よりシンプルな
バイナリ生物学の本

— よりシンプルなバイナリ生物学の設計と
PoC 実装によるデイジーワールド実験を紹介！ —

［著］大神祐真

技術書典 15 新刊
2023 年 11 月 11 日　 ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

はじめに

本書を手に取っていただき、ありがとうございます！ 本書は「バイナリ生物学」の理論を 1960 年代のコンピュータである PDP-7 でも動くくらいシンプルにした設計を解説し、その PoC 実装で行った実験とその結果を紹介する本です。

「バイナリ生物学」は「実行プログラム等のバイナリが生き物のように振る舞えるシステムを考え、実装する」という独自の取り組みで、これを OS のレイヤーで実装したものを「DaisyOS」と呼んでいます。本書は DaisyOS の PDP-7 版の設計と PoC 実装を紹介する本となります。と言っても設計として PDP-7 に依存する部分はごく一部で、そのような部分については都度解説しますので、特に PDP-7 に精通していなくても大丈夫です。

なお、PoC は、「SimH」と呼ばれる、PDP-7 を含む古いコンピュータのシミュレータ上で動作するように実装しました。[*1]そして PoC 実装を用いた実験では、「生物は地球と相互に関係し合って自己制御システムを作り上げているのではないか」という「ガイア理論」という理論の説明に用いられる「デイジーワールド」と呼ばれるモデル環境を再現する実験を行いました。「バイナリ生物学の実装の上で生物による自己制御のようなものが働くのかどうか」といった所を実験を用いて紹介します。

「バイナリ生物学」や「DaisyOS」といった取り組み自体、まだまだ発展途上なものではありますが、「こんなものが OS の一種としてあっても面白いかも」と思って楽しんでいただければ幸いです。

本書の更新情報等について

本書を含め、当サークルの同人誌・同人作品の情報は下記の筆者ウェブページにまとめています。

- `http://yuma.ohgami.jp/`

本書の内容について訂正や更新があった場合もこちらのページに記載します。何かおかしな点があった場合等は、まずこちらのページをご覧ください。

[*1] 基本的に設計をそのまま実装しているだけなので、本書では実装については特に説明しておりません。SimH というシミュレータ上での PDP-7 のプログラムの作り方については既刊「SimH で PDP-7 ベアメタルプログラミング」をご覧ください。

目次

第 1 章

PDP-7 とバイナリ生物学、DaisyOS について

この章では、本書の重要なキーワードである「PDP-7」と「バイナリ生物学」、「DaisyOS」の 3 つについて、本書でそれを扱う動機も含めて説明します。

1.1 PDP-7 とは

「PDP-7」は 1960 年代にデジタル・イクイップメント・コーポレーション (略称「DEC」) からリリースされたコンピュータです。PDP は「Programmed Data Processor」の略で、その名の通りプロセッサ部分を指すのですが、それが当時はミニコンピュータと呼ばれていた PDP-7 でさえ大型冷蔵庫を横に 3 台並べたくらいの大きさがありました。今はオープンソースのシミュレータ (SimH[*1]) が有志で作られており、これを用いると現代の PC 上で PDP-7 に限らず歴史的なコンピュータを試してみることができます。

PDP-7 が特に歴史的に重要なのが「UNIX が生まれたコンピュータである」という点です。1969 年にケン・トンプソンを中心に作られた UNIX は PDP-7 を対象に作られました。UNIX が現代の OS に与えている影響は大きく、「ファイル」や「プロセス」という概念[*2]は OS の基本的な概念として根付いています。そしてその様な

[*1] http://simh.trailing-edge.com/

[*2] なお、「ファイル」はケン・トンプソンが考案したものだそうですが、「プロセス」は UNIX 以前からあったそうです。ただそれが今知られているものと同じ様なものであったかどうかは不明で、今知られている様な「プロセス」は UNIX に由来するのかなと思います。ref.http://www.bell-l

背景を持つコンピュータで「ファイル」でも「プロセス」でもない独自理論に基づく OS を実現してみたいというのが本書で PDP-7 を扱う動機です。

PDP-7 については以下の既刊があります。「SimH」の使い方等について詳しくはこちらをご覧ください。

SimH で PDP-7 ベアメタルプログラミング

「SimH」というシミュレータ上でアセンブリ言語 (一部機械語) によるベアメタルプログラミングを行う本。

1.2　バイナリ生物学とは

「バイナリ生物学」は筆者の造語で、実行バイナリ等のバイナリデータが生き物のように振る舞える仕組みを考え実装する取り組みです。バイナリが生き物のように振る舞えることで、例えば実行バイナリの生成について「ソースコードをコンパイルして生成する」のではなく「生物の進化の仕組みで生み出す」といった事ができるようになります。

バイナリを生き物のように振る舞わせる考え方としては、「生き物の構成要素をバイナリの構成要素と対応付ける事で、生き物の振る舞いをバイナリが行えるようにする」というものです。想定する生き物は、おそらく生き物の中で最も簡単な構成である「単細胞生物[*3]」です。単細胞生物が行う「代謝」・「運動」・「成長」・「増殖」・「死」という 5 つの振る舞いを独自に簡単な形でモデル化し、それらをバイナリが行えるようなシステムを設計・実装します。

バイナリ生物学について詳しくは下記の同人誌をご覧ください。本書の内容は以下の 2 冊で紹介した設計を PDP-7 向けにシンプルにしたものです。

バイナリ生物学入門

実行バイナリを生物のように扱う処理系の構造と振る舞いのモデルと、実際に単細胞生物として扱う PoC の実験結果を紹介する本。

バイナリ生成環境「daisy-tools」実験報告

前著に突然変異による進化の仕組みを追加した設計と、それに基づいて作成した ELF バイナリ生成システム「daisy-tools」とその実験結果を紹介する本。

---

abs.com/usr/dmr/www/hist.html

[*3] 中でも特に「原核生物」を参考にしています。

1.3 DaisyOS とは

筆者は個人サークル「へにゃぺんて」で「自作 OS」を軸に活動をしているのですが、せっかくの趣味の自作 OS なら独自な OS として「ファイルやプロセスといった概念ではない OS」を作ってみたいと考えています。そこで、「バイナリが生き物として振る舞える環境を OS として提供する」という「バイナリ生物学に基づく OS」に取り組んでおり、その考えに基づいて設計・実装している OS を「DaisyOS」と呼んでいます。

