) Les fonctions associées aux instructions prennent en argument le CPU qui les exécute (et ont par-là accès au contenu de la mémoire grâce aux pointeurs vers la classe Univers de la classe CPU), et ne retournent rien, elles ne font qu'agir sur la mémoire et le CPU qui les lit, à travers le contenu de ses registres et pile et sa position dans la mémoire.

Nous avons créé 37 instructions, en conservant la fonction et l'appellation de celles de Tierra. Les spécificités de notre système nous ont toutefois invité à en modifier quelques-unes, notamment par

rapport à la gestion des adresses relatives et des templates, l'organisation de la mémoire n'étant plus la même. De plus, nous avons introduit les deux instructions cruciales 'new' et 'HFC', qui permettent respectivement de créer un nouveau CPU vierge ou de tuer le CPU lisant l'instruction.

Nous avons, comme Thomas Ray, écrit un programme 'ancêtre' autoréplicateur d'une cinquantaine d'instruction que nous allons placer à l'origine de notre évolution virtuelle. La fonction main crée une classe Univers avec une mémoire sur laquelle se trouve uniquement l'ancêtre, un CPU vierge situé au début du code de l'ancêtre, puis exécute les CPUs qui sont apparus instruction par instruction, pendant un nombre préprogrammé d'instructions, conservant en mémoire les différentes étapes de l'évolution. Régulièrement, une 'photographie' de la mémoire et de l'état des CPUs sera enregistrée, mais le reste du temps, seules les modifications liées à chaque exécution de CPU sont enregistrées, de manière à ce qu'on puisse avoir accès à n'importe quelle étape de l'évolution en réappliquant les modifications. La fréquence des 'photographies' sera déterminée de manière à trouver un compromis entre espace mémoire et rapidité d'accès à une étape précise de l'évolution. Ces fonctionnalités d'archivage sont gérées par la classe Enregistrement.

# 4. Analyse des données

En se gardant de prédéfinir la notion d'individu, nous avons ouvert éventuellement la voie à l'émergence d'organismes présentant divers niveaux d'organisation et utilisant de nombreux CPUs différenciés, synchronisant leur action, leur prolifération et leur élimination, mais nous avons aussi rendu l'analyse des résultats de notre système évolutif bien plus complexe. En effet, pour définir les différentes espèces peuplant son monde, Thomas Ray pouvait se contenter de comparer les suites d'instructions présentes dans les différentes cases mémoire, de les classer, puis de compter les éléments de chaque espèce pour étudier ensuite le fonctionnement des espèces prédominantes. Dans Turing-Tierra, les organismes ne sont pas sagement rangés dans des cases mémoires prédéfinies et ne doivent pas faire appel à la machine virtuelle (comme pour un recensement !) à chaque fois qu'ils ont besoin d'espace mémoire ou créent un CPU : notre mode voit cohabiter de multiples modules et sous modules qui s'imbriquent, se

chevauchent, et dont les frontières ne sont pas visibles a priori. De plus, diverses séquences d'instructions utilisées par un même organisme peuvent se situer en des endroits divers de la mémoire, et non pas seulement dans des zones contigües. Un individu est donc susceptible d'être constitué par un ensemble quelconque d'instructions plus ou moins éparses dans la mémoire.

