数理の窓

コードを自己最適するゲームプログラム

ファミコンの初期ROM（read only memory）カートリッジの容量はわずか40kB、本体のメモリにいたっては4kBしかなかった。この中にプログラムやキャラクターデザイン、マップ、サウンドデータをすべて詰めこむため、書ける命令コードは数千行程度に限られ、1クロック・1バイトを削る最適化が死活問題となった。

例えば、初期値（乱数シード）さえ決めておけば、乱数生成により同じマップ構造が再現できるため、マップデータそのものは不要となる。三角関数値をテーブルとして用意すれば、実行時の計算負荷を削減できるし、重複部分をサブルーチン化すれば、同じ命令を何度も書かずに済む。そしてより革新的なのは「自己修正コード」である。これは、プログラムが実行中に自身の命令コードやデータを書き換え、機能や実行内容を切り替える技法だ。たとえば、起動直後にだけ使う初期化コードを一度実行したら単なるNOP（何もしない命令）に置き換える、ループで毎回計算していた数値を使い回すために実行後に結果の参照命令へ差し替えるのだ。

ポイントは“プログラム”と“データ”の境界を曖昧にしながら、必要な情報処理だけを最小限の形で保持し続ける発想だ。通常は「プログラムは固定で、入力データが変化する」と考えるが、自己修正コードでは、プログラム自身も動的に書き換えられるデータとして扱われる。結果として、同じ情報量をコードで保持するか、データで保持するか、実行結果をコードに埋めこむかの選択肢が生まれ、いわば“動的最適化”がプログラム自身で行われる。これはメタプログラムともいえる。

もっとも、一度でも書き換えに失敗すれば、システム全体の挙動を破綻させかねないため、綿密な事前テストとデバッグが必須だった。ファミコンの場合は、電源を落とせばROMから再ロードされるという“ロールバック機能”が暗黙の安全策になっていたが、プログラマーは試行錯誤を重ね、限界まで最適化を突き詰めたのである。

現代では、CPUもメモリも飛躍的に向上し、これほどの節約や動的最適化はさほど必要ではなくなった。しかし、自己修正コードの概念は難読化やフォールトトレランス、進化的プログラミングなどで息づいており、“プログラムは固定的ではなく、柔軟に動的な修正が可能”というアイデアがもとになっている。

もし宇宙が壮大な“自己修正コード”であり、星々や銀河の衝突によるエネルギーと情報のやり取りがプログラムとデータの動的最適化だとするなら、ファミコンの1クロック・1バイトを削る技術に、宇宙の謎を読み解くヒントが潜んでいるかもしれない。

（外園 康智）