DaisyOS としては、バイナリが生き物のように振る舞える環境を OS というレイヤーで提供できることで何か新たな価値が生まれるのではないかと想定しています。ただ、前節で紹介したバイナリ生物学の理論に基づいた実装は何度か行なっているのですが、それによってどんな事ができるかというアイディアがまだあまり無く、本書で解説する PDP-7 向けや過去に行なったゲームボーイ向けの実装でも、現状でできる事は「設定された評価関数を満たす小さな実行プログラム (機械語列) を生成する」程度のものです。ただそれは近年の筆者の興味が「バイナリを生き物の進化の仕組みで生み出すという事をどれだけシンプルな仕組みで実現できるか」という所にあったためで、古いコンピュータを対象にしているのも半ば強制的に設計をシンプルにしていくためであったりもします。ちなみに、ゲームボーイ向けの実装についてはもし興味があれば以下をご覧ください。

DaisyOS(仮) v0.1.0

ゲームボーイ上で動作するバイナリ生物学のシステム。ROM ファイルを筆者ウェブページからリンクしている BOOTH のページで無料公開中。

第 2 章
PDP-7 向け設計の解説

この章では、PDP-7 でも動くくらいにシンプルに行った DaisyOS の設計を解説します。

2.1 設計の概要

実行バイナリを生き物として振る舞わせるために、まず「バイナリ生物」というデータ構造で実行バイナリを定義します。そしてこの「バイナリ生物」を生き物のように振る舞わせるために、「代謝/運動」(セットで 1 つの振る舞い)・「成長」・「増殖」・「定常的なエネルギー消費」・「死」という 5 つの振る舞いを各バイナリ生物へ周期的に実行します。これにより、評価関数を満たすように実行バイナリを進化させていく事ができます。

2.2 バイナリ生物のデータ構造

まず、バイナリ生物のデータ構造は表 2.1 の通りです。なお、オフセットやサイズの単位の「ワード」は PDP-7 の場合「18 ビット」で、これは PDP-7 が「18 ビット」のプロセッサであるためです。

バイナリ生物のデータ構造の各フィールドについて、「開始シグネチャ」と「終了シグネチャ」は全バイナリ生物で共通の固定値です。後述しますが、これは「増殖」の際に新たに生まれるバイナリ生物を配置する領域を探す際に用いられます。開始シグネチャから終了シグネチャまでを「既にバイナリ生物が存在している領域」という

▼表 2.1: バイナリ生物のデータ構造

オフセット [ワード]	サイズ [ワード]	フィールド名	説明
0	1	開始シグネチャ	バイナリ生物の開始を示すシグネチャ
1	1	エネルギー	エネルギー残量。毎周期で一定量減り 0 になると死。「成長」で獲得した量が加算される。「増殖」のコストもここから支払う
2	1	適応度	「代謝/運動」での評価結果の点数 (最大 100) 「成長」で獲得できるエネルギー量に関わる
3	1	命令数 (N)	「命令列」フィールドの命令の数
4	1	戻り先アドレス	「命令列」フィールドを関数として実行する際の戻り先アドレスを設定する場所
5	1...N	命令列	PDP-7 の機械語命令の列。最後は関数からリターンする命令であること
5 + N	1	終了シグネチャ	バイナリ生物の終了を示すシグネチャ

扱いにしているため、これらのシグネチャは PDP-7 の機械語命令やバイナリ生物のその他のフィールドと被らない様な値である必要があります。その様な値であればどの様な値でも構いませんが、次章で紹介する実験に用いた PoC 実装では開始シグネチャは 0o713130[*1][*2]に、終了シグネチャは 0o713137[*3]という値にしています。

「戻り先アドレス」というフィールドがあるのは、PDP-7 の関数呼び出し命令 (`JMS` 命令) の仕様によるものです。バイナリ生物のデータ構造では、自身の命令列を関数呼び出しで実行できるようにしているのですが、 `JMS` 命令は「指定されたアドレスに戻り先アドレス[*4]を設定し、その次のアドレスから実行する」という挙動であるため、「命令列」フィールドの直前に「戻り先アドレス」のフィールドを用意しています。

ちなみに関数からのリターンで使用する命令は `JMP I` 命令です。これは「指定されたアドレスが指す先のアドレスへジャンプする」という挙動で、「戻り先アドレス」フィールドのアドレスを指定することで関数からのリターンが行えます。[*5]

[*1] 「0o」は 8 進数表記を意味する接頭辞です。

[*2] 下位 5 桁の「13130」は、「13」を「B」、「0」を「O(オー)」と見立てて、「BBO(Binary Biological Object、バイナリ生物オブジェクト)」を示していたりします。最上位の桁の値に意味はなく、いかなる機械語命令にも該当しないように「7」にしてみました。

[*3] こちらは開始シグネチャの方をベースにして、対応する機械語が無い値を探した結果こうなっただけで特に意味はありません・・。

[*4] 関数からのリターンで戻ってくる場所のアドレスです。関数呼び出しを行なっている `JMS` 命令の次のアドレスになります。

[*5] これらの PDP-7 の命令についてより詳しくは前章で紹介した既刊「SimH で PDP-7 ベアメタルプログラミング」をご覧ください。

「命令数」や「命令列のワード数」の「N」について

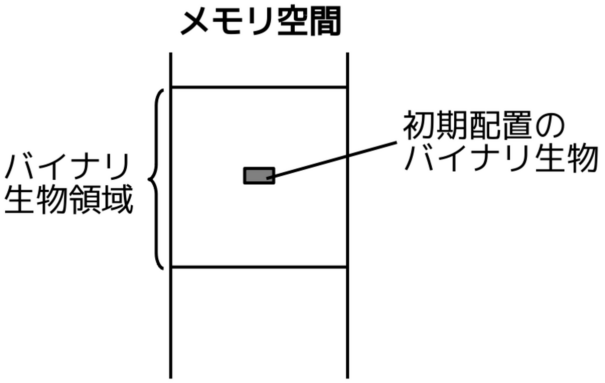
表 2.1 では、「命令数」も「命令列のワード数」も共に「N」としている通り、現状ではバイナリ生物が持つ各命令のサイズは 1 ワードであることを想定しています。PDP-7 の命令はほとんどが 1 ワードなのですが、一部、複数ワードで 1 つの命令となるものもあります。ですが、現状ではそのような命令はバイナリ生物の命令列として扱わないこととします。