Pour définir des 'entités' vivantes que nous pourrons étudier (nous nous garderons d'utiliser le terme d' 'individu', dans la mesure où nous avons voulu remettre en cause cette notion), nous nous baserons sur le critère de la périodicité et de la reproduction à l'identique. En effet, nous pouvons analyser un système évolutionnaire selon trois échelles temporelles : l'échelle de temps 'microscopique' ou phénoménologique, où nous n'observons que des phénomènes physiques a priori non vivants, ce qui correspond dans notre modèle à l'exécution d'un CPU ; l'échelle 'mésoscopique' ou reproductive, qui est l'échelle de reproduction d'une cellule ou d'un organisme multicellulaire selon le niveau d'étude ; et enfin l'échelle 'macroscopique' ou évolutionnaire, où les espèces ne sont pas stables mais évoluent sans cesse. La seconde échelle de temps nous intéresse ici afin d'identifier quelles sont les « entités » qui se reproduisent. De plus, pour ne pas sentir les effets de l'évolution, nous étudierons les espèces présentes à un instant donné de l'évolution virtuelle en relançant à partir de cette instant initial une simulation sans mutation sur un nombre prédéterminé d'étapes. En effet, de même que l'étude de tissus morts n'est que d'une aide limitée pour comprendre les processus vitaux, nous ne pourrons étudier les formes de vie peuplant notre système qu'en l'étudiant dans sa dynamique.

Toutefois, l'idée de suivre l'évolution d'un CPU unique serait problématique : cela reviendrait à prédéterminer de nouveau l'échelle organisationnelle des organismes : un organisme serait ainsi une entité qui utiliserait un unique CPU pour se reproduire. Toutefois, si une forme d'organisme 'pluri-CPU' vient au monde, le CPU que nous suivons ne pourrait être qu'un CPU transitoire dans l'organisme, voué à disparaître sans descendant : on peut penser à l'apoptose ou 'mort cellulaire' en biologie, qui élimine des cellules qui n'ont plus de fonction dans l'organisme une fois leur tâche accomplie, comme à la fin d'une réponse immunitaire, ou lors de la formation des doigts (élimination des palmes primitives entre les doigts). Dans ce cas, on ne pourrait penser que ce CPU appartient à un organisme non viable qui a été éliminé par la sélection naturelle, ce qui n'est pas le cas.

Nous devons donc suivre non pas le comportement des CPUs uniques, mais de tous les CPUS qui descendent d'un même CPU, dit 'ancêtre', à l'instant initial. En effet, les divisions successives de CPUs

forment à partir de chaque ancêtre un arbre binaire de CPUs, dont certaines branches peuvent être ‘mortes’ (un CPU a disparu avant de se diviser). Entre chaque nœud, un CPU, entre sa dernière division et sa division suivante et sa mort, va lire une certaine suite d’instructions. Nous stockons cette suite d’instructions, et nous pouvons ainsi créer, par dualité, à partir de la première division de l’ancêtre, un nouvel arbre binaire de suites d’instructions. On considère les instructions elles-mêmes et non leurs emplacements dans la mémoire car cela permet directement de reconnaître la parenté entre une séquence originelle et une séquence recopiée. De plus, on ne considère pas non plus le contenu des registres et de la pile des CPUs car ceux-ci peuvent contenir des adresses relatives qui peuvent varier entre un organisme et son descendant si son espèce met en place une stratégie élaborée d’organisation dans l’espace mémoire.

A partir d’un arbre binaire de suites d’instructions, on établit un dictionnaire de synonymes entre ces suites, en s’autorisant éventuellement une certaine distance d’édition de tolérance. Une unité serait une classe de suites, et on pourrait créer un arbre binaire d'entités. La recherche d’une périodicité dans cet arbre permettra de définir un organisme se reproduisant. Une fois cette analyse effectuée à partir d’un ancêtre, la comparaison des différends organismes identifiés (sous la forme de portions d’arbre binaire) permettra d’identifier des espèces distinctes, que nous analyserons pour en comprendre le fonctionnement.