「エネルギー」と「適応度」のフィールドをどのように使用するかについては、後ほどバイナリ生物に対するそれぞれの振る舞いを説明する際に紹介します。

2.3　初期状態

バイナリ生物は専用の領域をメモリ上に確保しそこへ配置します。以降、この領域を「バイナリ生物領域」を呼びます。

初期状態としては、バイナリ生物領域に初期配置のバイナリ生物のみが配置された状態です (図 2.1)。図 2.1 では例としてバイナリ生物が 1 つだけの状態を示していますが、初期状態で複数存在していても特に問題はありません。



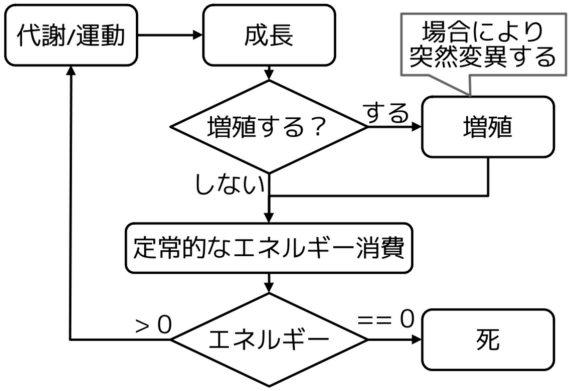
▲図 2.1: バイナリ生物領域の初期状態

その他に「評価関数」や「命令候補テーブル」といった要素も予め用意しておく

必要がありますが、これらについては後ほど個別の振る舞いを説明する際に紹介します。

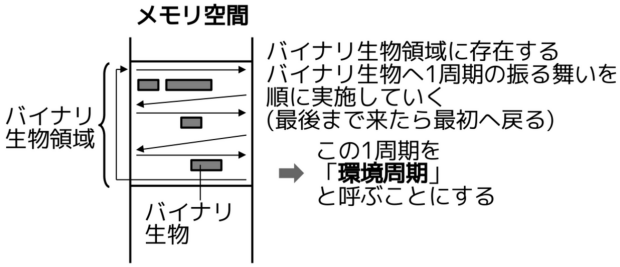
2.4　動作の流れ

バイナリを生き物として振る舞わせるために、「代謝/運動」(セットで 1 つの振る舞い)・「成長」・「増殖」・「定常的なエネルギー消費」・「死」という 5 つの振る舞いを実行します。図示すると図 2.2 の通りで、「代謝/運動」から再び「代謝/運動」に戻ってくる (あるいは「死」) までがバイナリ生物の 1 周期です。



▲図 2.2: バイナリ生物の周期動作

基本的な動作としては、この 1 周期をバイナリ生物領域に存在するバイナリ生物へ順に実施していくだけです (図 2.3)。なお、バイナリ生物領域内の全てのバイナリ生物で周期動作を終え、バイナリ生物領域の先頭に戻ってくるまでの周期を「環境周期」と呼ぶことにします。

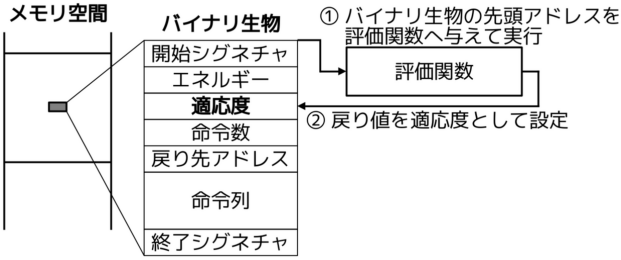


▲図 2.3: 基本動作

以降では、これら 5 つの振る舞いについて説明します。

♣ 代謝/運動

「代謝/運動」では、まずバイナリ生物の評価を行います。評価は予め用意した「評価関数」で行います。評価関数は、今対象にしているバイナリ生物を 100 点満点で評価し、点数を関数の戻り値として返すものです。そして、評価関数の戻り値で得られた点数を、評価したバイナリ生物の「適応度」へ設定します。代謝/運動の振る舞いを図示すると図 2.4 の通りです。



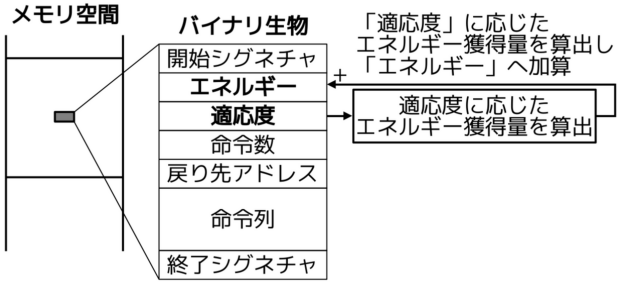
▲図 2.4: 代謝/運動

評価関数でどの様な評価を行うかによって、どの様なバイナリが生成されるかが決まります。評価の仕方は何でも構いません。例えば「試しに実行してみてその結果が期待通りか否かで評価する」というのがよく行う評価方法です。

♣ 成長

「成長」では、バイナリ生物の「適応度」に応じたエネルギー獲得量を算出し「エ

ネルギー」へ加算します (図 2.5)。エネルギー獲得量を算出する式もチューニング箇所の一つです。大前提として適応度が高くなるに連れてエネルギー獲得量も大きくなるように式を立てると良いです。本書で実験に用いた PoC 実装では 0 から 100 の適応度をそのままエネルギー獲得量としています。



▲図 2.5: 成長

♣ 増殖

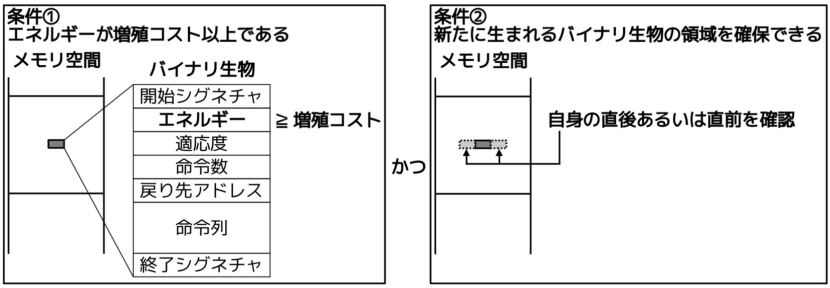
基本的な増殖の流れ

「増殖」では、増殖条件が成立している場合、自身のコピーをバイナリ生物領域に生成します (体細胞分裂)。増殖条件は、実装次第ではありますが、基本的に「エネルギーが増殖コスト以上である」かつ「新たに生まれるバイナリ生物の領域を確保できる」場合に増殖可能となります。

新たに生まれるバイナリ生物の領域をどう確保するかも実装次第ではありますが、本書で実験に用いた PoC 実装では分裂元のバイナリ生物の直後あるいは直前を確認します。このようにしている理由は、メモリ空間内に実際に生きている単細胞生物っぽくしたかった事[*6]と実装が容易であるためです。

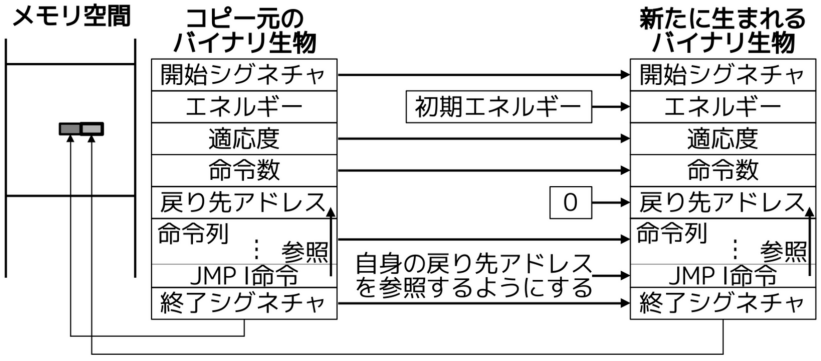
増殖条件の確認について図示すると図 2.6 の通りです。

[*6] 自身と離れた所に自身のコピーが生まれるというのは、体細胞分裂っぽくないなと思いまして・・。



▲図 2.6: 1. 増殖条件が成立しているかを確認

なお、体細胞分裂の際、「エネルギー」と「命令列内の `JMP I` 命令 (リターン命令)」は自身のコピーではなく新たに値を生成します。「エネルギー」には予め定義された「初期エネルギー」を設定します。「命令列内の `JMP I` 命令」は新たに生まれるバイナリ生物では「戻り先アドレス」のアドレスが変わっているので、新しい「戻り先アドレス」を指すように命令を作り直します。ちなみに「戻り先アドレス」は関数呼び出しの命令 (`JMS` 命令) の際に自動的に戻り先アドレスが書き込まれる領域なので初期値としては何でも良いです。自身の値をコピーしても害はないのですが、本書で実験に用いた PoC 実装では 0 を設定しています。これらを踏まえて体細胞分裂について図示すると図 2.7 の通りです。

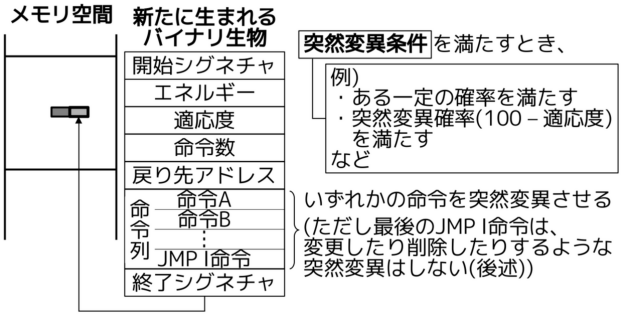


▲図 2.7: 2. 自身のコピーを作成 (体細胞分裂)

突然変異

体細胞分裂の際、突然変異条件が成立すると、新たに生まれるバイナリ生物の命

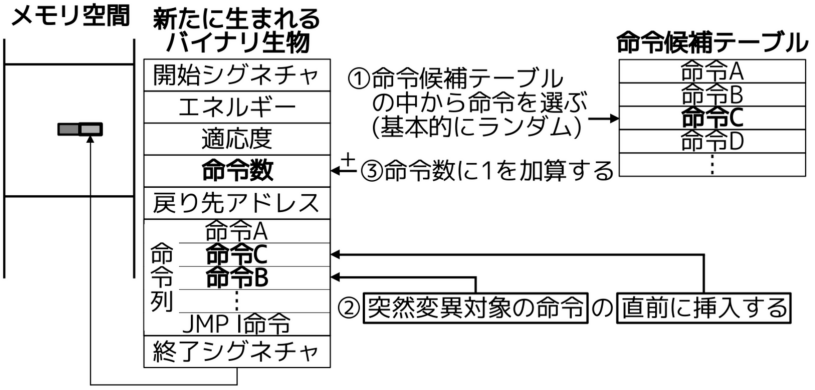
令列のいずれかの命令が突然変異します (図 2.8)。突然変異条件も実装次第/チューニング次第ですが、やり方としては例えば「ある一定の確率で突然変異する」とか『「100 - 適応度」を突然変異確率とする』等です。前者の場合、一度適応度 100 のものが生まれても、それが突然変異して逆に適応度が下がってしまう「退化」の可能性がありますが、後者の場合は適応度が高くなるに連れ突然変異しにくくなるので、その様な恐れはありません。ただ、後者の場合でも「適応度 90 辺りから格段に突然変異しにくくなるため、90 より高い適応度のものが生まれるのに時間がかかる」といった欠点があります。本書で実験に用いた PoC 実装では後者の『「100 - 適応度」を突然変異確率とする』としています。