Sans même avoir à étudier les instructions précises constituant les branches de l’arbres, l’étude de la structure de l’arbre de symboles nous permettra de comprendre en partie l’organisation de telle espèce vivante à partir de la gestion de ses CPUs. Ainsi, si nous étudions un organisme pluricellulaire ou une espèce d’insectes sociaux comme les fourmis, nous trouverons un graphe caractéristique. Dans le cas d’un organisme pluricellulaire, en plaçant sur les branches de l’arbre les cellules le composant et en suivant leur division, on remarque que les cellules, d’abord généralistes (cellules souches) se reproduisent en se spécialisant. Parmi ces spécialisations, on distingue les cellules terminales, appelées à mourir avec l’organisme, et les cellules germinales, qui vont être à l’origine d’un nouvel individu (voir schéma, noter qu’on a ici une reproduction non sexuée). Si nous étudions une fourmilière en plaçant les fourmis sur les branches des arbres, on remarque qu’une même branche, la reine, est à l’origine de toute les autres, les ouvrières, qui sont ‘terminales’, meurent sans se diviser, jusqu’à ce que la reine engendre une princesse, qui deviendra reine d’une nouvelle fourmilière (voir schéma, la reproduction est encore non sexuée). Ainsi, dans ces deux cas, l’unité réplicative n’est pas la cellule ou encore la fourmi : une cellule germinale

ou une ouvrière ne se reproduit pas, et une cellule terminale ou une princesse engendre des unités différentes. L'unité qui se reproduit se situe à l'échelle de l'organisme ou de la fourmilière. Nous espérons que notre analyse permettra d'obtenir de tels arbres, dont l'étude pourrait permettre de dégager la richesse des modèles complexes pouvant émerger d'une évolution en boucle ouverte.

Cependant, des statistiques sur l'activité évolutive ou du moins reproductive de notre système peuvent être obtenues même sans avoir identifié des individus. Nous étudierons entre autres, l'évolution au cours du temps du nombre de CPUS et de l'occupation de l'espace-mémoire, le taux de création de CPUs, le nombre moyen d'instructions exécutées par un CPU (au cours de sa vie ou entre deux divisions) et le nombre moyen d'exécutions par des CPUs d'une instruction donnée, la densité spatiale de CPUs, leur activité en écriture… Ces statistiques pourront constituer une première vérification de la capacité de notre système à permettre la prolifération d'une vie artificielle.

# Organisation

## 1. Organisation générale du travail

Nous sommes dans ce PSC un groupe de 5. Nous sommes assez variés en termes de disciplines choisies. En l'occurrence, Ariane est la chef d'équipe, qui par exemple synthétise des diverses conceptions en une idée commune de groupe et organise les réunions régulières avec le tuteur. Pierre, qui a suivi des cours de biologie peut munir ce projet a priori informatique d'un point de vue biologique intéressant. Arthur, Lucas et Yueh qui ont plus de connaissances en informatique, notamment en programmation, peuvent donner des conseils et des aides techniques dont nous avons besoin pour l'implémentation de notre programme.

Pour ce qui était de l'analyse de la bibliographie, nous avons chacun effectué des lectures concernant des domaines précis de notre sujet de recherche, par exemple chacun s'est intéressé à quelques modèles de vie artificielle précis, et nous avons échangé ensemble sur nos lectures pour que tout le monde soit à jour.

Dans la partie conception du projet, nous avons travaillé en groupe de cinq pour bien réfléchir tous ensemble aux orientations de notre projet, à la manière dont nous imaginons tous notre machine virtuelle. Nous avons donc débattu des différentes options possibles afin de savoir lesquelles pourraient nous permettre de nous rapprocher d'une évolution en boucle ouverte. Notre tuteur nous a été d'une grande aide et nous a aidé à obtenir un regard de chercheur sur notre travail, et à réfléchir à la manière dont nous voulions inscrire notre travail dans la lignée des recherches sur la vie artificielle et l'open-ended evolution. Même si nous avons aussi réfléchi séparément d'une séance à l'autre, le travail était surtout collectif. La différenciation des connaissances selon les membres du groupe a été utile : les connaissances de biologie de Ariane et Pierre ont pu servir d'inspirations pour notre modèle, tandis que

les connaissances de programmation de Yueh, Lucas et Arthur ont permis de réfléchir à la faisabilité du modèle, notamment concernant les questions de complexité, de stockage des états successifs et de gestion du parallélisme.