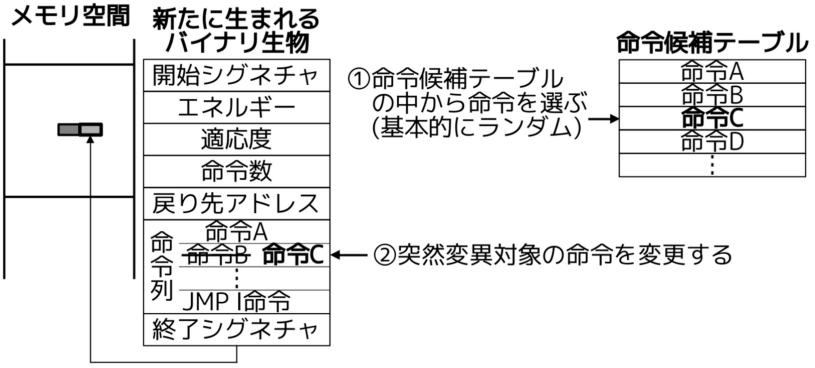
▲ 図 2.8: 突然変異

突然変異の際に「どの命令を選ぶか」も実装次第です。「ランダムに選ぶ」や「特定の場所の命令のみを対象とする」といったことが考えられます。

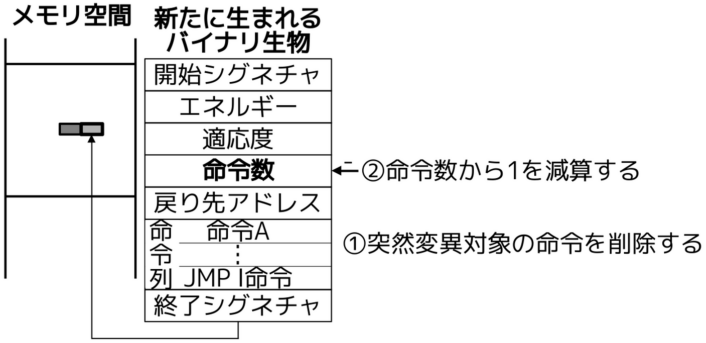
また、選ばれた命令の突然変異のさせ方としては「追加」・「変更」・「削除」のいずれかがあります。これらを「突然変異タイプ」と呼びます。突然変異タイプの「追加」は、選ばれた命令の直前に「何らかの命令」を追加するというものです (図 2.9)。次に「変更」は選ばれた命令を「何らかの命令」へ変更します (図 2.10)。最後に「削除」は選ばれた命令を削除します (図 2.11)。「追加」・「変更」の際の「何らかの命令」はメモリ空間上に予め用意した「命令候補テーブル」の中から選びます。「命令候補テーブル」は突然変異の際に使用して良い PDP-7 機械語命令を並べておくテーブルです。「突然変異タイプのどれを選ぶか」や「命令候補テーブルの中のどの命令を選ぶか」も今の所「ランダム」です。なお、突然変異する命令として命令列最後のリターン命令が選ばれた際は、突然変異タイプとして「変更」あるいは「削除」を選ばないようにする必要があります。



▲図 2.9: 突然変異タイプ (追加)



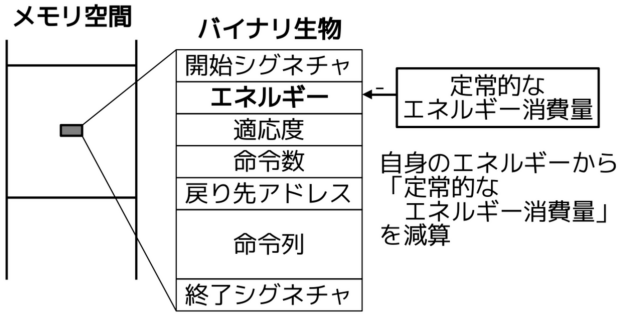
▲図 2.10: 突然変異タイプ (変更)



▲図 2.11: 突然変異タイプ (削除)

♣ 定常的なエネルギー消費

予め定義された「定常的なエネルギー消費量」を自身のエネルギーから減算します (図 2.12)。この消費量もチューニングパラメータの 1 つです。1 周期で獲得する最大エネルギー量より多くしておく事で、バイナリ生物が永久に生き続けることが無いようにできます。本書で実験に用いた PoC 実装では、「成長」でのエネルギー獲得量が適応度をそのまま使っているため最大 100 である事から、定常的なエネルギー消費量は 110 としてみています。[*7]

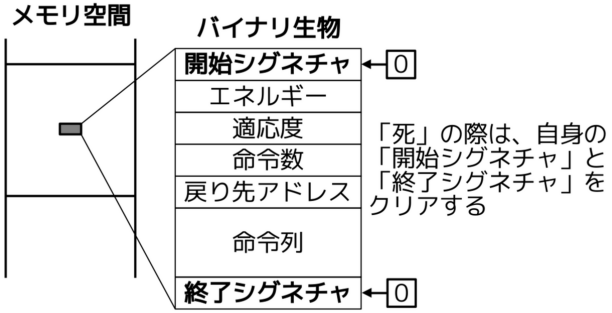


▲図 2.12: 定常的なエネルギー消費

[*7] これはあまり深く考えて決めた値では無いので、まだまだ調整の余地はあるかと思います。ただ、次章で紹介する実験結果の通り、この値のままでも一応ちゃんと動くようではあります。

♣ 死

エネルギーが 0 以下になると死にます。その際は自身の「開始シグネチャ」と「終了シグネチャ」をクリアします (図 2.13)。シグネチャをクリアすると領域が解放され、他のバイナリ生物の増殖時に使用できるようになります。



▲図 2.13: 死

第 3 章
PoC 実装を用いた実験の紹介

この章では、「ガイア理論」の「デイジーワールド」と呼ばれるモデル環境を、PoC 実装上で独自に再現した実験とその結果を紹介します。

3.1 PoC 実装について

前章までで紹介した内容を PoC として実装したものは以下の GitHub リポジトリで公開しています。使い方等はリポジトリ内の README をご覧ください。

- https://github.com/cupnes/daisy-os-pdp7

3.2 デイジーワールドについて

この章では、PoC 実装上に「デイジーワールド」と呼ばれる環境を設定し、実験を行った結果を紹介します。

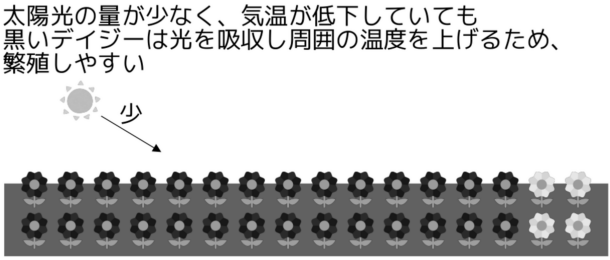
「デイジーワールド」とは、「ガイア理論」と呼ばれる理論の妥当性を示すために考えられたモデル環境です。[*1]「ガイア理論」は「ガイア仮説」とも呼ばれるもので「生物は地球と相互に関係し合い、自身の生存に適した環境を維持するための自己制御システムを作り上げている」とする仮説です。[*2]

デイジーワールドでは、この仮説を考えるために「白と黒の 2 種類のデイジーしか

[*1] ref.https://ja.wikipedia.org/wiki/デイジーワールド
[*2] ref.https://ja.wikipedia.org/wiki/ガイア理論

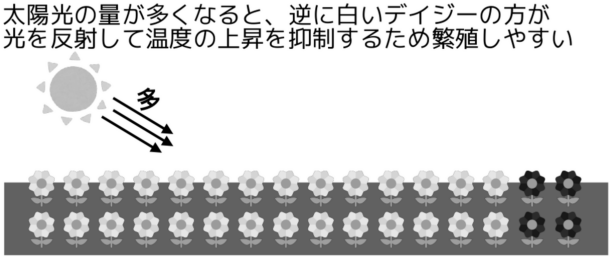
存在しない世界」を仮定し、その世界では「地表に降り注ぐ太陽光の量の変化に対し、地表付近の温度を一定に保とうとする恒常性が働く」ということを示します。

考え方をざっくりと説明すると、まず前提として、白と黒の 2 種類のデイジーは共に、生育できる適温は同じであるとします。その上で、例えば、太陽光の量が少なく地表付近の温度が低下している場合、黒いデイジーは光を吸収し周囲の温度を上げるため、光を反射し熱を放出してしまう白いデイジーに比べて繁殖しやすいと考えられます (図 3.1)。



▲図 3.1: 太陽光の量が少ない場合、黒いデイジーの方が繁殖しやすい

逆に、太陽光の量が多く地表付近の温度が上昇している場合、光を反射する白いデイジーの方が、黒いデイジーに比べて繁殖しやすいと考えられます (図 3.2)。



▲図 3.2: 太陽光の量が多い場合、白いデイジーの方が繁殖しやすい

この時、環境に存在する 2 種類のデイジーにより地表温度を生育適温に保とうとする恒常性が働いていると考えられます。ガイア理論では、このような仕組みが地球上にも存在するのではないかとしています。

この章では PoC 実装上でデイジーワールドを (少々独自に) 再現し、バイナリ生物

学の仕組みの上で自己制御のようなものが働くことができるかどうかを確認します。

3.3 実験設定

デイジーワールドを、前章までで紹介した設計 (そしてその PoC 実装) の上で再現するために行った実験設定を紹介します。[*3]

♣ 白と黒のデイジーのバイナリ生物表現

白と黒のデイジーをバイナリ生物としてどの様に表現したかを紹介します。白と黒のデイジーをバイナリ生物のデータ構造へ設定した結果は表 3.1 と表 3.2 の通りです。

▼表 3.1: 白のデイジーのバイナリ生物の初期値

フィールド名	初期値
開始シグネチャ	0o713130
エネルギー	1000
適応度	50
命令数	2
戻り先アドレス	0
命令列 [0]	TAD <定数-1 のアドレス>
命令列 [1]	JMP I <「戻り先アドレス」のアドレス>
終了シグネチャ	0o713137

表 3.1 と表 3.2 について、表のタイトルの通り、これは初期値であり「エネルギー」・「適応度」・「戻り先アドレス」は実行する中で変化していきます。

「命令列」について、表 3.1 と表 3.2 ではアセンブリ言語で記載していますが、実際には機械語で設定されます。`TAD` 命令は PDP-7 の命令の一つで「オペランドで指定されたアドレスの値を AC レジスタ[*4]へ加算する」という命令です。あらかじめ「-1」や「1」といった定数をメモリ上のどこかに置いておいて、そのアドレスを `TAD` 命令のオペランドに指定することで AC レジスタのデクリメントやインクリ

[*3] 厳密には「デイジーワールド」とは違う部分もあるかもしれません。「デイジーワールドっぽいもの」と見ていただければと思いますが、「生物が生きることで何らかの自己制御が働く」という事を確認する目的は変わりません。

[*4] 一般的に「アキュムレータ」と呼ばれるレジスタです。PDP-7 では演算を行う対象として基本的にこのレジスタを使用します。

▼表 3.2: 黒のデイジーのバイナリ生物の初期値

フィールド名	初期値
開始シグネチャ	0o713130
エネルギー	1000
適応度	50
命令数	2
戻り先アドレス	0
命令列 [0]	TAD <定数 1 のアドレス>
命令列 [1]	JMP I <「戻り先アドレス」のアドレス>
終了シグネチャ	0o713137

メントが行えます。PoC 実装では AC レジスタの値を「地表温度」とし[*5]、デクリメントやインクリメントを行うことで「白のデイジー」や「黒のデイジー」としています。白/黒のデイジー共に命令数は 2 で固定ですので、バイナリ生物のサイズとしてはいずれも 8 ワードとなります。なお、デイジー 1 つで 1 ℃の温度上昇/下降というのはデイジー 1 つの影響が大きい気もしますが、これに関しては後述する別のパラメータで対処しています。

そして、デイジーに関するその他のパラメータとして、生育適温は 20 ℃としました。[*6]

♣ バイナリ生物領域の初期状態

バイナリ生物領域のサイズは 4096 ワードとしました。前節で紹介した通り、白/黒共にデイジーのバイナリ生物としてのサイズは 8 ワードなので、配置可能な最大個体数は 512 個です。

そして、初期状態としては前半 2048 ワードの中に白いデイジーを 1 つ、後半 2048 ワードの中に黒いデイジーを 1 つ配置するようにしてみました。この初期状態のバイナリ生物領域の使用状況を色分けで図示すると図 3.3 の通りです。白色の領域が白いデイジーが存在する領域を、黒色の領域が黒いデイジーが存在する領域を示し、灰色の領域はいずれも存在しない領域を示します。

[*5] AC レジスタは汎用的に使用されるレジスタであるため、普段はメモリ上の専用の場所へ退避していて、命令列を実行する直前で AC レジスタへ復帰しています。

[*6] 実験前に軽く調べた際にデイジーの生育適温は 20 ℃辺りというのを見かけたのでここでは 20 ℃という設定で実験を行いました。ただ、実験後に改めて調べた所、10 ℃前後な様です・・。まあ、今回の実験のデイジーは暑さに強く寒がりということで・・。

▲図 3.3: バイナリ生物領域の初期状態

♣ 評価関数

シンプルな設計でデイジーワールドのような自己制御が働くかを確認する目的で、評価関数もなるべくシンプルにしました。実験で設定した評価関数を C 言語のような疑似コードで表すとリスト 3.1 の通りです。

▼リスト 3.1: 評価関数

```
#define 生育適温 20
int 地表温度;        /* 現在の地表温度を保持 */
int 評価関数(評価対象のバイナリ生物) {
        int 適応度;
        int 誤差;

        /* 評価対象のバイナリ生物の命令列を実行 */
        ACレジスタ = 地表温度;
        評価対象のバイナリ生物.命令列();       /* 関数呼び出しで実行 */
        地表温度 = ACレジスタ;

        /* 実行結果を評価 */
        誤差 = 地表温度 - 生育適温;
        if (誤差 >= 0) {
                /* 生育適温より高いため、白いデイジーへ高い適応度を設定 */

                if (誤差 > 50) {
                        誤差 = 50;
                }

                if (評価対象のバイナリ生物 == 白いデイジー) {
                        適応度 = 50 + 誤差;
                } else {
                        適応度 = 50 - 誤差;
                }
```

```
        } else {
                /* 生育適温より低いため、黒いデイジーへ高い適応度を設定 */

                if (誤差 < -50) {
                        誤差 = -50;
                }

                if (評価対象のバイナリ生物 == 白いデイジー) {
                        適応度 = 50 - 誤差の絶対値;
                } else {
                        適応度 = 50 + 誤差の絶対値;
                }
        }

        return 適応度;
}
```