Pour la partie implémentation, nous nous réunissons chaque semaine pour définir les tâches à effectuer, et chacun code sa partie séparément. Un GitHub nous permet de nous coordonner, pour travailler sur le même code et gérer les versions.

Lucas a codé la machine virtuelle, avec l'aide d'Arthur, Pierre et Yueh pour la création de toutes les instructions. Arthur a développé les fonctions permettant l'enregistrement des données au cours de l'exécution de la simulation et leur réutilisation. Ariane a créé l'environnement pour l'expérimentation et les fonctions de statistiques sur les résultats. De plus, pendant deux semaines, Yueh, Ariane et Pierre ont formé un sous-groupe pour réfléchir au problème de l'identification des individus et ont abouti à la conception des arbres binaires de suites d'instructions. À ce jour, Pierre et Lucas parachèvent et déboguent la machine virtuelle, Ariane aide au débogage général en créant des tests unitaires, Arthur termine l'enregistrement des données en coordination avec Yueh qui adapte un logiciel de clustering à nos besoins pour l'identifications des individus.

La machine virtuelle est globalement terminée, même si nous y effectuerons peut-être des petites modifications après les résultats des premières expériences. Notre Turing-Tierra fonctionne, mais il nous faut encore coder la partie analyse des données, qui est plus complexe mais non moins nécessaire.

## 2. Diagramme de Gant

**Bibliographie**

[1] Thomas Ray. 1992. Evolution, ecology and optimization of digital organisms. Santa Fe Institute working paper 92-08-042.
[2] Thomas Ray. 1994. Evolution, complexity, entropy and artificial reality. Physica D 75: 239-263
[3] Kurt Thearling, Thomas Ray. 1994. Evolving Multi-cellular Artificial Life. Proceedings of Artificial Life 4, Brooks and Maes. MIT Press.
[4] Christoph Adami. 1998. Introduction to Artificial Life. Springer, New York.
[5] Richard Lenski, Charles Ofria, Robert Pennock, Christpoph Adami. 2003. The evolutionary origin of complex features. Nature 423.
[6] A. Pargellis. 2001. Digital Life behavior in the Amoeba world. Artificial Life 7.
[7] A. Pargellis. 1996. The spontaneous generation of digital life. Physica D 91.
[8] Joshua Epstein, Robert Axtell. 1996. Growing artificial societies: social science from the bottom up. Brookings Institution Press.
[9] Larry Yaeger. Computational genetics, physiology, metabolism, neural systems, learning, vision, and behavior or PolyWorld: Life in a new context. Artificial Life III, Vol. XVII of SFI Studies in the Sciences of Complexity, Santa Fe Institute, 263-298
[10] Dave Cliff, Stephen Grand. 1999. The Creatures Global Digital Ecosystem. Artificial Life 5: 77-94
[11] Alastair Channon and Robert Damper. 1998. Perpetuating evolutionary emergence. From Animals to Animats 5. MIT Press.
[12] A. Channon. 2001. Passing the ALife test : Activity statisitics classify evolution in Geb as unbounded. Artificial Life: Proceedings of the Sixth European Conference on Artificial Life, Prague, p417-426, Springer Verlag.
[13] Lisa Soros, Kenneth Stanley. Identifying Necessary Conditions for Open-Ended Evolution through the Artificial Life World of Chromaria. Proc. of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14)

**Références**

Christopher Langton. 1997. Artificial Life, an Overview. MIT Press.
Andrew Adamatzky. 2005. Artificial Life Models in Software. Springer
Federico Moran, Alvaro Moreno, Juan Merelo, Pablo Chacon. 1995. Advances in Artificial Life : Third European Conference on Artificial Life, Granada, Spain, June 4 - 6, 1995 Proceedings. Springer
Jean Génermon. 1979. Les mécanismes de l'évolution. Dunod Université.
Karl Sims. 1994. Evolving Virtual Creatures. SIGGRAPH '94 Proceedings: 15–22