リスト 3.1 について簡単に説明すると、生育適温に対して地表温度が高い場合は熱を放出する必要があるので白いデイジーへ高い適応度を設定し、逆に低い場合は熱を蓄える必要があるため黒いデイジーへ高い適応度を設定するようにしています。

♣ 増殖コスト

デイジー 1 つで 1 ℃も温度が上下するため、個体数を低く抑えるために、増殖コストは 900 という高い値を設定しました。[*7]

♣ 突然変異関連

突然変異関連でデイジーワールド固有の部分としては、「突然変異対象の命令が常に命令列の最初の命令である」事があります。「白/黒のどちらのデイジーであるか」は「命令列の最初の `TAD` 命令で AC レジスタをデクリメントするかインクリメントするか」です。加えて、白/黒のデイジー以外へは突然変異しない世界なので、突然変異の際に対象となる命令は常に最初の `TAD` 命令となります。

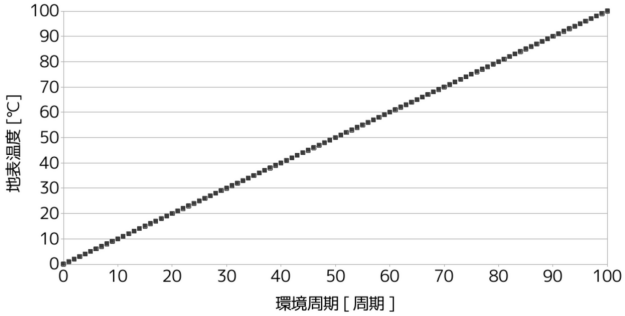
また、同様の理由で、突然変異タイプも「変更」のみです。そして、命令候補テーブルに置いておく命令も、インクリメントとデクリメントの `TAD` 命令のみとなります。

♣ 外乱

デイジーの存在に関わらず環境周期 1 周期ごとに地表温度を 1 ℃ずつ上昇するという外乱の下で実験を行いました (開始時は 0 ℃)。参考として、この外乱を設定した

[*7] ここで調整するよりも、デイジー 1 つあたりの地表温度への影響を低くした方が良かった気もしますが、その事に気づくのが遅すぎました・・。

上でバイナリ生物領域に何も存在しない状態で環境周期 100 周期分を実行した際の地表温度の推移としては図 3.4 の通りでした。



▲図 3.4: デイジーが存在しない場合の地表温度の推移

0 ℃から始まり、1 ℃ずつ温度が上昇していることが確認できます。このような外乱の下、デイジーの存在によって地表付近の温度がどうなるかを実験で確認します。

なお、実験パターンによってはこの他にも外乱を与える場合がありますが、それは個別に紹介します。

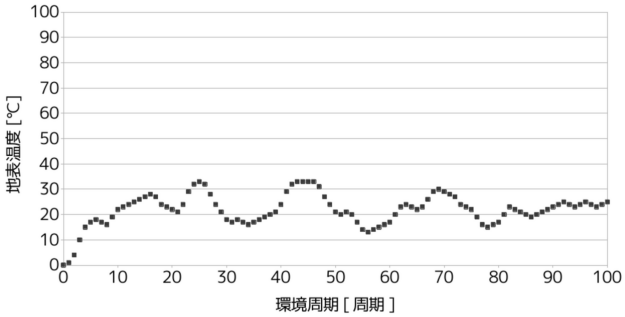
3.4　実験結果と考察

以降ではいくつかのパターンで実施した実験とその結果を紹介します。

なお、以降で紹介する実験はいずれも一回の実験の期間は環境周期 100 周期分です。

♣ 地表温度の推移

前節の実験設定で実験した結果、地表温度の推移として図 3.5 の結果が得られました。



▲図 3.5: デイジーが存在する場合の地表温度の推移

生育適温 (20 ℃) ピッタリという訳ではないですが、地表温度と生育適温の差が 0 になるように調節しているような働きを確認できます。以降では地表温度と生育適温の差のことを「誤差」と呼ぶことにします。

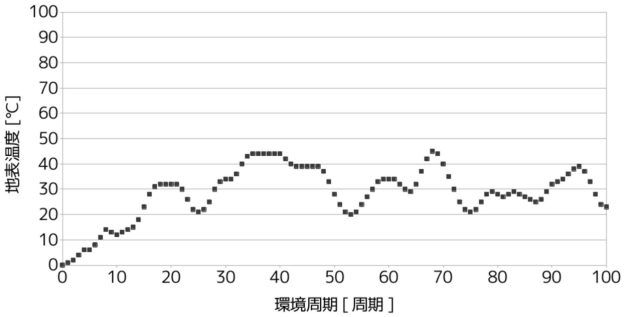
この実験での各環境周期における誤差について簡単な統計処理を行ってみた結果は表 3.3 の通りでした。

▼表 3.3: この実験での誤差の統計処理結果

方法	誤差 [℃]
平均	4.78
最大	19
最小	0
パーセンタイル	
50(中央値)	4
90	11

90 パーセンタイルの誤差が 11 ℃であることから、この実験での誤差は九割方で約 10 ℃以下であり、地表温度を概ね 10 ℃〜30 ℃辺りで調節できていると言えそうです。

なお、実はこの実験結果は、同条件で実験を 100 回行った中で誤差の 90 パーセンタイル値が最も低かった結果で、言わば「良くてこれくらい」という結果です。逆に「悪くてこれくらい」というか「大半がこれくらいの誤差に収まっている」ということを確認するために、100 回の実験それぞれの 90 パーセンタイル値を昇順に並べた際の第 90 位 (90 パーセンタイル) の結果を示すと図 3.6 と表 3.4 の通りでした。



▲図 3.6: 地表温度の推移 (100 回中の 90 位)

▼表 3.4: 90 位の統計処理結果

方法	誤差 [℃]
平均	11.68
最大	25
最小	0
パーセンタイル	
50(中央値)	12
90	22

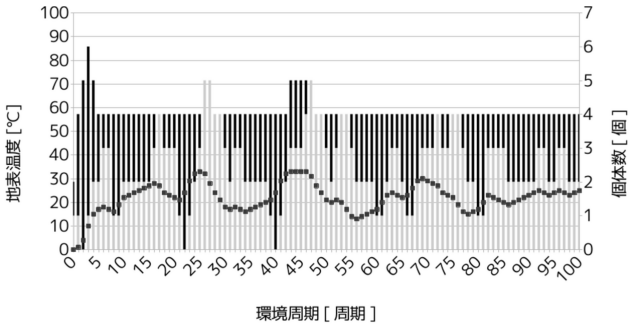
以上から、PoC 実装上で今回の実験設定においては、誤差は九割方で 20 ℃程に抑えられており、地表温度を概ね 0 ℃〜40 ℃辺りで調整できると言えそうです。

良く調整できた場合でも生育適温から 10 ℃もブレる可能性があるというのは、植物としては辛そうな感じです・・。これに関してはやはりデイジー 1 個で 1 ℃も温度が増減するというのが大きすぎたかと思います。

ただ、あくまでも今回の実験の目的は「バイナリ生物学の仕組みの上で自己制御のようなものが働くことができるかどうかを確認すること」であり、それについては「確認できた」と言えるかと思います。

♣ 白/黒のデイジーの個体数の推移

白と黒のデイジーの個体数の推移がどうなっていたのかも確認しておきます。前項で紹介した 90 パーセンタイル値が最も低かった実験結果の際に、白と黒のデイジーの個体数の推移がどうなっていたかを見てみます。前項の図 3.5 に個体数の推移のグラフを重ねると図 3.7 の通りです。



▲図 3.7: デイジーが存在する場合の地表温度と個体数の推移

追加されている積み上げグラフが個体数のグラフで、灰色が白いデイジーの個体数を、黒色が黒いデイジーの個体数を表しています。

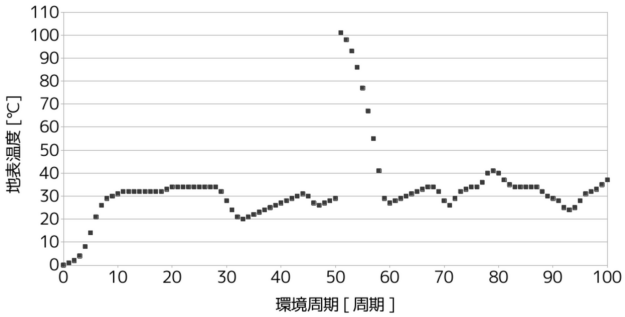
図 3.7 から、まず全体の個体数は 4 前後であったことが分かります。これは実験設定で増殖コストを高くしたためで、意図通りに個体数を低く抑えられていることが分かります。

また、最初の環境周期 0 から 5 辺りや 60 から 65 辺りを見ると、地表温度が生育適温より低い場合に黒いデイジーが増えて温度を上げていることが確認できます。そして、環境周期 25 から 30 辺りや 45 から 50 辺りを見ると、地表温度が生育適温より高い場合には逆に白いデイジーが増えて温度を下げていることが確認できます。

なお、65 周期目辺りに関しては、生育適温に達した後も黒いデイジーが増えてしまい生育適温を超えて温度が上昇してしまっています。ただ、これに関しては、突然変異はランダムに起こるものなので仕方ない所かなと思います。同時にこのような不安定な所が「生物的」な所であるように感じます。それ故に 1 個体の環境への影響力は低くした方が良かったことも分かります・・。

♣ 他にも外乱を与えた場合

追加の実験として他にも外乱を与えてみました。試しに環境周期 50 で地表温度を 100 ℃にしてみた結果は図 3.8 の通りでした。



▲図 3.8: 環境周期 50 で地表温度を 100 ℃にした場合

途中で突然地表温度が急激に変化しても、それを生育適温へ戻そうとする働きが確認できます。この結果からも、バイナリ生物学の仕組みの上で自己制御のようなものが働くことができると言えるかと思います。

おわりに

ここまで読んでいただきありがとうございます！ 本書ではバイナリ生物学の理論を PDP-7 という 60 年代のコンピュータでも動作するくらいシンプルにした設計とその PoC 実装による実験について紹介しました。「バイナリ生物学」自体、突飛なものであるかと思いますが、本書を通してその考え方や仕組みについて少しでも面白いと感じていただければ幸いです。

本書で設計のシンプルさにこだわった理由としては、「UNIX が生まれた PDP-7 でも動く程にしたかった」事に加えて、「ライフゲームに影響を受けていた」事もあります。ライフゲームは「生命の誕生や死を計算機上でシミュレーションするゲーム」で、碁盤の目状のマスで構成された世界の上でたった数個のルールでそのようなシミュレーションを行います。[*8]

ライフゲームについて私がすごいと思うところは、少ないルールで「確かにそれっぽく」見える所です。生物学的に厳密に考えると「生命の誕生や死」というものは数個のルールで網羅できているものでは無いとは思いますが、「生物の密度が誕生や死に関わる」という観点においてはそれっぽくシミュレートできているのかと思います。本書で紹介した設計は、そのような事を思いながら、「バイナリが生き物のように進化していく上で必要最低限の仕組みは何か」という観点でこれまでのバイナリ生物学の設計を見直した結果のものです。

ライフゲームのすごさとしては他にも、これだけシンプルであるからこそ実装が手軽で動作にコンピュータの性能を要するものでもないといった所があります。それもあって「自分でも実装してみたい」・「ルールを追加したり変更したりして試してみたい」という思いを起こさせる力があると思っています。本書のバイナリ生物学の設計のシンプルさは、ライフゲームには及びませんが、60 年代のコンピュータでも動くことは (一応) 示せたかと思います。本書を読んで、PDP-7 でなくて構いませんので、お好きな環境で「自分でもバイナリ生物学を実装してみたい」と思っていただけたら幸いです。

[*8] ref.http://math.shinshu-u.ac.jp/~hanaki/lifegame/

よりシンプルなバイナリ生物学の本

よりシンプルなバイナリ生物学の設計と
PoC 実装によるデイジーワールド実験を紹介！

2023 年 11 月 11 日　ver 1.0 （技術書典 15）

著　者　大神祐真
発行者　大神祐真
連絡先　yuma@ohgami.jp
　　http://yuma.ohgami.jp
　　@yohgami (https://twitter.com/yohgami)
印刷所　日光企画

(powered by Re:VIEW Starter)